Chapter 4
Neural Networks in System Identification

Gábor HORVÁTH
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Magyar tudósok körútja 2, 1521 Budapest, Hungary

Abstract. System identification is an important way of investigating and understanding the world around. Identification is a process of deriving a mathematical model of a predefined part of the world, using observations. There are several different approaches of system identification, and these approaches utilize different forms of knowledge about the system. When only input-output observations are used behavioral or black box model can be constructed. In black box modeling neural networks play an important role. The purpose of this paper is to give an overview of the application of neural networks in system identification. It defines the task of system identification, shows the basic questions and introduces the different approaches can be applied. It deals with the basic neural network architectures, the capability of neural networks and shows the motivations why neural networks are applied in system identification. The paper presents the main steps of neural identification and details the most important special problems, which must be solved when neural networks are used in system modeling. The general statements are illustrated by a real world complex industrial application example, where important practical questions and the strength and weakness of neural identification are also discussed.

4.1. Introduction

System identification is the process of deriving a mathematical model of a system using observed data. Modeling is an essentially important way of exploring, studying and understanding the world around. A model is a formal description of a system, which is a separated part of the world. A model describes certain essential aspects of a system.

In system modeling three main principles have to be considered. These are separation, selection and parsimony.

The world around is a collection of objects, which are in interactions with each other: the operation of one object may have influence on the behavior of others. In modeling we have to separate one part of the world from all the rest. This part is called the system to be modeled. *Separation* means that the boundaries which separate the system from its environment have to be defined

The second key principle is *selection*. Selection means that in modeling only some essential aspects of a system are considered. There are many different interactions between the parts of a system and between the system and its environment. However, in a modeling task all interactions cannot be considered. Some types of interactions have to be taken into account while others must be neglected. The selection of the aspects to be considered depends on the final goal of modeling. Some aspects are important and must be represented in one case, while entirely different aspects are to be represented in another case, even if the system is the same. This means that a model is always imperfect, it is a simplified

representation of a system, it only approximates a system. This approximation may be better, more accurate in certain aspects and less accurate in others. Because of this simplification, to work with models is always easier than to work with real systems. However, it also means that the validity of the results obtained using a model of a system is always limited.

The third principle is *parsimony*. In model building many different models can be built using the same observations and all these models can be consistent with the observed data. Some guiding principle has to be used to select one model from the possible ones. Parsimony means that in modeling it is always desirable to use as simple model as possible. Parsimony principle is formulated as the Occam's razor: *The most likely hypothesis is the simplest one that is consistent with all observations.* Or in other words: *The simpler of two theories, two models (when both are consistent with the observed data) is to be preferred.*

4.2. The main steps of modeling

In every modeling task the following main steps can be distinguished:

- collection of prior information,
- selection of model set, model structure,
- experiment design and data collection,
- model parameter estimation,
- model validation.

The role of these steps in the whole identification process is depicted in Figure 1.

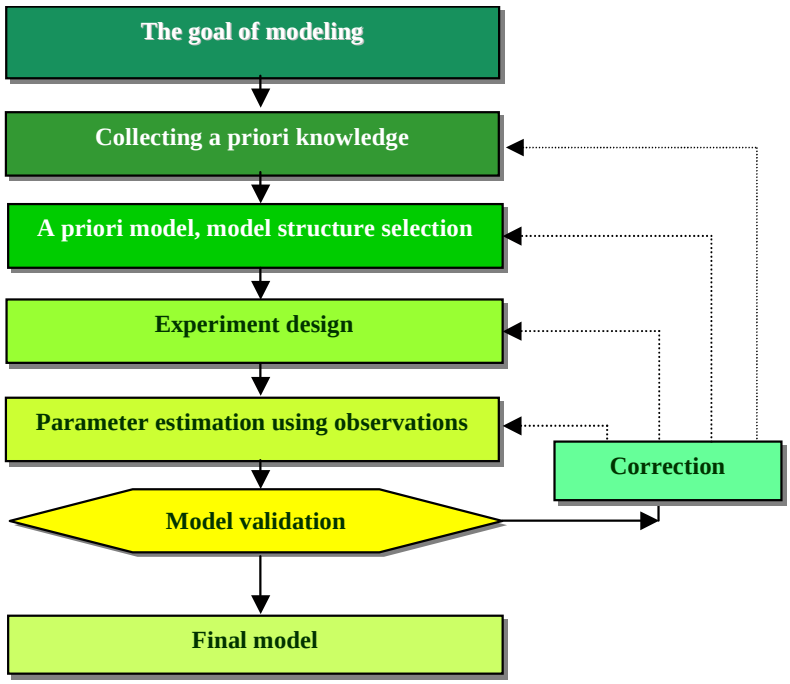

Figure 1: System identification as an iterative process.

In system identification many different approaches can be applied depending on the prior information available, the goal of modeling, what part of the world has to be modeled and what aspects are to be considered, etc.

Model set selection means that the relation between inputs and outputs of a system is formulated in a general mathematical form. This mathematical form defines the structure of the model and defines a set of parameters, the values of which have to be determined during the identification process.

Model classes can be categorized in different ways depending on the aspects taken into consideration.

Based on the *system characteristics* we can distinguish between

- static or dynamic,
- deterministic or stochastic,
- continuous-time or discrete-time,
- lumped parameter or distributed parameter,
- linear or non-linear,
- time invariant or time variant, etc.

models.

All these differentiations are important for the further steps of the whole identification process.

Independently from the previous aspect we can build *parametric* or *nonparametric* models.

In parametric models a definite model structure is selected and only a limited number of parameters must be estimated using observations. In many cases there are some physical insight about the system, we know what important parts of the system can be distinguished, how these parts are connected, etc, so we know the structure of the model. In these cases physical models can be built. Physical models are typical parametric models, where the structure of the model is determined using physical insight.

In nonparametric models there is no definite model structure and the system's behavior is described by the response of the system for special excitation signal. Nonparametric models can be built if we have less knowledge about the system. Typical nonparametric description of a system is the impulse response or the frequency characteristics.

4.2.1 Model set selection

Model set selection is basically determined by the available information. The more information is available the better model can be constructed and the more similarity will be between the system and its model. Based on *prior information* we can speak about white box, grey box or black box models.

When both the structure and the parameters of the model are completely known – complete physical knowledge is available - we have a white box model. White box models can be constructed from the prior information without the need of any observations.

When the model construction is based only on observed data, we speak about input-output or behavioral model. An input-output model is often called empirical or *black box* model as the system to be modeled is considered as a black box, which is characterized with its input-output behavior without any detailed information about its structure. In black box modeling the model structure does not reflect the structure of the physical system, thus the elements of the model structure have no physical meaning. Instead, such model structure has to be chosen that is flexible enough to represent a large class of systems.

Of course the white box and the black box models represent extremes. Models actually employed usually lie somewhere in between. In most of the identification tasks we have certain physical information, however this is not complete (incomplete theoretical knowledge). We can construct a model, the structure of which is selected using available physical insight, so the structure of the model will correspond to that of the physical system. The parameters of the model, however, are not known or only partly known, and they must be estimated from observed data. The model will be fitted empirically using observations. Physical modeling is a typical example of grey-box modeling. The more complete the physical insight the "lighter" grey box model can be obtained and vice verse. The "darkness" of model depends on the missing and known information as shown in Figure 2.

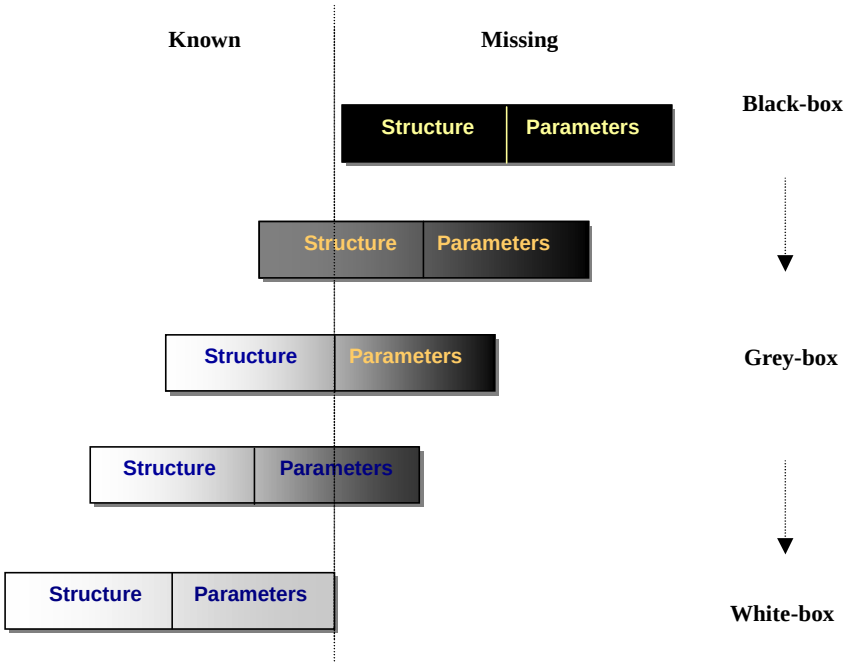

Figure 2: Model categories based on prior information.

The approach used in modeling depends not only on prior information, but the complexity of the modeling procedure and the goal of modeling as well. As building black box models may be much simpler than physical modeling, it is used not only when the lack of physical insight does not let us build physical models, but also in such cases when we have enough physical knowledge, but it is too complex, there are mathematical difficulties, the cost of building physical models is too high, etc.

In black box modeling – contrary to physical ones – the model structure is not determined entirely by selecting the model class. We have to determine the size of the structure, the number of model parameters (e.g., in a polynomial model class the maximum order of the polynomial, etc.). To determine the proper size of the model and the numerical values of the parameters additional information about the system have to be used. This additional information can be obtained from observations. For collecting observations we have to design experiments, to design input signals, and measure the output signals as responses for these input ones.

4.2.2 Experiment design

Experiment design has an important role of getting relevant observations. In the step of experiment design the circumstances of input–output data collection is determined, the excitation signals are designed. The construction of excitation signal depends on the prior knowledge about the system. For example, different excitation signals have to be used to identify a linear and a non-linear system; the excitation depends on whether the system is static or dynamic, deterministic or stochastic, etc. In non-linear system identification the selection of excitation signal depends on the required validity range of the model. Different excitations can be used if model validity is required only for the neighborhood of an operating point or if such model is needed that reflects some important aspects of the system in many different operating points, etc.

In general we have to select input signals that will excite the system in such a way that the input–output data can be observed during the experiment carry enough knowledge about the system. In system identification it is often required to design new and significantly modified experiments during the identification process, where the knowledge collected from the previous experiments are utilized.

In many cases experiment design means to determine what signals can be measured at all, so this step depends largely on the practical identification task. In some identification problems there is no possibility to design excitation, we can only measure the input and output data available in normal operating conditions. This situation may happen when experiment design would be too expensive or when the system to be modeled is an autonomous one, which operates without explicit input signals, etc.

The general and special questions of experiment design are beyond the scope of this paper, interested readers can consult relevant books, e.g. [1,2].

4.2.3 Model parameter estimation

Model set selection means that the relation between inputs and outputs of a system is formulated in a general mathematical form. This mathematical form defines the structure of the model and defines a set of parameters, the values of which have to be determined during the further steps of the identification process. In the sequel we assume that the system implements an $f : R^N \rightarrow R$ mapping, however the scalar output is used only for simplicity. This mapping is represented by a set of input-output measurement data $\{\mathbf{x}(i), y(i)\}_{i=1}^{P}$.

The relation between the input and output measurement data can be described as

$$y(i) = f(\mathbf{x}(i)) + n(i) \tag{1}$$

where $n(i)$ is the observation noise.

This system will be modeled by a general model structure. The mapping of the model $\hat{f}$ will approximate in some sense the mapping of the system.

$$y_M(i) = \hat{f}(\mathbf{x}(i), \boldsymbol{\Theta}) \tag{2}$$

The model also implements an $R^N \rightarrow R$ mapping; y_M is the output of the model and $\boldsymbol{\Theta}$ is the parameter vector of the model structure.

Having selected a parametrized model class, the parameters of the model have to be determined. There are well-developed methods, which give estimates for the numerical values of the parameters. These parameter estimation methods utilize different types of knowledge available about the system to be modeled. We may have prior information about the nature of the parameters to be determined (e.g., we may have physical knowledge about the possible range of certain parameters, we may have information if some parameters are deterministic ones or can be considered as random variables with known probability distribution, etc.), but the essential part of the knowledge used for parameter estimation is a set of measurement data, a set of observations $\{\mathbf{x}(i), y(i)\}_{i=1}^{P}$ about the system.

Parameter estimation is a way of adjusting the model parameters for fitting the observations according to some criterion function. The parameter estimation process is shown in Figure 3.

Depending on the criterion function (which also may depend on the prior information about our system) we can speak about least square (LS) estimation, weighted least square (WLS) estimation, maximum likelihood (ML) estimation or Bayes estimation.

A criterion function is a measure of the quality of the model, it is a function of the error between the model output y_M and the system output y:

$$C(\boldsymbol{\Theta}) = C\left(y - y_M\left(\boldsymbol{\Theta}, Z^P\right)\right) \tag{3}$$

where Z^P denotes the set of measured data pairs

$$Z^P = \{\mathbf{x}(i), y(i)\}_{i=1}^{P} \tag{4}$$

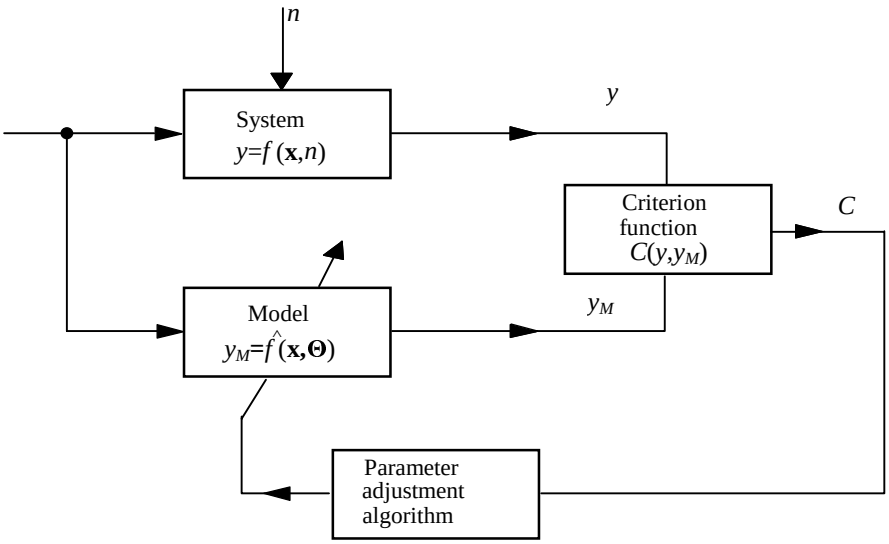

Figure 3: The parameter estimation process.

