On Neural Network Model Structures in System Identification

L. Ljung, J. Sjöberg, and H. Hjalmarsson

Department of Electrical Engineering
Linköping University
S-581 83 Linköping
Sweden
E-mail: ljung@isy.liu.se, sjoberg@isy.liu.se, hakan@isy.liu.se

1. Introduction and Summary

1.1 What is the Problem?

The identification problem is to infer relationships between past input-output data and future outputs. Collect a finite number of past inputs $u(k)$ and outputs $y(k)$ into the vector $\varphi(t)$

$$\varphi(t) = [y(t-1)\dots y(t-n_a)\ u(t-1)\dots u(t-n_b)]^T \tag{1.1}$$

For simplicity we let $y(t)$ be scalar. Let $d = n_a + n_b$. Then $\varphi(t) \in \mathbb{R}^d$. The problem then is to understand the relationship between the next output $y(t)$ and $\varphi(t)$:

$$y(t) \leftrightarrow \varphi(t) \tag{1.2}$$

To obtain this understanding we have available a set of observed data (sometimes called the "training set")

$$Z^N = \{[y(t), \varphi(t)] |\ t = 1, \dots N\} \tag{1.3}$$

From these data we infer a relationship

$$\hat{y}(t) = \hat{g}_N(\varphi(t)) \tag{1.4}$$

We index the function g with a "hat" and N to emphasize that it has been inferred from (1.3). We also place a "hat" on $y(t)$ to stress that (1.4) will in practice not be an exact relationship between $\varphi(t)$ and the observed $y(t)$. Rather $\hat{y}(t)$ is the "best guess" of $y(t)$ given the information $\varphi(t)$.

1.2 Black Boxes

How to infer the function $\hat{g}_N$? Basically we search for it in a parameterized family of functions

$$\mathcal{G} = \{g(\varphi(t), \theta) | \theta \in D_{\mathcal{M}}\} \tag{1.5}$$

How to choose this parameterization? A good, but demanding, choice of parameterization is to base it on physical insight. Perhaps we know the relationship between $y(t)$ and $\varphi(t)$ on physical grounds, up to a handful of physical parameters (heat transfer coefficients, resistances,...). Then parameterize (1.5) accordingly.

This paper only deals with the situation when physical insight is *not* used; i.e. when (1.5) is chosen as a flexible set of functions capable of describing almost any true relationship between y and φ. This is the *black-box approach.*

Typically, function expansions of the type

$$g(\varphi, \theta) = \sum_k \theta(k) g_k(\varphi) \tag{1.6}$$

are used, where

$$g_k(\varphi): \quad \mathbb{R}^d \to \mathbb{R}$$

and $\theta(k)$ are the components of the vector θ. For example, let

$$g_k(\varphi) = \varphi_k \ \ (k\text{:th component of } \varphi)\ k = 1, \dots, d.$$

Then, with (1.1)

$$y(t) = g(\varphi(t), \theta)$$

reads

$$y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = b_1 u(t-1) + \dots + b_{n_b} u(t-n_b)$$

if

$$a_i = -\theta(i) \qquad b_i = \theta(n_a + i)$$

so the familiar ARX-structure is a special case of (1.6), with a linear relationship between y and φ.

1.3 Nonlinear Black Box Models

The challenge now is the non-linear case: to describe general, non-linear, dynamics. How to select $\{g_k(\varphi)\}$ in this general case? We should thus be prepared to describe a "true" relationship

$$\hat{y}(t) = g_0(\varphi(t))$$

for any reasonable function $g_0 : \mathbb{R}^d \to \mathbb{R}$. The first requirement should be that $\{g_k(\varphi)\}$ is a **basis** for such functions, *i.e.* that we can write

$$[R1]: \quad g_0(\varphi) = \sum_{k=1}^{\infty} \theta(k) g_k(\varphi) \tag{1.7}$$

for any reasonable function g_0 using suitable coefficients $\theta(k)$. There is of course an infinite number of choices of $\{g_k\}$ that satisfy this requirement, the classical perhaps being the basis of polynomials. For $d = 1$ we would then have

$$g_k(\varphi) = \varphi^k$$

and (1.7) becomes **Taylor** or **Volterra** expansion. In practice we cannot work with infinite expansions like (1.7). A second requirement on $\{g_k\}$ is therefore to produce "good" approximations for finite sums: In loose notation:

$$[R2]: \quad \| g_0(\varphi) - \sum_{k=1}^{n} \theta(k) g_k(\varphi) \| \quad \text{"decreases quickly as } n \text{ increases"} \tag{1.8}$$

There is clearly no uniformly good choice of $\{g_k\}$ from this respect: It will all depend on the class of functions g_0 that are to be approximated.