If both model structure and model size are fixed, model parameters have to be estimated. In parameter estimation the selection of criterion function mainly depends on the prior information. The most common measure of discrepancy is the sum of squared error,

$$C(\boldsymbol{\Theta}) = \frac{1}{2}\boldsymbol{\varepsilon}(\boldsymbol{\Theta})^T \boldsymbol{\varepsilon}(\boldsymbol{\Theta}) = \frac{1}{2}\sum_{i=1}^{P}(\varepsilon(i))^2 = \frac{1}{2}\sum_{i=1}^{P}(y(i) - y_M(i))^2 \tag{5}$$

or the average of the squared error between the model outputs and the observations, which is often called empirical risk:

$$C_{emp}(\boldsymbol{\Theta}) = \frac{1}{P}\sum_{i=1}^{P}(y(i) - y_M(i))^2 \tag{6}$$

i.e., usually quadratic criterion functions are used.

Quadratic criterion function can always be applied, because it requires only the observed input – output data of the system and the output data of the model for the known input data. The parameter estimate based on this quadratic criterion function is the least square estimate:

$$\boldsymbol{\Theta}_{LS} = \arg\min_{\boldsymbol{\Theta}} C(\boldsymbol{\Theta}) \tag{7}$$

The observations are noisy measurements, so if something is known about the statistical properties of the measurement noise some statistical estimation can be applied. One of the most common statistical estimations is maximum likelihood (ML) estimation, when we select the estimate, which makes the given observations most probable.

$$\boldsymbol{\Theta}_{ML} = \arg\max_{\boldsymbol{\Theta}} f(\mathbf{y}_P | \mathbf{x}_P, \boldsymbol{\Theta}) \tag{8}$$

where $f(\mathbf{y}_P | \mathbf{x}_P, \boldsymbol{\Theta})$ denotes the conditional probability density function of the observations. The maximum likelihood estimate is illustrated in Figure 4.

If the parameter to be estimated is a random variable and if its probability density function is known, we can apply Bayes estimation. Although Bayes estimation has certain optimality property, it is rarely applied because it requires more prior information than ML or LS estimations.

There is no place to discuss the classical estimation methods in detail. There are many excellent books and papers dealing with the classical system identification methods; they

give detailed discussion of parameter estimation methods as well, especially for linear dynamic systems, see e.g. [1-7].

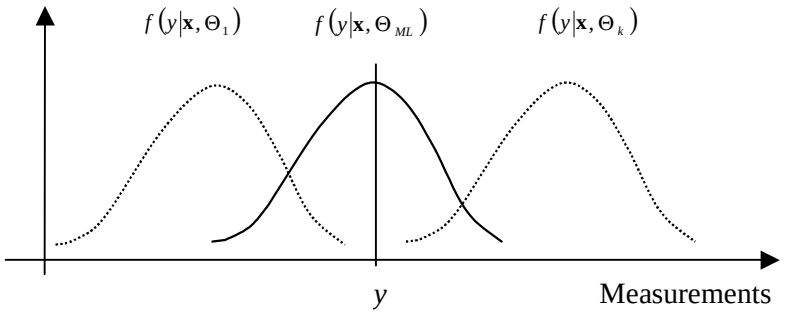

Figure 4: Maximum likelihood estimation.

4.2.4 Model validation

The final step of system identification is the *validation* of the model. For validation a proper criterion as a fitness of the model must be used. The choice of this criterion is extremely important as it determines a measure of the quality of the model. From the result of the validation we can decide whether or not the model is good enough for our purpose. If not, an iterative cycle of structure selection (model class and model size selection), experiment design, parameter estimation and model evaluation must be repeated until a suitable representation is found; so system identification is an iterative process.

4.3. Black box model structures

When we have no prior information about the system to build physical model, black box modeling approach can be used. In black box modeling a general model structure must be selected, which is flexible enough to build models for a wide range of different systems. In this paper we assume that the input – output mapping of the system to be modeled can be described by a continuous function $y=f(\mathbf{x})$. However, as the function is unknown we try to build a model solely on observations about the system's behavior. In a practical black box modeling problem we can observe noisy measurements, where the relation between the measured input and output data can be described again as before:

$$y(i) = f(\mathbf{x}(i)) + n(i) \tag{9}$$

From this point of view black box identification is similar to the general identification case, except that there is no other knowledge about the system than the observations:

$$Z^P = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}_{i=1}^{P} \tag{10}$$

A black box model will give a relationship between the observed inputs and outputs. The mapping of the model can be described as $y_M(i) = \hat{f}(\mathbf{x}(i), \boldsymbol{\Theta})$, i=1,2,...P, where $\boldsymbol{\Theta}$ is the parameter vector of the model.

There are several different forms of this relationship, however, a general form can be described as a weighted sum of given basis functions $\{G_j(.)\}_{j=1}^{M}$:

$$y_M(k) = \sum_{j=1}^{M} \alpha_j G_j(\mathbf{x}(k)) \tag{11}$$

where the parameter vector is defined as $\boldsymbol{\Theta} = [\alpha_1 \alpha_2 \ldots \alpha_n]^T$.

There are many possible basis function sets, which can be applied successfully in system identification (nonlinear function approximation). For example, we can form polynomial functions, when the mapping of the system is approximated by a polynomial, or we can use complex exponentials, which means, that the mapping of the system is approximated by a Fourier series. But Taylor expansion, wavelets or Volterra series can also be applied. Among the black box structures neural networks play an important role.

The selection between the possibilities usually based on prior information about the system, or on some general (theoretical or practical) advantages or drawbacks of the different black box architectures.

Having selected a basis function set two problems must be solved: (i) how many basis functions are required in this representation, and (ii) how the parameters of the model can be estimated. The first question belongs to the model selection problem, the selection of the size of the model, while the second question is a parameter estimation problem.

The answers to these questions can be divided into two groups. There are general solutions, which are valid for all black box modeling approaches, and there are special results which apply only for a given black box architecture. The general answers are related mainly to the model size problem, while for the parameter estimation task different methods have been developed for the different black box architectures. Most of these methods are discussed in detail in the basic literature of system identification, here only such methods will be presented that are directly related to neural modeling.

The next sections give an overview of neural networks, presents the most important neural architectures and the most important features of the neural paradigm, it shows why neural networks are important in system modeling. The special problems, difficulties in neural modeling and possible solutions to avoid these difficulties will also be discussed.

4.4. Neural networks

Neural networks are distributed information processing systems made up of a great number of highly interconnected identical or similar simple processing units, which are doing local processing, and are arranged in ordered topology. An important feature of these networks is their adaptive nature, which means that its knowledge is acquired from its environment through an adaptive process called learning. The construction of neural networks uses this iterative process instead of applying the conventional construction steps (e.g., programming) of a computing device. The roots of neural networks are in neurobiology; most of the neural network architectures mimic biological neural networks, however in engineering applications this neurobiological origin has only a limited importance and limited effects.

In neural networks several slightly different elementary neurons are used, however, the neural networks used for system modeling usually apply two basic processing elements. The first one is the perceptron and the second is the basis function neuron.

The *perceptron* is a nonlinear model of a neuron. This simple neural model consists of two basic parts: a linear combiner and a nonlinear activation function. The linear combiner computes the scalar product of the input vector $\mathbf{x}$ of the neuron and a parameter vector (weight vector) $\mathbf{w}$:

$$s = \sum_{i=0}^{N} w_i x_i = \mathbf{w}^T \mathbf{x} \tag{12}$$

Every element of the weight vector determines the strength of the connection from the corresponding input. As $x_0=1$, w_0 serves as a bias value. The bias has the effect of increasing or decreasing the input signal level of the activation function depending on its sign. The nonlinear activation function is applied for the output of the linear combiner. It is

responsible for the nonlinear behavior of the neuron model. The mapping of the elementary neuron is:

$$y=g(s)=g(\mathbf{w}^T\mathbf{x}) \tag{13}$$

where $g(.)$ denotes a the nonlinear activation function. In most cases the activation function is a monotonously increasing smooth squashing function, as it limits the permissible amplitude range of the output to some final value. The typical activation functions belong to the family of the sigmoidal functions. The most common elements of this family are the logistic function, $y=\mathrm{sgm}(s)=\frac{1}{1+e^{-s}}$ and the hyperbolic tangent function, $y=\tanh(s)=\frac{1-e^{-2s}}{1+e^{-2s}}$.

A *basis function neuron* receives simultaneously all components of the *N*-dimensional real-valued input vector **x**, then applies a nonlinear basis function on it. The mapping of a basis function neuron usually depends on one or more parameters. The general form of this mapping is given by:

$$y=g(\mathbf{x})=g\|\mathbf{x}-\mathbf{c}\| \tag{14}$$

where $g(.)$ is a nonlinear basis function and **c** is a parameter of the basis function. Typical basis functions are the radially symmetric functions, like a Gaussian function, where **c** is a centre parameter. In Gaussian basis functions there is another parameter, the width •, as a Gaussian function is given by:

$$g_i(\mathbf{x})=\exp\left[-\|\mathbf{x}-\mathbf{c}_i\|^2/2\sigma_i^2\right] \tag{15}$$

Both neuron types can be used in many, different neural architectures. Here only such architectures will be discussed which can be used for system modeling.

For constructing a neural network first its architecture must be selected, than the free parameters of the architecture must be determined. To select the architecture we must determine what type and how many elementary neurons are to be used and how they should be organized into a certain - usually regular – structure. The values of the free parameters can be determined using the networks' adaptive nature, their learning capability.

System identification usually means identification of dynamic systems, so when dealing with neural architectures the emphasis will be on dynamic neural networks. However, as dynamic networks are based on static ones, first a short overview of the basic static neural architectures will be given.

For presenting the most important dynamic neural structures two different approaches will be followed. We will begin with the classical dynamic neural architectures, then a general approach will be shown, where the nonlinear dynamic mapping is represented as a nonlinear function of a regressor vector. Using this approach, which has been introduced in linear dynamic system identification, we can define important basic nonlinear dynamic model classes.

4.5. Static neural network architectures

The most common neural architecture is the multi-layer perceptron (MLP). An MLP is a feed-forward network built up of perceptron-type neurons, arranged in layers. An MLP has an input layer, one or more hidden layers and an output layer. In Figure 5 a single hidden layer multi-input – multi-output MLP is shown. An MLP is a fully connected network,

which means that every node (neuron) in each layer of the network is connected to every other neuron in the adjacent forward layer. The *k*-th output of a single hidden layer MLP can be written as:

$$y_k = g\left(\sum_{j=0}^{N_2} w_{kj}^{(2)} g\left(\sum_{i=0}^{N_1} w_{ji}^{(1)} x_i \right) \right) \tag{16}$$

Here $w_{kj}^{(l)}$ denotes a weight of the MLP, which belongs to the *k*-th neuron in layer *l* and which is connected to the *j*-th neuron's output of the previous layer. The *g*(.)-s in Eq. (16) stand for the activation functions. In the figure $\mathbf{w}^{(l)}$ contains all weights of layer *l*.

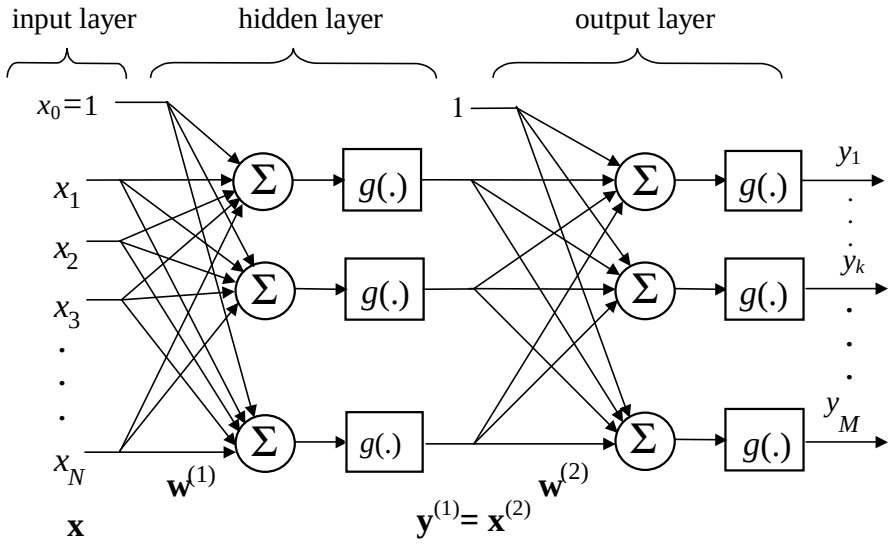

Figure 5: A multi-layer perceptron with one hidden layer.

Perhaps the most important question arising about MLPs is its computational or modeling capability. Concerning this question the main result is that a one hidden-layer feed-forward MLP with sufficient number of hidden processing elements of sigmoidal type, and a single linear output neuron is capable of approximating any continuous function f: $R^N \rightarrow R$ to any desired accuracy.

There are several slightly different mathematical results formulating the universal approximation capability, the most important of which were developed by Hornik [8], Cybenko [9], Funahashi [10], Leshno et al. [11], etc. Here only the result of Cybenko will be cited:

Let *g* be any continuous sigmoid-type function, then give any continuous real-valued function *f* on $[0,1]^N$ or any other compact subset of R^N and $\varepsilon > 0$, there exist vectors $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$, and a parametrized function $\hat{f}(\mathbf{x}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}) : [0,1]^N \bullet R$ such that

$$\left| f(\mathbf{x}) - \hat{f}(\mathbf{x}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}) \right| < \varepsilon \quad \text{for all } \mathbf{x} \in [0,1]^N \tag{17}$$

where

$$\hat{f}(\mathbf{x}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}) = \sum_{j=1}^{M} w_j^{(2)} g\left(\mathbf{x}^T \mathbf{w}_j^{(1)}\right) = \sum_{j=1}^{M} w_j^{(2)} g\left(\sum_{i=0}^{N} x_i w_{ij}^{(1)} \right) \tag{18}$$

In Eq. (18) $\mathbf{w}^{(1)} = \left[\mathbf{w}_1^{(1)}, \mathbf{w}_2^{(1)}, ..., \mathbf{w}_M^{(1)}\right]$ is the weight vector of the first computing layer (what is usually called hidden layer), where $\mathbf{w}_j^{(1)} \in R^{N+1}$ j=1,2,…M, is the weight vector of

the *j*-th hidden neuron, x_0=1 as defined earlier and $\mathbf{w}^{(2)} = \left\lfloor w_1^{(2)}, w_2^{(2)}, ..., w_M^{(2)} \right\rfloor$ is the weight vector of the linear output layer.

This theorem states, that for getting an MLP with universal approximation property only the hidden neurons must be nonlinear, the output neuron may be a simple linear combiner. Moreover it states that one hidden layer is enough. In spite of this result in practical applications two or more hidden layers are often used, as an MLP with more hidden layers may have certain advantages. Such advantage may be, that the total number of neurons is less using more hidden layers or the training of the network, the estimation of its parameters may be easier.

An MLP with one hidden layer can be represented as a weighted sum of some nonlinear basis functions. The general architecture of these networks is depicted in Figure 6.

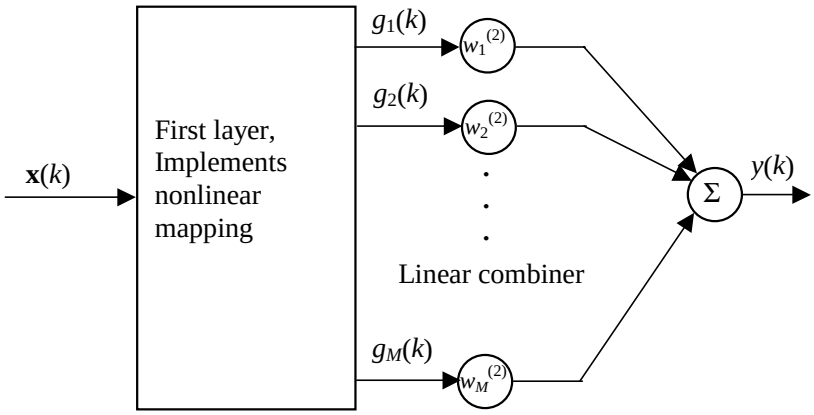

Figure 6: General network with nonlinear hidden layer and linear output layer.

The network has two computing layers: the first one is responsible for an $R^N \to R^M$ non-linear mapping, which results in an intermediate vector $\mathbf{g}(k) = [g_1(k), g_2(k), ..., g_M(k)]^T$. The elements of this intermediate vector are the responses of the basis functions. The output of the mapping is then taken to be a linear combination of the basis functions.

In an MLP the basis functions are parametrized sigmoidal functions where the parameters are the weight values of the hidden layer. So a single hidden layer MLP has two parameter sets: $\mathbf{w}^{(1)}$ consists of all weights of the hidden layer and $\mathbf{w}^{(2)}$ is formed from the weights of the output linear combiner.

There are several further neural network architectures, which also implement weighted sum of basis functions, but where these basis functions are not sigmoidal ones.

When radial basis functions are used the *Radial Basis Function* (RBF) neural network is obtained, but the *Cerebellar Model Articulation Controller* (CMAC) [12] and the *Functional Link Network* (FLN) [13] or the *Polynomial Neural Network* (PNN) [14], etc. are also elements of the two-computing-layer networks, where nonlinear mapping is implemented only in the first (hidden) layer.

Perhaps the most important member of this family and the second most popular network architecture behind MLP is RBF. In an RBF network all neurons of the first computing layer simultaneously receive the *N*-dimensional real-valued input vector **x,** so this layer consists of basis function neurons**.** The outputs of these neurons are not calculated using the weighted-sum/sigmoidal activation mechanism as in an MLP. The output of each hidden basis function neuron is obtained by calculating the "closeness" of the input **x** to an *N*-dimensional parameter vector $\mathbf{c}_j$ associated to the *j*-th hidden unit. The response of the *j*-th hidden element is given by:

$$g_j(\mathbf{x}) = g\left(\left\|\mathbf{x} - \mathbf{c}_j\right\|\right) \tag{19}$$

Typical radial basis functions are the Gaussian functions of Eq. (15) where the $\mathbf{c}_i$ vectors are properly selected centres and the σ_i values are the width parameters of the basis functions. The centres are all different for the different hidden neurons, the width parameters may be different, but often a common width parameter σ is used for all basis functions. A Gaussian function is a local basis function where its locality is determined by the width parameter.

The RBF networks – similarly to the MLPs – are also universal approximators [15], where the degree of accuracy can be controlled by three parameters: the number of basis functions used, their location (the centre parameters) and their width. Because of the similar modeling capabilities of MLPs and RBFs, they are alternative neural architectures in black box system identification.

Besides their similarities these two architectures differ from each other in several aspects. These differences - although do not influence their essential modeling capability - may be important from practical point of view. One architecture may require smaller number of nodes and parameters than the other; there may be significant differences between the learning speed of the two architectures, etc. However, all these differences can be considered as technical ones; their detailed discussion is beyond the scope of this paper. Interested readers can consult some excellent books, e.g. [16,17].

CMAC is also a feed-forward network with similar capability. It uses hidden units with local basis functions of predefined-positions. In the simplest case, in binary CMAC [12] - finite support rectangular basis functions are used, but higher-order CMACs can also be defined, when higher order basic splines are applied as local basis functions [17]. The modeling capability of a CMAC is slightly inferior to that of an MLP [18,19] (a binary CMAC implements a piecewise linear mapping, and only higher order CMACs can implement continuous input-output mapping), but it has significant implementation advantages especially when embedded hardware solutions are required [20].

4.6. Dynamic neural architectures

The basic neural network architectures presented in the previous section all implement static nonlinear mapping between their inputs and output,

$$y_M(k) = f(\mathbf{x}(k)) \tag{20}$$

that is the output at a discrete time step k depends only on the input at the same time instant. Static networks can be applied for static nonlinear system modeling.

In black box system identification, however, the really important task is to build models for dynamic systems. In dynamic systems the output at a given time instant depends not only on its current inputs, but on the previous behavior of the system. Dynamic systems are systems with memory.

4.6.1 Extensions to dynamic neural architectures

There are several ways to form dynamic neural networks using static neurons, however in all ways we use storage elements and/or apply feedback. Both approaches can result in several different dynamic neural network architectures.

Storage elements can be used in different parts of a static network. For example, some storage modules can be associated with each neuron, with the inputs or with any intermediate nodes of a static network. As an example a feed-forward dynamic network can be constructed from a static multi-input – single-output network (e.g., from an MLP or RBF) if a tapped delay line is added as shown in Figure 7. This means that the static network is extended by an embedded memory, which stores the past values of the inputs.

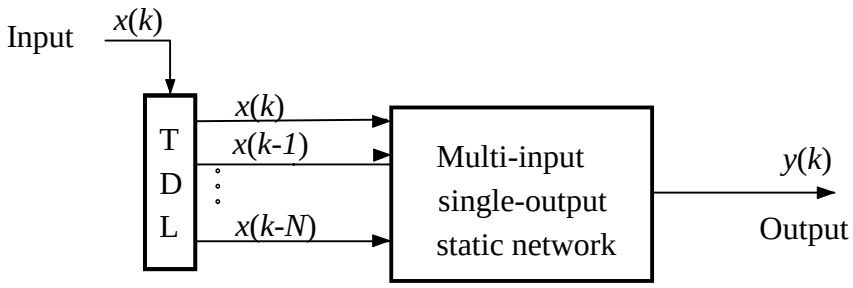

Figure 7. Feed-forward dynamic neural network architecture.

Tapped delay lines can be used not only in the input signal path, but at the intermediate nodes of the network or in the output signal path.

A feed-forward dynamic neural architecture can also be obtained if tapped delay lines are applied for the inputs of all neurons, that is all weights of a static network are replaced by linear filters. If finite impulse response (FIR) filters are used, the resulted dynamic architecture is the FIR-MLP, which is shown in Figure 8.

The output of the *i*-th neuron in layer *l* is given as:

$$y_i^{(l)}(k) = g\left(\sum_j z_{ij}^{(l)}(k) = \right) = g\left(\sum_j \mathbf{w}_{ij}^{(l)T}\mathbf{x}_j^{(l)}(k)\right) \tag{21}$$

where $\mathbf{w}_{ij}^{(l)} = \left[w_{ij,0}^{(l)}, w_{ij,1}^{(l)}, ..., w_{ij,M_i^{(l)}}^{(l)}\right]^T$ is the *j*-th filter coefficient vector of node *i* in layer *l*, the elements of which are associated with the corresponding taps of the FIR filter. The input vector of this filter is formed from the delayed outputs of the *j*-th neuron of the previous layer:

$$\mathbf{y}_j^{(l-1)}(k) = \mathbf{x}_j^{(l)}(k) = \left[x_j^{(l)}(k), x_j^{(l)}(k-1), ..., x_j^{(l)}\left(k - M_i^{(l)}\right)\right]^T \tag{22}$$

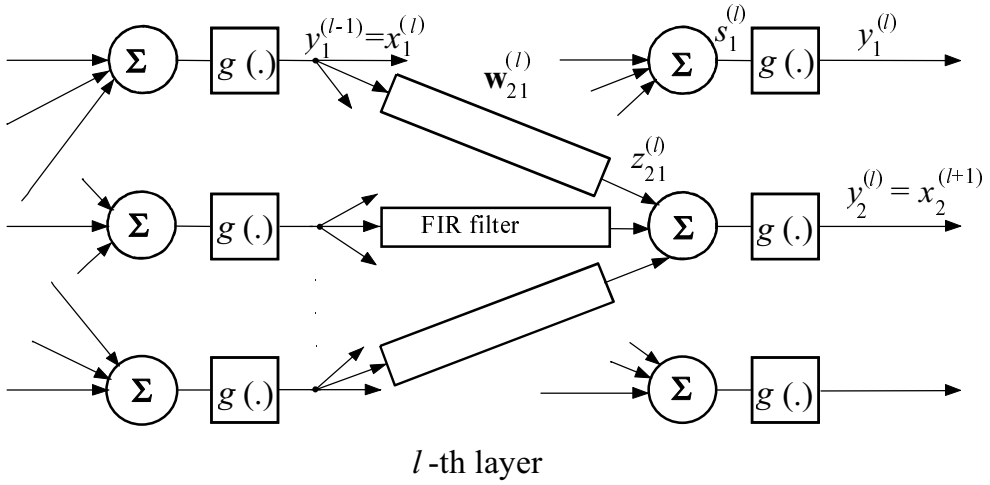

Figure 8. FIR-MLP feed-forward neural network architecture.

If the tapped delay line is used in the output signal path, a feedback architecture can be constructed, where the inputs or some of the inputs of a feed-forward network consist of delayed outputs of the network. The resulted network is a recurrent one. A possible architecture where tapped delay lines are used both in the input and in the output signal paths is shown in Figure. 9.

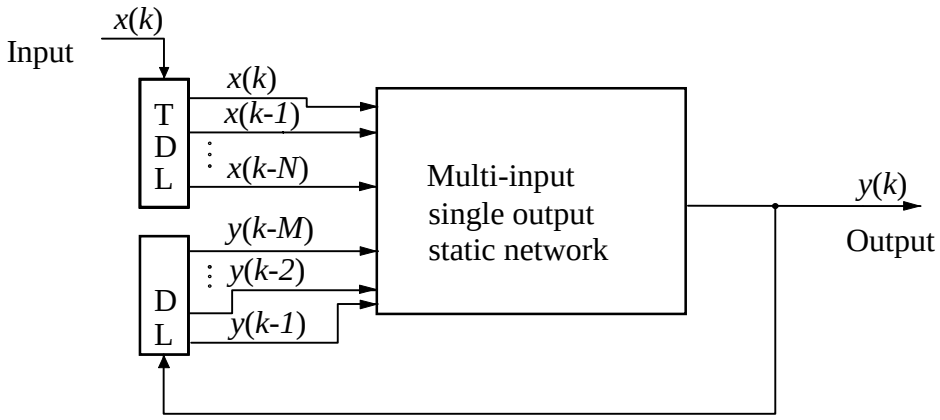

Figure 9: A dynamic neural architecture with feedback.

These dynamic neural networks are general dynamic nonlinear modeling architectures as they are based on static networks with universal approximation property. In these architectures dynamics is introduced into the network using past values of the system inputs, of the intermediate signals and/or of the outputs.

The structure in Figure 9 applies global feedback from the output to the input. However, dynamic behavior can also be obtained if local feedback is used. In this case not the network's output but the output of one or more neuron(s) are applied as inputs of either the same or different neurons. Some possibilities are shown in Figure 10. Such typical dynamic neural architectures are the Jordan and the Elman network [21].

A further possibility to construct dynamic neural network is to combine static neural networks and dynamic linear networks. Within this approach both feed-forward and feedback architectures can be defined as proposed by Narendra [22]. In Figure 11 some combined architectures are shown. In the figure N stands for static neural networks, while $H(z)$ denotes linear dynamic systems.

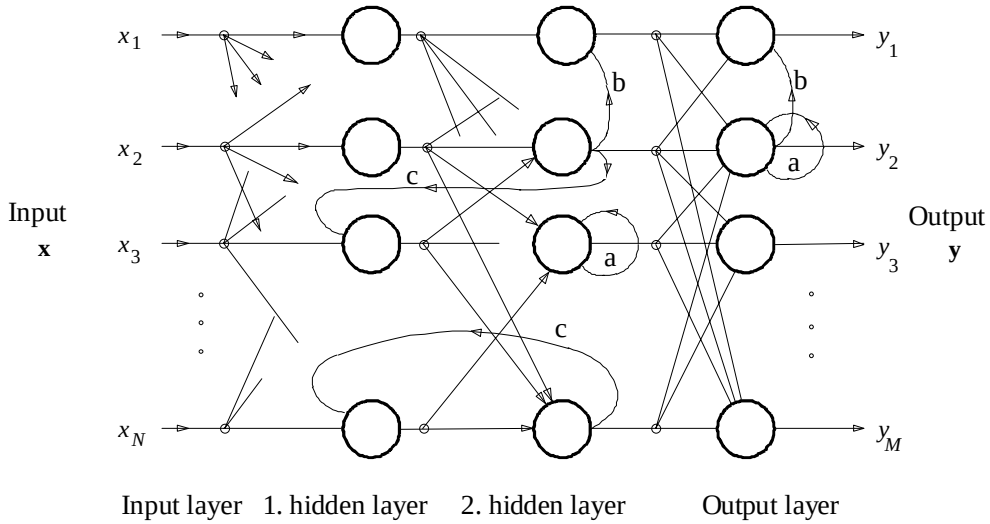

Figure 10: Dynamic neural architecture with local feedback.