1.4 Estimating $\hat{g}_N$

Suppose now that a basis $\{g_k\}$ has been chosen, and we try to approximate the true relationship by a finite number of the basis functions:

$$\hat{y}(t|\theta) = g(\varphi(t), \theta) = \sum_{k=1}^{n} \theta(k) g_k(\varphi(t)) \tag{1.9}$$

where we introduce the notation $\hat{y}(t|\theta)$ to stress that $g(\varphi(t), \theta)$ is a "guess" for $y(t)$ given the information in $\varphi(t)$ and given a particular parameter value θ. The "best" value of θ is then determined from the data set Z^N in (1.9) by

$$\hat{\theta}_N = \arg\min \sum_{k=1}^{N} |y(t) - \hat{y}(t|\theta)|^2 \tag{1.10}$$

The model will be

$$\hat{y}(t) = \hat{y}(t|\hat{\theta}_N) = \hat{g}_N(\varphi(t)) = g(\varphi(t), \hat{\theta}_N) \tag{1.11}$$

1.5 Properties of the Estimated Model

Suppose that the actual data have been generated by

$$y(t) = g_0(\varphi(t)) + e(t) \tag{1.12}$$

where $\{e(t)\}$ is white noise with variance λ. The estimated model (1.11) (*i.e.* the estimated parameter vector $\hat{\theta}_N$) will then be a random variable that depends on the realizations of both $e(t), t = 1, \ldots, N$ and $\varphi(t), t = 1, \ldots, N$. Denote its expected value by

$$\mathrm{E}\hat{g}_N = g_n^* = \sum_{k=1}^{n} \theta^*(k) g_k \tag{1.13}$$

where we used subscript n to emphasize the number of terms used in the function approximation.

Then under quite general conditions

$$\mathrm{E}|\hat{g}_N(\varphi(t)) - g_n^*(\varphi(t))|^2 = \lambda \cdot \frac{m}{N} \tag{1.14}$$

where E denotes expectation both with respect to $\varphi(t)$ and $\hat{\theta}_N$. Moreover, m is the number of estimated parameters, *i.e.*, dim θ. The total error thus becomes

$$\mathrm{E}|\hat{g}_N(\varphi(t)) - g_0(\varphi(t))|^2 = \| g_0(\varphi(t)) - g_n^*(\varphi(t)) \|^2 + \lambda \cdot \frac{m}{N} \tag{1.15}$$

The first term here is an approximation error of the type (1.8). It follows from (1.15) that there is a trade-off in the choice of how many basis functions to use. Each included basis function increases the variance error by λ/N, while it decreases the bias error by an amount that could be less than so. A third requirement on the choice of $\{g_k\}$ is thus to

[R3] Have a scheme that allows the exclusion of spurious basis functions from the expansion.

Such a scheme could be based on a priori knowledge as well as on information in Z^N.

1.6 Basis Functions

Out of the many possible choice of basis functions, a large family of special ones have received most of the current interest. They are all based on just one fundamental function $\sigma(\varphi)$, which is scaled in various ways, and centered at different points, *i.e.*

$$g_k(\varphi) = \sigma(\beta_k^T(\varphi + \tilde{\gamma}_k)) = \sigma(\beta_k^T \varphi + \gamma_k) = \sigma(\varphi, \eta_k) \tag{1.16}$$

where $\gamma_k = \beta_k^T \tilde{\gamma}_k$ and η_k is the $d+1$-vector

$$\eta_k = [\beta_k, \gamma_k] \tag{1.17}$$

Such a choice is not at all strange. A very simplistic approach would be to take $\sigma(\varphi)$ to be the indicator function (in the case $d = 1$) for the interval $[0, 1]$:

$$\sigma(\varphi) = \begin{cases} 1 & \varphi \in [0,1] \\ 0 & \varphi \notin [0,1] \end{cases}$$

For a countable collection of η_k (e.g. assuming all rational numbers) the functions $g_k(\varphi)$ would then contain indicator functions for any interval, arbitrarily small and placed anywhere along the real axis. Not surprisingly, these $\{g_k\}$ will be a basis for all continuous functions. Equivalently, it could be threshold function

$$\sigma(\varphi) = \begin{cases} 1 & \varphi > 0 \\ 0 & \varphi \leq 0 \end{cases} \tag{1.18}$$

since the basic indicator function is just the difference between two threshold functions.