The model of Figure 11 a.) is also known as Hammerstein model, while the model of b.) as Hammerstein-Wiener model [2]. Similarly to the Hammerstein model a Wiener model can be constructed where the order of the static nonlinear part and the dynamic linear part is changed over. Also there is a model structure called Wiener – Hammerstein model, which is similar to model b.) except that a static nonlinear system is placed between two linear dynamic ones.

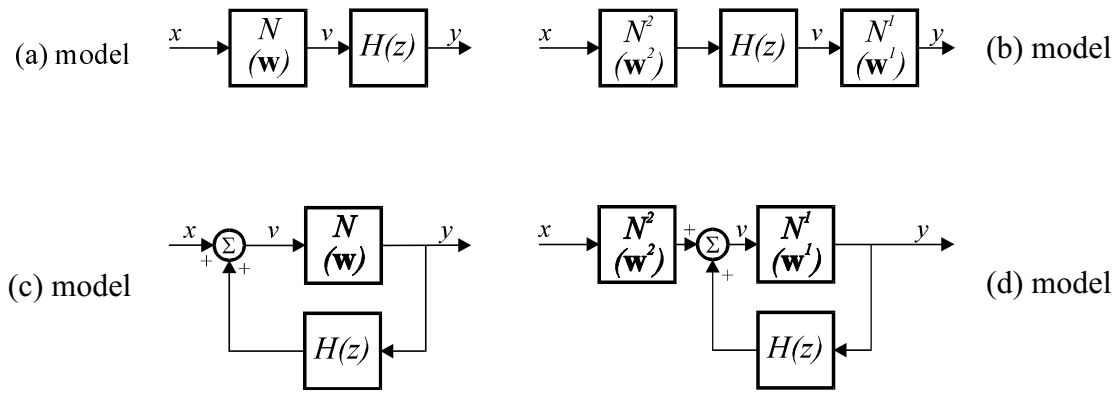

Figure 11: Combined dynamic neural architectures.

4.6.2 General dynamic model structures

Previously many different dynamic neural network architectures were presented. In nonlinear system identification, however, a much more general approach can be followed. In this approach – similarly to the building of linear dynamic black box models – general nonlinear model structures can be formed.

In these dynamic model structures a regressor vector is used, and the output of the model is described as a parametrized function of this regressor vector [23]:

$$y_M(k) = f(\Theta, \varphi(k)) \tag{23}$$

where Θ is the parameter vector and $\varphi(k)$ denotes the regressor vector.

The regressor can be formed from past inputs, past system outputs, past model outputs etc. according to the model structure selected. The following regressors can be defined:

When only the past inputs are used the regressor is formed as:

$$\varphi(k) = [x(k-1), x(k-2), \ldots, x(k-N)] \tag{24}$$

Based on this regressor a feed-forward nonlinear model structure can be constructed. This model - similarly to its linear counterpart - is called an NFIR model. An NFIR model does not contain feedback so it cannot be unstable using any parameter vector. This is the simplest case of regressor-based architectures.

If both past inputs and system outputs are used in the regressor

$$\varphi(k) = [x(k-1), x(k-2), \ldots, x(k-N), y(k-1), y(k-2), \ldots, y(k-P)] \tag{25}$$

the NARX model can be constructed. This model is often called series-parallel model [22], as it uses a feedback, however this feedback comes from the system's - and not the from the model's output, let us avoid forming a really recurrent model architecture.

The regressor can be formed from the past inputs and past model outputs

$$\varphi(k) = [x(k-1), x(k-2), \ldots, x(k-N), y_M(k-1), y_M(k-2), \ldots, y_M(k-P)] \tag{26}$$

The corresponding structure is the NOE model. In a NOE model there is a feedback from model output to its input, so this is a recurrent network. Sometimes NOE model is called as parallel model [22]. Because of its recurrent architecture serious instability problem may arise, which cannot be easily handled.

In the NARMAX model the past inputs, the past system outputs and the past model outputs are all used. Usually the past model outputs are used to compute the past values of the difference between the outputs of the system and the model,

$$\varepsilon(k-i) = y(k-i) - y_M(k-i), \quad i = 1, 2, \ldots, L \tag{27}$$

so the regressor is as follows:

$$\boldsymbol{\varphi}(k)=[x(k-1),\ldots,x(k-N),y(k-1),\ldots,y(k-P),\varepsilon(k-1),\ldots,\varepsilon(k-L)] \tag{28}$$

The regressor for the NBJ models is formed from past inputs, past model outputs and the past values of two different errors, ε and ε_x. Here ε is defined as before, while ε_x is

$$\varepsilon_x(k-i)=y(k-i)-y_{Mx}(k-i),\quad i=1,2,\ldots,K \tag{29}$$

In this equation $y_{Mx}(k-i)$ is the model output when only the past inputs are used. The corresponding regressor is

$$\boldsymbol{\varphi}(k)=[x(k-1),\ldots,x(k-N),y_M(k-1),\ldots,y_M(k-P),\varepsilon(k-1),\ldots,\varepsilon(k-L),\varepsilon_x(k-1),\ldots,\varepsilon_x(k-K)] \tag{30}$$

Although the definitions of these general model classes are different from the definition of the classical dynamic neural architectures, those structures can be classified according to these general classes. For example, an FIR-MLP is an NFIR network, but the combined models a) and b) in Figure 11 also belong to the NFIR model class, while the neural structure of Figure 9 is a typical NOE model.

The selection of the proper model class for a given identification problem is not an easy task. Prior information about the problem may help in the selection, although these model classes are considered as general black box architectures and black box approach is usually used if no prior information is available.

The general principle of parsimony can also help to select among the several possible model classes. As formulated by the Occam's razor we always have to select the simplest model, which is consistent with the observations. This means that we should start with linear models and only if the modeling accuracy is not good enough we can go further to the more complex NFIR, NARX, NOE, etc., model structures.

The selection of model structure is only the first step of the neural model construction, further important steps are required: to determine the model size and the model parameters. All these steps need the validation of the model, so model class and model size selection as well as the model parameter estimation cannot be done independently from model validation. The question of model size selection will be discussed in the section of model validation, some basic question of parameter estimation – the learning – are the subject of the next section.

4.7. Model parameter estimation, neural network training

In neural networks the estimation of parameters, the determination of the numerical values of the weights is called learning. As it was mentioned, learning is an iterative process, when the weight values of the network are adjusted step by step until we can achieve the best fit between observed data and the model. The learning rules of neural networks can be categorized as *supervised learning*, which is also referred to as learning with a teacher and *unsupervised learning*. In both cases the learning process utilizes the knowledge available in observation data, what is called training data.

4.7.1 Training of static networks

Neural networks used for system modeling are trained with supervised training. In this case the weights of a neural network are modified by applying a set of labeled training samples $Z^P=\{\mathbf{x}^i,y^i\}_{i=1}^P$. Each training sample consists of a unique input $\mathbf{x}(i)$ and a corresponding

desired output $y(i)$. During training every samples are applied to the network: a training sample is selected usually at random from the training set, the input is given to the network and the corresponding response of the network is calculated, then this response, the output of a network is compared to the corresponding desired output. For evaluating the network response, a criterion function is defined which is a function of the difference between the network's output and the desired output.

$$\varepsilon(i) = y(\mathbf{x}(i)) - y_M\left(\mathbf{x}(i), \boldsymbol{\Theta}\right) \tag{31}$$

The network output (and the modeling error too) depends on the network parameters, $\boldsymbol{\Theta}$. Here $\boldsymbol{\Theta}$ consists of all weights of a neural network. Usually a quadratic criterion function is used: the most common measure of discrepancy for neural networks is the squared error

$$C(\boldsymbol{\varepsilon}) = \frac{1}{2}\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} = \frac{1}{2}\sum_{i=1}^{P}\left(\varepsilon(i)\right)^2 \tag{32}$$

so the supervised learning process is an LS estimation process.

The figure of merit can be defined in a more complex way. In addition to the standard quadratic error performance measure, a second term can be added.

$$C_c = C(\boldsymbol{\varepsilon}) + \lambda C_r, \tag{33}$$

where $C(\boldsymbol{\varepsilon})$ is the standard criterion function, C_r is a so called *regularization* term and λ is the regularization parameter, which represents the relative importance of the second term. This approach is based on the regularization theory developed by Tikhonov [24]. The regularization term usually adds some constraint to the optimization process. The constraint may reflect some prior knowledge (e.g., smoothness) about the function approximated by the network, can represent a complexity penalty term, or in some cases it is used to improve the statistical stability of the learning process.

When regularization is used for complexity reduction the regularization term can be defined as the sum of all weights of the network:

$$C_r = \|\mathbf{w}\|^2 = \sum_i w_i^2 \tag{34}$$

Using this term in the criterion function the minimization procedure will force some of the weights of the network to take values close to zero, while permitting other weights to retain their relatively large values. The learning procedure using this penalty term is called *weight-decay procedure*. This is a parametric form of regularization as the regularization term depends on the parameters of the network.

There are other forms of regularization, like

$$C_r(\mathbf{w}) = C(\mathbf{w}) + \lambda\Phi\left(\hat{f}(x)\right) \tag{35}$$

where $\Phi\left(\hat{f}(x)\right)$ is some measure of smoothness. This latter is a typical form of nonparametric reularization. Regularization can often lead to significantly improved network performance.

The performance measure is a function of the network parameters; the optimal weights values of the network are reached when the criterion function has a minimum value. For neural networks used for function approximation the criterion function is a continuous function of the parameter vector, thus it can be interpreted as a continuous error surface in the weight space. From this point of view network training is nothing else than a minimum seeking process, where we are looking for a minimum point of the error surface in the weight space.

The error surface depends on the definition of the criterion function and the neural network architecture. For networks having trainable weights only in the linear output layer

(e.g., networks with architecture shown in Figure 6) and if the sum of squares error is used as criterion, the error surface will be a quadratic function of the weight vector; the error surface will have a general multidimensional parabolic form. In these networks the first layer is responsible for the nonlinear mapping, but this nonlinear mapping has no adjustable parameters. These networks implement nonlinear but *linear-in-the-parameter* mappings. Typical networks with quadratic error surface is an RBF network if the centre and width parameters are fixed, and a CMAC network where there is no trainable parameter in the first nonlinear layer.

The consequence of the parabolic error surface is that there will be a single minimum, which can be located using rather simple ways. For a quadratic error surface analytic solution can be obtained, however even for such cases usually iterative algorithms, e.g., gradient search methods are used. In gradient-based learning algorithms first the gradient of the error surface at a given weight vector should be determined, then the weight vector is modified in the direction of the negative gradient:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu \nabla(k) \tag{36}$$

Here $\nabla(k)$ is the gradient of the error surface at the k-th iteration, μ is a parameter called learning rate, which determines the size of the step done in the direction of the negative gradient.

Eq. (36) is a general form of the gradient algorithm. For networks with one trainable layer the gradient can be computed directly, however for networks with more than one trainable layer the gradient calculation needs to propagate the error back, as the criterion function gives errors only at the outputs. Such networks, like MLPs require this error back propagation process. The result is the *error backpropagation learning algorithm*, which calculates the gradients using the chain rule of derivative calculus. Because of the need of propagating the error back to the hidden layers, the training of a multi-layer network may be rather computation intensive.

Moreover, the error function for networks with more than one trainable layer may be highly nonlinear and there may exist many minima in the error surface. These networks – like MLPs – implement nonlinear mappings, which are at least partly *nonlinear-in-the-parameter* mappings. Among minima there may be one or more for which the value of the error is the smallest, this is (these are) the global minimum (minima); all the other minimum points are called local minima. For nonlinear-in-the-parameter error surfaces we cannot find general closed form solutions. Instead iterative – usually gradient based – methods are used. Although an iterative, gradient-based algorithm does not guarantee that the global minimum will be reached, the learning rules applied for nonlinear-in-the-parameter neural networks usually are also gradient based algorithms.

A more general gradient-based learning rule can be written as:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \mathbf{Q}(-\nabla(k)) \tag{37}$$

where $\mathbf{Q}$ is a matrix, which modifies the search direction and which usually reflects some knowledge about the error surface.

Several different gradient rules can be derived from this general one if we specify $\mathbf{Q}$. If $\mathbf{Q} = \mathbf{I}$ the identity matrix, we can get the steepest descent algorithm (Eq. 36). With $\mathbf{Q} = \mathbf{H}^{-1}$ and $\mu = 1/2$ the Newton algorithm is obtained, where $\mathbf{H}^{-1}$ is the inverse of the Hessian of the criterion function. The Hessian matrix is defined by

$$\mathbf{H} = \nabla \nabla C(\varepsilon) = \left[\frac{\partial^2 C}{\partial w_i \partial w_j} \right] \tag{38}$$

From the general form of the gradient learning rule the Levenberg-Marquardt rule [16] can also be obtained. In this case an approximation of the Hessian is applied to reduce the

computational complexity. The different gradient-based algorithms can reach the minimum using less learning iterations, however, one iteration requires more complex computations than the simple steepest descent method.

4.7.2 Training of dynamic networks

The learning rules discussed so far can be applied for static neural networks. For training dynamic networks some additional problems must be solved. Dynamic networks are sequential networks, which means that they implement nonlinear mapping between input- and output data sequences. So the training samples of input–output data pairs of static networks are replaced by input–output data sequences and the goal of the training is to reduce a squared error derived from the elements of the corresponding error sequences. If $\varepsilon(k)$ is the output error of a dynamic network at discrete time step *k*, the squared total error can be defined as:

$$\varepsilon_{total} = \sum_{k=1}^{K} \varepsilon^2(k) \tag{39}$$

where *K* denotes the length of the sequence.

Dynamic networks have memory, and this needs significant modification of the training algorithms. The basic training rules for dynamic systems are also gradient-based algorithms. A common feature of these learning rules is, that - instead of modifying the weights in every step when a new sample is used (as it is usually done in static networks) - the weights are modified only after a whole training sequence were applied to the network. This will let the network be unchanged during a whole training data sequence is applied. The most important family of learning rules appropriate for dynamic networks is called *dynamic backpropagation*.

For training dynamic networks different versions of the dynamic backpropagation have been developed [22]. For feed-forward networks a possible approach is to unfold the network in time. This strategy first removes all time delays in the network by expanding it into an equivalent static network. However, the resulted static network will be much larger, moreover several weights of the extended static network represent actually the same weights, which must be updated in an equivalent way. For feed-forward networks unfolding in time is effective only if tapped delay lines are short. A more efficient learning for such NFIR network as an FIR-MLP (shown in Figure 8) is the temporal backpropagation [25].

For recurrent networks two different approaches are applied most often. The first one uses also unfolding in time, which means that a recurrent dynamic network is transformed into a corresponding feed-forward static one. This transformation maps the neurons with their states at every time step into a new layer, where the number of resulting layers is equal to the length of the unfolding time interval. In the unfolded network all weights of the original recurrent network are repeated in every layer. The resulted static network can be trained by standard backpropagation rule, except that these weights are physically identical and should be modified by the same value in one training step. The unfolding-in-time approach is called backpropagation through time (BPTT) [26].

BPTT can be explained most easily in an example. Figure 12 a.) shows a simple recurrent network with only two neurons. Suppose that a four-length input sequence is used, the corresponding unfolded feed-forward static network is shown in Figure 12 b.). The two networks are equivalent for these four steps, however, we have to care that the weight with the same indexes are identical, they exist only ones, although several copies of the weight are drawn in the unfolded version. Unfolding-in-time is a rather simple way of handling recurrent networks, however it is effective only if the time interval is small.

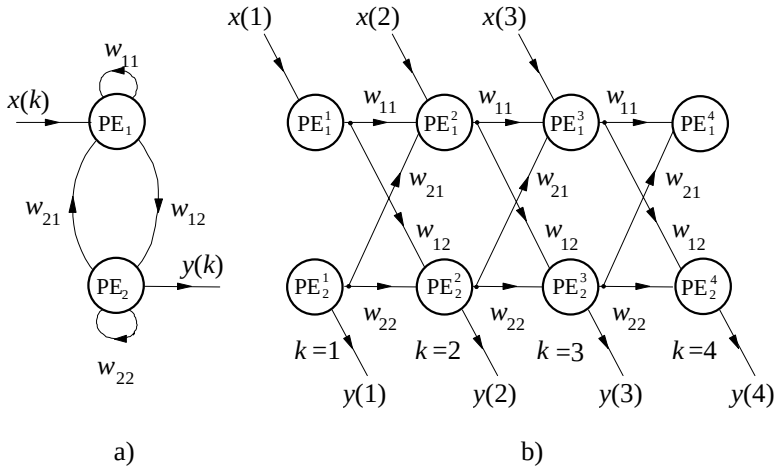

Figure 12: Unfolding-in-time for a simple recurrent network
a) original recurrent network, b) unfolded static feed-forward network

Another method to train a recurrent network is the real-time recurrent learning (RTRL), where the evolution of the gradient over time steps can be written in recursive form [27]. In RTRL the weights are modified in every time steps. This violates the requirement of updating the weights only after a whole training sequence was applied, however, it was found that updating the weights after each time step works well as long as the learning rate μ is kept sufficiently small. Sufficiently small learning rate means that the time scale of the weight changes is much smaller then the time scale of the network operation. Real time recurrent learning avoids the need for allocating memory proportional to the maximum sequence length and leads to rather simple implementations.

During training all training data are usually used many times. The number of training cycles may be quite large, and it is important to find when to stop training. To determine the optimal stopping time the performance of the network must be checked, the network must be validated. So validation helps not only to determine the proper complexity of the network as was indicated before, it is also used to decide whether we have to stop training at a given training cycle.

4.8. Model validation

The goal of the application of neural networks in system identification is to build a black box model of a system using training data. However, this goal is not reached if the model represents the system only at the training points, we need to build an accurate model of the system in the whole operating range of interest. An important feature of a model is that it can approximate well the behavior of a system not only at the training points, but in the whole operating range. This feature is called *generalization*. A neural network without any generalization can only memorize the training points, so it works as a simple lookup table.

Validation is used to estimate the performance of the model, to check its generalization capability. Validation serves several sub-goals. There are validation methods to check if model complexity was selected properly, and there are validation methods what can be used in the learning phase. However, these two sub-goals cannot be reached separately. Usually only a trained model can be validated, which means, that the adequacy of the selected model class and model size can be determined only after model parameters are also determined.

A model of proper complexity is used if both the model class (NFIR, NARX, … etc.) and the model size (model order, the number of free parameters) are chosen appropriately. A proper model class can be selected either using prior knowledge about the system, or -

according to the principle of parsimony - we have to select as simple model class as possible. For model size selection there are general validation methods used in linear or nonlinear system identification and there are special ones developed for neural networks. To check if the network is trained well, several different validation methods are used. Among them there are methods, which are used for both purposes: to check model complexity and check model parameters.

It is well known, that the more complex model is used the better approximation can be reached at the training points. The reason is that increasing the number of the parameters the degree of freedom will be increased, which means that we can adjust the model parameters to fit the training data more. However, reducing the training error does not reduce necessarily the error at different points obtained from the same problem, but not used in training, so reducing the training error does not mean to get better generalization. For checking the generalization capability of the model we need a set of test data from the same problem, a test set, which is not used in training. Using different data sets for constructing a model and for validating it is an important principle. The validation method based on this principle is called *cross-validation* and it has a distinguished role in neural modeling. The effect of model complexity on the performance of the model can be followed in Figure 13.

It shows the training and test errors versus model complexity. The performance is measured as usual, e.g., they are the sum of the squared errors at all training-points and at all test-points, respectively. It can be seen that as model complexity increases first both the training and the test errors decrease. This behavior can be found until we reach a given complexity. From this point the lowering of the training error goes on, while test error is getting larger. A model of optimal complexity, a model with the best generalization property is obtained at the minimum point of the test error.

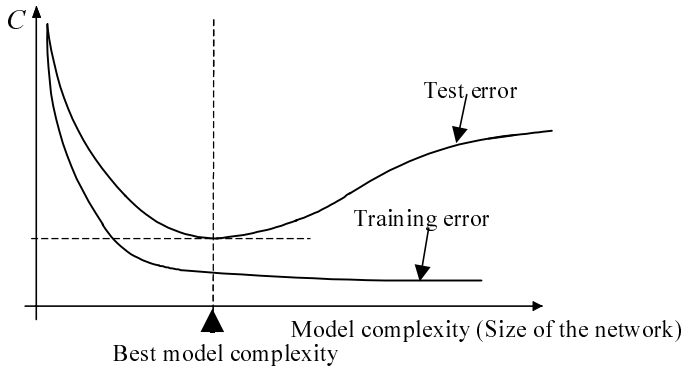

Figure 13. Training and test error versus model complexity

The question of optimal model complexity can be discussed from another point of view. This is the *bias-variance trade-off*. The significance of bias-variance trade-off can be shown if the modeling error is decomposed into a bias and a variance term. As it was defined by Eq. (5), the modeling error is the sum of the squared errors or the average of the squared error

$$MSE_{emp}(\boldsymbol{\Theta}) = \frac{1}{P}\sum_{k=1}^{P}\left(\varepsilon(k)\right)^2 = \frac{1}{P}\sum_{k=1}^{P}\left(y(k) - y_M(k)\right)^2 \tag{40}$$

where $\varepsilon(k)$ can be written in a more general form

$$\varepsilon(k) = y(k) - y_M\left(\boldsymbol{\varphi}(k), \boldsymbol{\Theta}\right) \tag{41}$$

This error definition is valid for all model structures: if $\boldsymbol{\varphi}(k) = \mathbf{x}(k)$ we will have a static model and if $\boldsymbol{\varphi}(k)$ is one of the regressors defined in section 6, it refers to the error of a dynamic network.

Now, consider the limit in which the number of training data samples goes to infinity, the average of the squared error approximates the mean square error, the expected value of the squared error, where expectation is taken over the whole data set.

$$MSE(\boldsymbol{\Theta}) = E\left\{\left(y - y_M(\boldsymbol{\Theta})\right)^2\right\} \tag{42}$$

This expression can be decomposed as:

$$MSE(\boldsymbol{\Theta}) = E\left\{\left(y - y_M(\boldsymbol{\Theta})\right)^2\right\} = E\left\{\left(y_M(\boldsymbol{\Theta}) - E\{y_M(\boldsymbol{\Theta})\}\right)^2\right\} + \left(y - E\{y_M(\boldsymbol{\Theta})\}\right)^2 \tag{43}$$

Here the first term is the variance and the second one is the squared bias.

$$MSE\left(y_M(\boldsymbol{\Theta})\right) = \mathrm{var}\left(y_M(\boldsymbol{\Theta})\right) + \mathrm{bias}^2\left(y_M(\boldsymbol{\Theta})\right) \tag{44}$$

The size of the model, the model order will have an effect on the bias-variance trade-off. A small model with fewer than enough free parameters will not have enough complexity to represent the variability of the system's mapping, the bias will generally be high, while the variance is small. A model with too many parameters can fit all training data perfectly, even if they are noisy. In this case the bias term vanishes or at least decreases, but the variance will be significant. (Figure 14.)

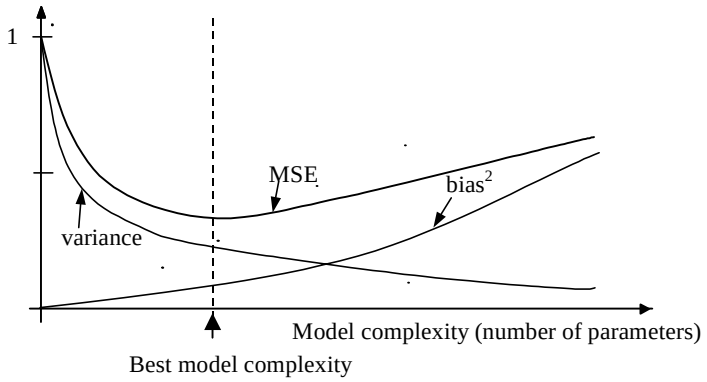

Figure 14: Illustration to the bias-variance trade-off.

In static neural models the model complexity can be adjusted by the number of the hidden neurons. In dynamic models, however this question is more complex. First a proper size of the selected model class must be determined, e.g., for an NFIR architecture we have to select the length of the tapped delay line, or for a NARX or a NARMAX model the lengths of the corresponding tapped delay lines, etc., then the number of hidden neurons which implement the nonlinear mapping have to be determined. Moreover, it can be shown that the selection of the proper model complexity cannot be done independently from the number of available training data samples. There must be some balance between model complexity and the number of training data. The less training points are used, the less knowledge is available from the system and the less free parameters can be used to get a model of good generalization. Of course model complexity must reflect the complexity of the system, more complex systems need more data, which allows building more complex models: models with more parameters.

The question of model complexity versus number of training points and model performance (generalization capability) has been studied from different points of view. One early result for static neural networks gives an upper bound of MSE as a function of the smoothness of the mapping to be approximated, the complexity of the network and the number of training points [28].

$$MSE \leq O\left(\frac{C_f^2}{M}\right) + O\left(\frac{MN}{P}\log P\right) \tag{45}$$

where C_f is a measure of smoothness or regularity of the function f, M is the number of hidden neurons, and N is the size of the dimension of the input data.

Another approach is used by the *statistical learning theory* [29] where an upper bound on the generalization error can be derived. For regression problems MSE bounded with probability of at least (1-η) as:

$$MSE(\mathbf{\Theta}) \leq \frac{MSE_{\text{emp}}(\mathbf{\Theta})}{1 - c\sqrt{v(h)}} \tag{46}$$

Here

$$v(h) = v\left(\frac{P}{h}, \frac{-\ln \eta}{P}\right) \tag{47}$$

where h is the VC-dimension. VC-dimension is a characteristic parameter of the function set used in the approximation. For the validity of Eq. (46) we need that the probability of observing large values of the error is small [30]. It can be proved that models with good generalization property can be obtained only if h is finite [29]. The generalization bound of Eq. (46) is particularly important for model selection, since it provides an upper limit for complexity for a given sample size P and confidence level η.

4.8.1 Model order selection for dynamic networks

For dynamic systems modeling proper model order selection is especially important. As the correct model order is often not known a priori it makes sense to postulate several different model orders. Based on these, some criterion can be computed that indicated which model order to choose. One intuitive approach would be to construct models of increasing order until the computed squared error reaches a minimum. However as it was shown previously the training error decreases monotonically with increasing model order. Thus, the training error alone might not be sufficient to indicate when to terminate the search for the proper model complexity; model complexity must be penalized to avoid using too complex model structures.

Based on this approach several general criteria were proposed. The most important ones are the Akaike Information Criteria (AIC) [31] and the Minimum Description Length (MDL) [32], which were developed for linear system modeling. Recently for MLPs a network information criterion (NIC) was proposed by Amari [33], which was derived from AIC. The common feature of these criteria is that they have two terms: the first one depends on the approximation error for the training data (i.e. the empirical error), while the second is a penalty term. This penalty grows with the number of free parameters. Thus, if the model is too simple it will give a large value for the criterion because the residual training error is large, while a too complex model will have a large value for the criterion because the complexity term is large.

The methods based on the different criteria need to build and analyze different models, so these methods are rather computation intensive ones and their applicability is questionable in practical cases. Recently a new heuristic method was proposed for identifying the orders of input-output models for unknown nonlinear dynamic systems [34]. This approach is based on the continuity property of the nonlinear functions, which represent input-output mappings of continuous dynamic systems. The interesting and attractive feature of this approach is that it solely depends on the training data. The model orders can be determined using the following index:

$$q^{(N)} = \left(\prod_{k=1}^{p} \sqrt{N}\, q^{(N)}(k) \right)^{1/p} \tag{48}$$

where $q^{(N)}(k)$ is the k-th largest *Lipschitz quotient* among all $q_{ij}^{(N)}$ ($i \neq j$; $i, j = 1, 2, \ldots, P$) N is the number of input variables and p is a positive number: usually $0.01P$ -$0.02P$. Here the q_{ij} Lipschitz quotient is defined as:

$$q_{ij} = \frac{|y(i) - y(j)|}{|\mathbf{x}(i) - \mathbf{x}(j)|} \tag{49}$$

where the $\{\mathbf{x}(i), y(i)\}$ i=1, 2, … , P pairs are the measured input-output data samples from which the nonlinear function $f(.)$ have to be reconstructed. This index has the property that $q^{(N+1)}$ is very close to $q^{(N)}$, while $q^{(N-1)}$ is much larger than $q^{(N)}$ if N is the optimal number of the input variables, so a typical curve of $q^{(N)}$ versus N has a definite point (N_0) where the decreasing tendency stops and $q^{(N)}$enters a saturated range. For an NFIR model N_0 is the optimal number of input order. Figure 15 (a) shows a typical curve for $q^{(N)}$.

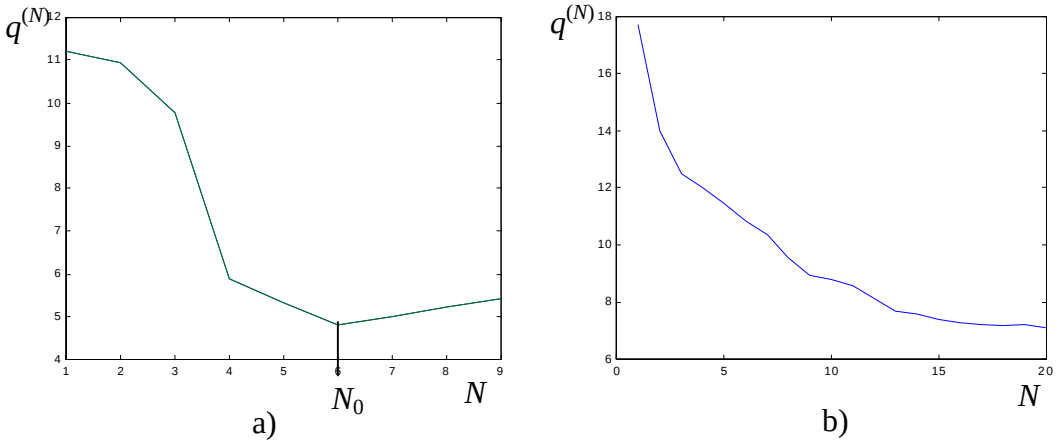

Figure 15: Typical curves of Lipschitz indexes
(a) for noiseless data or data with low noise level, (b) for data with high noise level.

The Lipschitz index can be applied not only for NFIR structures but also for NARX model classes, where two, the order of the feed-forward and the feedback paths must be determined. For NARX model class

$$y_M(k) = f(\boldsymbol{\varphi}(k)) = f[x(k-1), x(k-2), \ldots, x(k-M), y(k-1), y(k-2), \ldots, y(k-L)] \tag{50}$$

the following strategy can be used. The Lipschitz index $q^{(N)} = q^{(L+M)}$ should be computed for different model orders, where L denotes the feedback and M the feed-forward order values. Starting with N=1, where only $y(k\text{-}1)$ is used as input $q^{(1+0)}$ can be computed. Then let N = 2, where the both $x(k\text{-}1)$ and $y(k\text{-}1)$ are used as inputs and $q^{(1+1)}$ can be computed. For N=3 the third input of the dynamic networks will be $y(k\text{-}2)$ and $q^{(2+1)}$ will be computed. This strategy can be followed increasing step by step the feedback and the feed-forward orders. If at a given L and M one can observe that $q^{(L+M)}$ is much smaller than $q^{(L-1+M)}$or $q^{(L+M-1)}$, but is very close to $q^{(L+1+M)}$ or $q^{(L+M+1)}$, we reached the appropriate order values.

The most important advantage of this method is that it can give an estimate of the model order without building and validating different complexity models, so it is a much more efficient way of order estimation then the criteria based approaches. However, there is a significant weakness of the Lipschitz method: it is highly sensitive to observation noise. Using noisy data for model construction - depending on the noise level - we can often get a

typical curve for the Lipschitz index as shown in Figure 15 (b). The most important feature of this figure is that there is no definite break point.

4.8.2 Cross-validation

Modeling error can be used in another way for model validation. This technique is called cross-validation. In *cross-validation* – as it was mentioned before - the available data set is separated into two parts, a training set and a test set. The basic idea of cross-validation is that one part of the available data set is used for model construction and another part for validation. Cross-validation is a standard tool in statistics [35] and can be used both at the model structure selection and at parameter estimation. Here its role in the training process will be presented.

The previous validation techniques for selecting the proper model structure and size are rather complex, computation intensive methods. This is the most important reason why they are applied only rarely in practical neural model construction. The most common practical way of selecting the size of a neural network is the trial and error approach. First a network structure is selected, then the parameters are trained. Cross-validation is used to decide whether or not the performance of the trained network is good enough. Cross-validation, however, is used for another purpose too.

As it was mentioned in the previous section to determine the stopping time of training is rather difficult as a network with quite large number of free parameters can learn the training data almost perfectly. The more training cycles are applied the smaller error can be achieved on the training set. However, small training error does not guarantee good generalization. Generalization capability can be measured using a set of test data consists of samples never seen during training.

Figure 16 shows two learning curves, the learning curves of the training and the test data. It shows, that usually the training error is smaller than the test error, and both curves decrease monotonically with the number of training iterations till a point, from where the learning curve for the test set starts to increase. The phenomenon when the decrease of the training error is going on, while the test error starts to increase is called overlearning or overfitting. In this case the network will memorize the training points more and more while at the test points the network's response is getting worse, we get a network with poor generalization. Overlearning can be avoided if training is stopped at the minimum point of the test learning curve. This is called *early stopping* and it is an effective way to improve the generalization of the network even if its size is larger than required.

For cross-validation we need a training set and a test set of known examples. However there is a question which must be answered: in what ratio, the data points should be divided into training and testing sets in order to obtain the optimum performance. Using statistical theory a definite answer can be given to this question [36]. When the number of network parameters M is large, the best strategy is to use almost all available known examples in the training set and only $\frac{1}{\sqrt{2M}}$ examples in the testing set, e.g., when $M = 100$, this means that only 7% of the training data points are to be used in the test set to determine the point for early stopping. These results were confirmed by large-scale simulations. The results show that when $P > 30M$ cross-validation is not necessary, because the generalization error becomes worse by using test data to obtain adequate stopping time. However, for $P < 30M$, i.e. the number of the known examples is relatively small compared to the number of network parameters, overtraining occurs and using cross-validation and early stopping improves generalization.

Cross-validation can be used not only for finding the optimal stopping point, but to estimate the generalization error of the network too. In network validation several versions

of cross-validation are used. A version called one-leave-out cross-validation is used, especially if the number of known data is small.

The one-leave-out cross-validation is an efficient way of using the examples available. Here we divide the set of examples into two sets as it was proposed before, but only one example will be omitted from the training set and this point will be used for testing. The process will be repeated *P* times, every time a different example is omitted for testing. Such a procedure allows us to use a high proportion of the available data (all but one) to train the network, while also making use of all data points in evaluating the cross-validation error. The disadvantage of this method is that it requires the training process to be repeated *P* times.

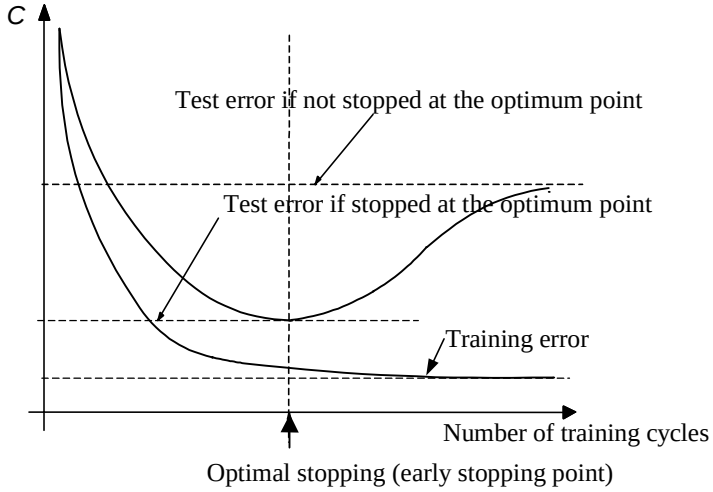

Figure 16: Learning curves for the training and the test data.

4.9. Why neural networks?

In the previous sections we have presented some results, which show that neural networks are general black box structures, so they can be used in black box system identifications. However, using neural networks in system modeling is only one approach among the many available possible ones. There are other black box architectures, and all these architectures can be used to approximate nonlinear mappings of static or dynamic systems, to model nonlinear static or dynamic systems. Moreover, using any of these architectures the steps of model construction are also similar: we have to select a general model structure, a model class, then we have to concretize this model by determining the model size and the model parameters. In all cases the whole process of model building is based on observations and - if any - on prior information.

However, among all these black box architectures neural networks are far the most popular ones. The reasons – at least partly – come from the roots of neural networks: from their neurobiological origin, their ability to learn from examples and from the extremely good problem solving capability of "biological systems", which can be mimicked by artificial neural networks. The historical roots, however, would not be enough for this long time popularity. The real reasons come from the practical advantages of neural modeling.

The application of neural networks has many practical advantages. Among them one can find their relatively simple architecture, their universal approximation capability, etc., but there is an especially important feature of neural networks, mainly MLPs and MLP based dynamic architectures. These networks are not very sensitive to the proper selection of their size; similar performance can be obtained using rather different-size neural models.

In black box modeling to determine the proper size of a model structure is usually a hard task, and choosing improper size often leads to poor models. A too small model is not able to approximate a complex system well enough, a too large model with many free

parameters, however, may be very prone to overfitting. These general statements are more or less valid for all modeling approaches, among them for neural networks. MLPs, however, using backpropagation learning rule have a special feature. They may be biased towards implementing smooth interpolation between the training points, which means that they may have rather limited proneness to overfitting.

The effect of this bias is that even using overly complex neural model, overfitting can be avoided. Backpropagation can result in the underutilization of network resources, mainly at the beginning phase of learning, and this can be definitely observed on the training curves. As it was shown in Figure 16 overlearning can be avoided using early stopping. This behavior of MPLs with backpropagation is justified by extensive experimental studies (e.g., [37]), and by explicit analysis, which shows that neural modeling is often ill conditioned, the efficient number of parameters is much less than the nominal number of the network parameters [38,39].

During learning a network can be forced to reduce the number of efficient parameters using regularization, as it was discussed in section 7. However, for MLPs with backpropagation training an implicit regularization, a regularization effect without using an explicit regularization term can be observed. The resulted smooth mapping is an advantageous feature of neural identification as long as the systems to be modeled are continuous ones. Although this implicit regularization cannot be found in other neural networks, similar properties can be obtained easily using some form of explicit regularization, so some inductive bias that is characterized as smooth interpolation between training points can be found not only in MLPs with backpropagation learning, but in RBF or even in CMAC networks.

4.10. Modeling of a complex industrial process using neural networks: special difficulties and solutions (case study)

In industry many complex modeling problems can be found where exact or even approximate theoretical/mathematical relationship between input and output cannot be formulated. The reasons behind this can be the unsatisfactory knowledge we have about the basic underlying physical behavior, chemical reactions, etc., or the high complexity of the input-output relationship. At the same time there is a possibility to collect observations from the system, we can measure input and output data, so an experimental black box model based on the observations can be constructed.

In the previous sections of this paper many general questions of black box modeling and neural networks were discussed. In this section some practical questions will be addressed through a real-world complex industrial modeling example: modeling of a Linz-Donawitz (LD) steel converter.

4.10.1 LD steel-making

Steel-making with an LD converter is a complex physico-chemical process where many parameters have influences on the quality of the resulted steel [40,41]. The complexity of the whole process and the fact that there are many effects that cannot be taken into consideration make this task difficult. The main features of the process are the followings: a large (~150-ton) converter is filled with waste iron (~30 tons), molten pig iron (~ 110 tons) and many additives, then this fluid compound is blasted through with pure oxygen to oxidize the unwanted contamination (e.g., silicon, most of the carbon, etc.).

At the end of the oxygen blowing the quality of the steel is tested and its temperature is measured. If the main quality parameters and the temperature at the end of the steel-making process are within the acceptable and rather narrow range, the whole process is finished and the slag and the steel is tapped off for further processing.

The quality of the steel is influenced by many parameters, however the amount of oxygen used during blasting is the main parameter that can be controlled to obtain predetermined quality steel. From the point of view of steel-making parameters mean the main features, measurement data of components of the input compounds e.g., mass, temperature and the quality parameters of the pig iron and the waste iron, the mass and some quality parameters of all additives, as well as the amount of oxygen used during the blasting process, etc. It is an important and rather hard task to create a reliable predictor for determining the necessary amount of oxygen. To give a reliable prediction we have to know the relation between the input and the output parameters of the steel-making process, therefore we have to build a model of the steel converter. The inputs of the model are formed by all available observations can be obtained from a charge. The outputs are the most important quality parameters of the steel produced, namely its temperature and the carbon content at the end of the blasting.

To present all details of such a complex modeling task is well beyond the possibilities of this paper, so the goal of this section is not to go into the details, instead to point out that besides the basic tasks of system identification mentioned in the previous sections there are important additional ones which cannot be neglected.

A large part of these additional tasks are related to the database construction.

4.10.2 Data base construction for black box identification

In black box modeling the primary knowledge that can be used for model building is a collection of input–output data. So the first task of modeling is to build a proper data base. One serious problem in real-world tasks is that in many cases the number of available data is limited and rather small.

In steel-making the data base can be built only from measurements and observations done during the regular everyday operation of the converter. Steel-making is a typical example where there is no possibility to design special excitation signals and to design experiments for data collection.

Steel production with an LD-converter is organized in campaigns. During one campaign the production is contiguous and in one campaign about 3000 charges of steel is produced. This means that the maximum number of known examples is limited and it cannot be increased. Moreover, the data base collected in one campaign contains typical and special cases, where the data of special cases cannot be used for modeling because of technological reasons. The ratio of special-to-all cases is rather high, it is around 25-30%. The only possibility to increase the size of the data base is to collect data from more campaigns, however, from campaign to campaign the physical parameters of the steel converter are changing significantly and this changing must be followed by the model as well, so one should take care when and how to use the extended data set.

In forming a proper database the further problems have to be considered:

- the problem of dimensionality,
- the problem of uneven distribution of data,
- the problem of noisy and imprecise data,
- the problem of missing data,
- the effects of the correlation between consecutive data.

The problem of dimensionality if often referred to as the curse of dimension. For neural modeling we need representative data, which cover the whole input space. This means that – depending on the dimension of the input space – a rather large number of training and test patterns is required. If N-dimensional inputs are used and if each input component can take R different values in their validity range the number of all possible input data samples is R^N, so it grows exponentially with the dimensionality of the input space. This means that dimension reduction is an important step, especially when the number of training samples

cannot be increased arbitrarily. To reduce dimension the following two main approaches can be used:

- Applying some mathematical data compression algorithms, like independent component analysis (ICA), principal component analysis (PCA) or factor analysis. The basic thought behind this approach is that the components of the input data vectors are usually correlated, so without significantly reducing their "information content" less new components can be formed from the original ones.
- By analyzing the raw data and using domain knowledge, the rank of importance of the data components can be estimated and the less important components can be omitted.

In some cases the two approaches can be combined: first – using domain knowledge – we can select the most important input parameters, then on the selected data mathematical data compression algorithms can be applied. In the steel-making problem both methods were considered for reducing the dimension of the observed data, however, the reduction based on domain knowledge proved to be more useful. Instead of using all recorded data, only some 20 most important input components of the original ~50-component data records were used during the training.

The importance of the components was determined using detailed analysis of the data and by the results of some preliminary trained networks. These trained networks were used to determine the sensitivity of the model output to the input components. It turned out that there are some components that have very limited effect on the results, so they could be omitted without significant degradation in the performance of the model. The extensive discussions with skilled personnel of the steel factory about the role of the input components have helped us also to select the most important ones.

As a result three major groups were formed. The first group contained measurement data of clearly high importance, such as mass and temperature values, the waiting time between the finishing of a charge and the start of the next one (this waiting time has an effect on the temperature of the converter before filling it with the new workload). The second group contained clearly negligible data, while the third group contained data of questionable importance. The third group was tested by building several neural models based on the same records of the initial data base, but where the input components of the records were different.

Comparing the performances of the trained networks and analyzing the sensitivity of the model outputs to the different input components the most relevant ones were selected. After 5-10 experiments we could reduce the input parameters from the starting 50 to about 20.

Another common feature of industrial problems is that the input data *are typically not uniformly distributed* over their possible ranges. This means that there may have some clusters, and within these clusters quite a lot and representing data points are available, while there may be other parts of the input space from where only a few examples can be collected. For operating modes from where many data can be collected, appropriate models can be constructed, while in underrepresented operating modes the available data are not enough to build proper black box models.

A further problem is that due to the industrial environment the *registered data are frequently inaccurate and unreliable*. Some of the parameters are measured values (e.g., temperature of pig iron), others are estimated values (e.g., the ratio of the different components of the waste iron), where the acceptable ranges of the values are quite large. It is also typical that some measurements are missing from a record. The precision of the values is rather different even in the case of measured data. If wrong or suspicious data are found, or in case of missing data there are two possibilities: either the data can be corrected, or the whole record is cancelled. Correction is preferred, because of the mentioned dimensionality problem. The large dimensionality and the limited number of data examples makes it very important to save as many patterns as possible.

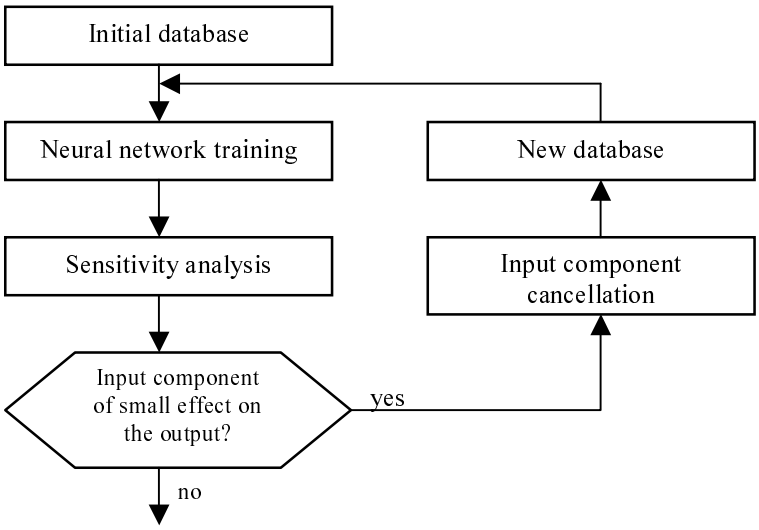

Figure 17: The iterative process of database construction.

Handling of noisy data is a general problem of black box modeling. The methods developed for this problem need some additional information (at least some statistical properties of the measurement noise) and using this additional information a more robust model can be built. Such method is the Errors In Variable (EIV) approach, but Support Vector Machines (SVMs) can also take the noise level into consideration.

The Errors In Variables training method was introduced to reduce the negative effects of measurement noise [42]. The idea behind the method is that knowing some properties of the additive noise, the training process can be modified to compensate the error effects. In EIV approach, instead of the standard quadratic criterion function, a new weighted quadratic criterion function is used, where the weights are the reciprocal values of the variances of the corresponding measurement noise:

$$C_{EIV} = \frac{1}{N} \cdot \sum_{i=1}^{N} \left[\frac{(y(i) - y_M(\boldsymbol{\Theta}, \mathbf{x}^*(i)))^2}{\sigma_{y,i}^2} + \frac{(\mathbf{x}(i) - \mathbf{x}^*(i))}{\sigma_{x,i}^2} \right] \tag{51}$$

In this expression $\{y(i), \mathbf{x}(i)\}$ i=1, 2, … , P denote the measured noisy input-output training examples, $\mathbf{x}^*(i)$ denote the noiseless and naturally not known inputs (during the EIV method estimates of these inputs are also determined), $\sigma_{x,i}^2$ and $\sigma_{y,i}^2$ are the variances of the input and output noise, respectively. The classical LS estimation results in biased estimates of the model parameters, if the input data are noisy. The most attractive feature of the EIV approach is, that it can reduce this bias. This property can be proved if it is applied for training neural networks [43]. The drawback of EIV is its larger computational complexity and the fact that using EIV criterion function the learning process is very prone to overfitting. This latter effect, however, can be avoided using early stopping.

Support Vector Machines are also applies a criterion function that can take the measurement noise into consideration. The criterion function used in SVM is the ε-insensitive function given by Eq. (52).

$$C_{\varepsilon} = \begin{cases} 0 & \text{for} \quad |y - y_M| < \varepsilon \\ |y - y_M| - \varepsilon & \text{otherwise} \end{cases} \tag{52}$$

Using SVMs, the steps of the neural network constructions are rather different from those of the classical neural network approach. An interesting feature of Support Vector Machines is that the size of the model, the model complexity is determined "automatically"

while a network of good generalization can be obtained. Another essential difference between the construction of classical neural networks and SVMs is that no training is used in the classical sense, instead the weights of the networks are determined by a quadratic optimization process. The main disadvantage of SVMs is that this quadratic optimization is a rather time and memory consuming method. For details see e.g. [29].

An important feature of the data base is whether or not the *consecutive records are correlated.* This question is closely related to the model class selection, namely if a static or a dynamic model is to be used, and if dynamic one what regressor should be preferred.

4.10.3 Model class selection

Using the principle of parsimony first static and linear models were used. However, using this simple approach the results were far from satisfactory, more complex model class had to be selected. For model class selection prior physical information has great importance. From physical insight it is almost evident that for this industrial process an adequate model can be achieved only if a dynamic model class is chosen. Using this approach it must be taken into consideration that the output quality parameters of a charge depend not only on the current input parameters, but the current state of the converter (e.g., the end temperature of the steel in one charge will have significant effect on the next charge; there is significant difference between the situations when the starting temperature of the empty converter is around the environment temperature of 0-30 °C or it is around 1000 °C. Surely an LD converter is a system with memory.)

A rather simple but useful way to check if static or dynamic model should be chosen is a simple correlation test. Strong correlation between the data records of consecutive charges is an indication that the system has "memory" and dynamic model must be built. This more sophisticated modeling approach can result in more accurate models than pure static ones, as the production of the consecutive charges is not handled as independent elements of a series of similar events.

Using NARX and NARMAX classes the performance of the model can be increased. For dynamic models, however the model order should also be chosen. In this converter modeling task Lipschitz index was used for finding approximate values of model orders. The results show that a NARX model with orders of (3,3) seems to be the best, where the two order parameters refer to the input and the system output orders (see Eq. (29)). However, because of the noisy measurement data, definite break-point on the Lipschitz curve cannot be found. The brake-point can be sharpened using a combined EIV and Lipschitz method [44], when EIV is used for reducing the effects of measurement noise. Another possibility is to use cross-validation for different-order models around the order obtained from the Lipschitz method.

4.10.4 Modular networks

The experiences gained from this industrial modeling task showed, that using a single neural model satisfactory result cannot be obtained. There may be many different reasons behind this experience. One reason can be found in the special characteristics of the data base. As it was mentioned the known examples can be categorized into at least two groups: typical and special ones. The operation of the converter is different in these two cases, and these differences should be reflected by different models. The solution is to use a modular architecture, which contains more models. The selection of the appropriate one is based on the operating mode. The information about the operating mode of the converter can be obtained from some measurement data or some additional information (e.g., it is known that we are at the beginning, in the middle or near the end of a campaign, there may have

some information, that the blowing process is greatly different from the standard one, the goal parameters are rather special which occurs rarely, etc.).

This type of modular architecture consists of such models from which one and only one is used in a given case. Other modular architectures can also be constructed where different neural models are cooperating. Instead of using a single neural model an ensemble of models can be used.

There are heuristic and mathematical motivations that justify the use ensemble of networks. According to the heuristic explanation combining several different networks can often improve the performance, however, only if the models implemented by the elements of an ensemble are different.

The advantage of using an ensemble of neural networks can also be justified by a simple mathematical analysis [45]. Let us consider the task of modeling a system's mapping f: $R^N \bullet R$. We assume that we can obtain only noisy samples of this mapping and assume that an ensemble of T independent neural models is available. We define a modular architecture using the ensemble of models and the final output of the ensemble is given by a weighted average as:

$$\bar{y}(\mathbf{x},\boldsymbol{\alpha}) = \sum_{j=0}^{T} \alpha_j y_{M_j}(\mathbf{x}) \tag{53}$$

where y_{Mj} is the output of the j-th model. We can define two quality measures, the ambiguity and the squared error for every members of the ensemble and for the whole ensemble. The ambiguity of a single member of the ensemble is

$$a_j(\mathbf{x}) = \left[y_{M_j}(\mathbf{x}) - \bar{y}(\mathbf{x},\alpha)\right]^2 \tag{54}$$

and the ensemble ambiguity is

$$\bar{a}(\mathbf{x},\boldsymbol{\alpha}) = \sum_{j=0}^{T} \alpha_j a_j(\mathbf{x}) \tag{55}$$

This quantifies the disagreement among the models on input **x.** Similarly the quadratic error of model j and the whole ensemble are defined as follows

$$\varepsilon_j(\mathbf{x}) = \left[y(\mathbf{x}) - y_{M_i}(\mathbf{x})\right]^2 \tag{56}$$

and

$$\varepsilon(\mathbf{x}) = \left[y(\mathbf{x}) - \bar{y}(\mathbf{x},\boldsymbol{\alpha})\right]^2 \tag{57}$$

It can be shown easily that the ensemble quadratic error can be written as:

$$\varepsilon(\mathbf{x}) = \bar{\varepsilon}(\mathbf{x},\boldsymbol{\alpha}) - \bar{a}(\mathbf{x},\boldsymbol{\alpha}) \tag{58}$$

if $\sum_{j=1}^{T} \alpha_j = 1$. In Eq. (58) $\bar{\boldsymbol{\varepsilon}}(\mathbf{x},\boldsymbol{\alpha}) = \sum_{j=0}^{T} \alpha_j \boldsymbol{\varepsilon}_j(\mathbf{x})$ is the weighted error and $\bar{a}(\mathbf{x},\boldsymbol{\alpha})$ is the weighted ambiguity of the models as defined by Eq. (55). Eq. (58) shows that the ensemble quadratic error on **x** can be expressed as the difference between the weighted error and the weighted ambiguity. Taking expectations according to the input distribution we can get the average ensemble generalization error

$$\varepsilon = \bar{\varepsilon} - \bar{a} \tag{59}$$

where ε denotes the expected value of $\varepsilon(\mathbf{x})$ and $\bar{a}$ the expected value of $\bar{a}(\mathbf{x})$. This

expression shows that for getting small ensemble generalization error we need accurate and diverse individual models, i.e. they must be as accurate as possible while they must disagree.

The weights of the individual networks in the ensemble can be estimated from the training example too. There are different ways of this estimation: one of the possibilities is to use a *mixture of experts* (MOE) architecture [46], where the α_j weights as well as the weights of the neural networks are estimated using a joint training process and where the results of training are the maximum likelihood estimates of the needed values. The values of the α_j weights depend on the inputs of the models and they are implemented as outputs of an auxiliary network called gating network.

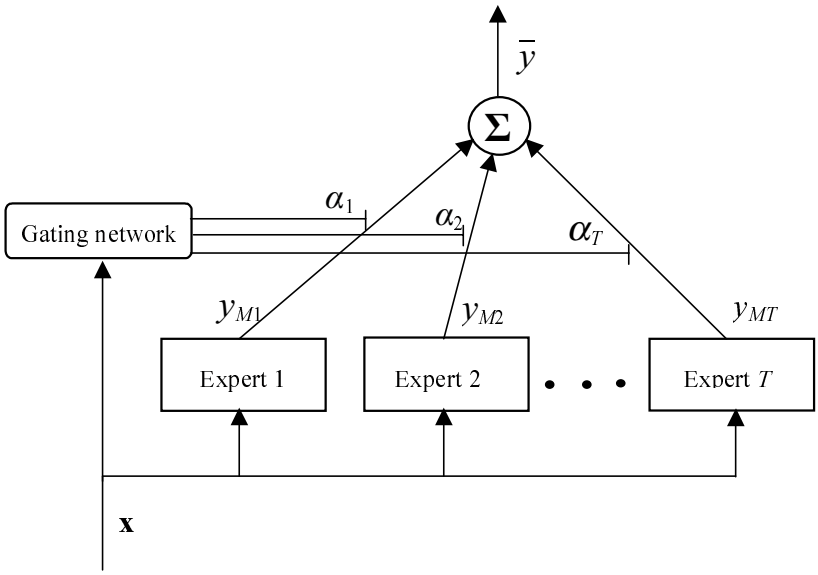

Figure 18: The mixture of experts architecture.

4.10.5 Hybrid models

MOE is a general architecture, where different approaches can be used for implementing the individual experts. Any expert may be a neural model, but any other adaptive or fixed model - like an exact mathematical model, a fuzzy model, a rule based expert system, etc. - can be used as an expert.

The modular philosophy was applied in the steel-making converter modeling task. An important advantage of the modular architecture is, that it can integrate different forms of knowledge available about the physical system, so it is a hybrid-neural modeling system.

In a real-world system identification task usually there is certain prior information, physical knowledge, even if it is not enough to build physical models. To utilize all available knowledge in an efficient way has great importance. The implemented architecture is shown in Figure 19 [47].

The system has three layers. The first (input) layer is an expert system and it is responsible for data preprocessing, data filtering, data correction, filling the gaps in the data base, etc. It is also responsible to find inconsistency of the data, and to find - if any - clusters of the data that can be handled by different means, different approaches. The input expert system has to decide how to handle the current data record, if it is a standard case or it has to be treated specially. It decides according to the given rules of the current model, which neural network or other model must be used. It also can correct some of the data according to the knowledge about measurement noise or measurement device errors. It records this decision also in the knowledge base to be used by the later experts to calculate correction terms and to integrate the results.

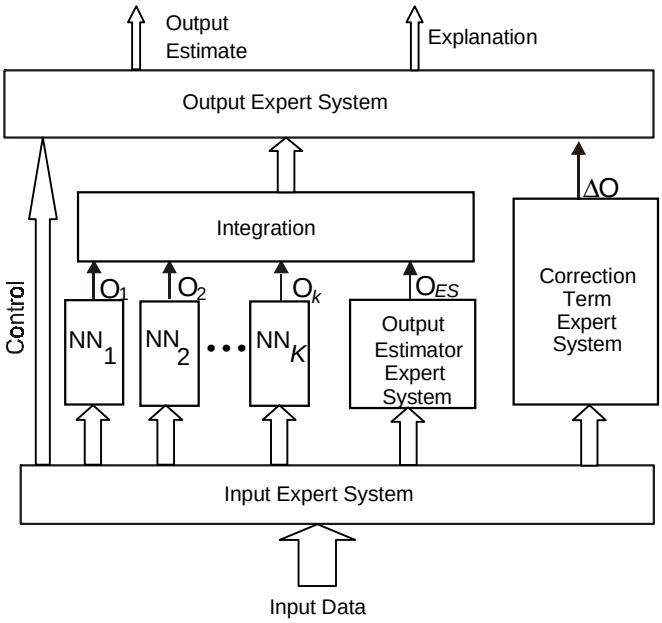

Figure 19: The hybrid-neural modeling system.

The second layer contains the direct modeling devices. It is formed from different neural models that can work with the data belonging to different clusters. In some cases such models cannot be used alone, it may happen that they should be used just together with certain correction terms that modify the result of a neural model. The system makes it possible to build any other modeling device (e.g., mathematical models or expert systems) into this layer in addition to the neural models. However, at present neither mathematical models, nor expert systems can compete with neural ones. So far only such mathematical models could be formed that gave reliable prediction in a small neighborhood of some special working points. These models can be used in the validation of the neural models, or in the explanation generation (see below).

The third or output layer is the decision-maker of the whole modeling system. It has two main tasks: to validate the results, and to make the final prediction using some direct information from the first layer. This layer also uses symbolic rules. It validates the result of the second layer and makes a decision if the result can be accepted at all. This decision-making is based on different information: for example, some direct information from the input layer, or the information obtained from more than one experts of the second layer. As an example for the first case it may happen that the input data are so special that there is no valid model for them in the second layer. Although it is a rare situation, this must be detected by the input expert system and the whole system must be able to give some valid answer even in such cases. This answer informs the staff that in this special case the whole system cannot give reliable output, they must determine it using any other (e.g., conventional) method. In the second case validation is based on the results of more than one expert modules of the second layer. Using these results the output expert system will form the final answer, which may be some combination of the results of more experts or a corrected value of a given expert. The correction term can be determined using the results of other expert modules (e.g., other neural networks), or a separated expert system, the role of which is to determine correction terms directly for the special cases.

A further important task of the output layer is the *explanation generation* what is also based on built-in expert knowledge. As neural networks themselves form black-box models, they cannot generate explanation of the result automatically. However, the acceptance of such results by an industrial community is rather questionable even if this result is quite good. The purpose of explanation generation is to increase the acceptance of the results of the modeling system.

4.11. Conclusions

The purpose of this paper was to give an overview about system identification and to show the important role of neural networks in this field. It was shown that neural networks are general black box modeling devices, which have many attractive features: they are universal approximators, they have the capability of adaptation, fault tolerance, robustness, etc. For system modeling several different static and dynamic neural architectures can be constructed, so neural architectures are flexible enough for a rather large class of identification tasks. The construction of neural models - as they are black box architectures - is mainly based on measurement data observed about the system. This is why one of the most important parts of black box modeling is the collection of as much relevant data as possible, which cover the whole operating range of interest. As it was shown in the example of LD converter modeling, the construction of data base needs to solve many additional problems; to handle noisy data, missing data, unreliable data, to separate the whole data base into training set and test set, etc. All these problems need proper preprocessing, the importance of which cannot be overemphasized.

Moreover, according to the experiences obtained from real-world modeling tasks, prior information and any additional knowledge to the observation has great importance. Prior information helps us to select proper model structure, to design excitation signal if it is possible to use excitation signals at all, to determine the operating range where valid model should be obtained, etc. An important implication obtained from complex real-world identification problems is that using only one approach, one paradigm usually cannot results in satisfactory model. Combining different paradigms, however can join the advantages of the different approaches, can utilize different representations of knowledge, and can help to understand the result obtained. This latter is especially important in neural modeling, because neural models cannot give explanation of the model, and without explanation, the lack of physical meaning may reduce the acceptance of the black box models even if their behavior is rather close to that of the system.

References

[1] L. Ljung, System Identification - Theory for the User. Prentice-Hall, N.J. 2nd edition, 1999.
[2] J. Schoukens and R. Pintelon, System Identification. A Frequency Domain Approach, IEEE Press, New York, 2001.
[3] T. Söderström and P. Stoica, System Indentification, Prentice Hall, Enhlewood Cliffs, NJ. 1989.
[4] P. Eykhoff, System Identification, Parameter and State Estimation, Wiley, New York, 1974.
[5] A. P. Sage and J. L. Melsa, Estimation Theory with Application to Communications and Control, McGraw-Hill, New York, 1971.
[6] H. L. Van Trees, Detection Estimation and Modulation Theory, Part I. Wiley, New York, 1968.
[7] G. C. Goodwin and R. L. Payne, Dynamic System Identification, Academic Press, New York, 1977.
[8] K. Hornik, M. Stinchcombe and H. White, Multilayer Feed-forward Networks are Universal Approximators", Neural Networks Vol. 2. 1989. pp. 359-366.
[9] G. Cybenko, Approximation by Superposition of Sigmoidal Functions, Mathematical Control Signals Systems, Vol. 2. pp. 303-314, 1989.
[10] K. I. Funahashi, On the Approximate Realization of Continuous Mappings by Neural Networks", Neural Networks, Vol. 2. No. 3. pp. 1989. 183-192.
[11] M. Leshno, V. Y. Lin, A. Pinkus and S. Schocken, Multilayer Feed-forward Networks With a Nonpolynomial Activation Function Can Approximate Any Function, Neural Networks, Vol. 6. 1993. pp. 861-67
[12] J. S. Albus, A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC), Transaction of the ASME, Sep. 1975. pp. 220-227.
[13] Y. H. Pao, Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, Reading, Mass., 1989, pp. 197-222.
[14] D. F. Specht, Polynomial Neural Networks, Neural Networks, Vol.3. No. 1 pp. 1990. pp. 109-118,
[15] J. Park and I. W. Sandberg, Approximation and Radial-Basis-Function Networks, Neural Computation, Vol 5. No. 2. 1993. pp. 305-316.
[16] S. Haykin, Neural Networks. A comprehensive foundation, Second Edition, Prentice Hall, N. J.1999.

[17] M. H. Hassoun, Fundamentals of Artificial Neural Networks, MIT Press, Cambridge, MA. 1995.
[18] M. Brown and C.Harris, Neurofuzzy Adaptive Modelling and Control, Prentice Hall, New York, 1994.
[19] G. Horváth and T. Szabó, CMAC Neural Network with Improved Generalization Property for System Modelling, Proc. of the IEEE Instrumenation and Measurement Conference, Anchorage, 2002.
[20] T. Szabó and G. Horváth, CMAC and its Extensions for Efficient System Modelling and Diagnosis, Intnl. Journal of Applied Mathematics and Computer Science, Vol. 9. No. 3, pp.571-598, 1999.
[21] J. Hertz, A. Krogh and R. G. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley Publishing Co. 1991.
[22] K. S. Narendra and K. Pathasarathy, Identification and Control of Dynamical Systems Using Neural Networks, IEEE Trans. Neural Networks, Vol. 1. 1990. pp.
[23] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson and A. Juditsky: "Non-linear black-box modeling in system identification: a unified overview", Automatica, 31:1691-1724, 1995.
[24] A.N. Tikhonov, V.Y. Arsenin, Solutions of Ill-posed Problems, Washington, DC: W.H. Winston, 1997
[25] E. A. Wan, Temporal Backpropagation for FIR Neural Networks, Proc. of the 1990 IJCNN, Vol. I. pp. 575-580.
[26] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning Internal Representations by Error Propagation, in Rumelhart, D.E. - McClelland, J.L. (Eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1. MIT Press. pp. 318-362. 1986.
[27] R. J. Williams and D. Zipser, A Learning Algorithm for Continually Running Fully Recurrent Neural Networks, Neural Computation, Vol. 1. 1989. pp. 270-280.
[28] A. R. Barron, Universal Approximation Bounds for Superposition of Sigmoidal Functions, IEEE Trans. on Information Theory, Vol. 39. No. 3. 1993. pp. 930-945.
[29] V. N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[30] V.Cherkassky, F.Mulier, Learning from Data, Concepts, Theory and Methods, Wiley, New York, 1998
[31] H. Akaike, Information Theory and an Extension of the Maximum Likelihood Principle, Second Intnl. Symposium on Information Theory, Akadémiai Kiadó, Budapest, pp. 267-281. 1972.
[32] J. Rissanen, Modelling by Shortest Data Description, Automatica, Vol. 14. pp. 465-471, 1978.
[33] N. Murata, S. Yoshizawa and Shun-Ichi Amari, Network Information Criterion - Determining the Number of Hidden Units for an Artificial Neural Network Model, IEEE Trans. on Neural Networks, Vol. 5. No. 6. Pp. 865-871.
[34] X. He and H. Asada, A New Method for Identifying Orders of Input-Output Models for Nonlinear Dynamic Systems, Proc. of the American Control Conference, 1993. San Francisco, CA. USA. pp. 2520-2523.
[35] M. Stone, Cross-Validatory Choice and Assesment of Statistical Predictions, Journal of Royal Statistical Society. Se. B. Vol. 36. pp. 111-147.
[36] S. Amari, N. Murata, K.-R. Müller, M. Finke and, H. Yang, Asymptotic Statistical Theory of Ovetraining and Cross-Validation, IEEE Trans. on Neural Networks, Vol. 8. No. 5. pp. 985-998, 1997.
[37] S. Lawrence, C. Lee Giles and Ah Chung Tsoi, What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation, Technical Report, UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, 1996. p. 33.
[38] S. Saarinen, B. Bramley and G. Cybenko, Ill-conditioning in Neural Network Training Problems, SIAM Journal for Scientific and Statistical Computing, 1991.
[39] L. Ljung and J. Sjöberg, A System Identification Perspective on Neural Networks, 1992.
[40] B. Pataki, G. Horváth, Gy. Strausz, and Zs. Talata, Inverse Neural Modeling of a Linz-Donawitz Steel Converter, e & i Elektrotechnik und Informationstechnik, Vol. 117. No. 1. 2000. pp. 13-17.
[41] G. Horváth, B. Pataki and Gy. Strausz, Black box Modeling of a Complex Industrial Process, Proc. of the 1999 IEEE Conference and Workshop on Engineering of Computer Based Systems, Nashville, TN, USA. 1999. pp. 60-66.
[42] M. Deistler, Linear Dynamic Errors-in-Variables Models, Journal of Applied Probability, Vol. 23. pp. 23-39, 1986.
[43] J. Van Gorp, J. Schoukens and R. Pintelon, Learning Neural Networks with Noisy Inputs Using the Errors-In-Variables Approach, IEEE Trans. on Neural Networks, Vol. 11. No.2 . pp. 402-414. 2000.
[44] G. Horváth, L. Sragner and T. Laczó, Impoved Model Order Estimation by Combining Errors-in-Variables and Lipschitz Methods, a forthcoming paper
[45] P. Sollich and A. Krogh, Learning with Ensembles: How over-fitting can be useful, In Advances in Neural Information Processing Systems 8. D. S. Touretzky, M. C. Mozer and M. E. Hasselmo, eds, MIT Press, pp. 190-196, 1996.
[46] R. A. Jacobs, M. I. Jordan, S. J. Nowlan and G. E. Hinton, Adaptive Mixture of Local Experts, Neural Computation Vol. 3. No.1 pp. 79-87.1991.
[47] P. Berényi, G. Horváth, B. Pataki and Gy. Strausz, Hybrid-Neural Modeling of a Complex Industrial Process, Proc. of the IEEE Instrumentation and Measurement Technology Conference, Vol. III. pp.1424-1429. 2001.