1.7 What Is the Neural Network Identification Approach?

The basic Neural Network (NN) used for System Identification (one hidden layer feedforward net) is indeed the choice (1.16) with a smooth approximation for (1.18), often

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Include the parameter η in (1.16)-(1.17) among the parameters to be estimated, θ, and insert into (1.9). This gives the Neural Network model structure

$$\hat{y}(t|\theta) = \sum_{k=1}^{n} \alpha_k \sigma(\beta_k \varphi + \gamma_k)$$

$$\theta = [\alpha_k, \beta_k, \gamma_k],\ k = 1, \ldots, n \tag{1.19}$$

The $n \cdot (d+2)$-dimensional parameter vector θ is then estimated by (1.10).

1.8 Why Have Neural Networks Attracted So Much Interest?

This tutorial points at two main facts.

1. The NN function expansion has good properties regarding requirement [R2] for nonlinear functions g_0 that are "localized"; *i.e.* there is not much nonlinear effects going to the infinity. This is a reasonable property for most real life physical functions. More precisely, see (7.1) in Section 7..
2. There is a good way to handle requirement [R3] by *implicit or explicit regularization* (See Section 3.)

1.9 Related Approaches

Actually, the general family of basis functions (1.16), is behind both *Wavelet Transform Networks* and estimation of *Fuzzy Models*. The paper [2] explains these connections in an excellent manner.

2. The Problem

2.1 Inferring Relationships from Data

A wide class of problems in disciplines such as classification, pattern recognition and system identification can be fit into the following framework.

A set of observations (data)

$$Z^N = \{y(t),\ \Phi(t)\}_{t=1}^N$$

of two physical quantities $y \in \mathbb{R}^p$ and $\Phi \in \mathbb{R}^r$ is given. It may or may not be known which variables in Φ influence y. There may also be other, non-measured, variables v that influence y. Based on the observations Z^N, infer how the variables in Φ influence y.

Let φ be the variables in Φ that influence y, then we could represent the relation between φ, v and y by a function g_0

$$y = g_0(\varphi, v) \tag{2.1}$$

The problem is thus two-fold:

1. Find which variables in Φ that should be used in φ.
2. Determine g_0.

In identification of dynamical systems, finding the right φ is the model order selection problem. Then t represents the time index and $\Phi(t)$ would be the collection of all past inputs and outputs.

There are two issues that have to be dealt with when determining g_0:

1. Only finite observations in the φ-space are available.
2. The observations are perturbed by the non-measurable variable $\{v(t)\}$.

1) represents the function approximation problem, *i.e.* how to do interpolation and extrapolation, which in itself is an interesting problem. Notice that there would be no problem at all if y was given for all values of φ (if we neglect the non-measurable input) since the function then in fact would be defined by the data. 2) increases the difficulty further since then we cannot infer exactly how φ influences y even at the points of observations. Blended together, these two problems are very challenging. Below we will try to disclose the essential ingredients. For further insight in this problem see also [2].

2.2 Prior Assumptions

Notice that as stated, the problem is ill-posed. There will be far too many unfalsified models, i.e., models satisfying (2.1), if any function g and any non-measurable sequence $\{v(t)\}$ is allowed. Thus, it is necessary to include some a priori information in order to limit the number of possible candidates. However, often it is difficult to provide a priori knowledge that is so precise that the problem becomes well-defined. To ease the burden it is common to resort to some general principles:

1) Non-measurable inputs are additive. This means that g_0 is additive in its second argument, i.e.,

$$g_0(\varphi, v) = g_0(\varphi) + v$$

This is, for example, a relevant assumption when $\{v(t)\}$ is due mainly to measurement errors. Therefore v is often called disturbance or noise.

2) Try simple things first (Occam's razor). There is no reason to choose a complicated model unless needed. Thus, among all unfalsified models, select the simplest one. Typically, the simplest means the one that in some sense has the smoothest surface. An example is spline smoothing. Among the class C^2 of all twice differentiable functions on an interval I, the solution to

$$\min_{g \in C^2} \sum (y(t) - g(\varphi(t)))^2 + \lambda \int_I (g''(\varphi))^2 d\varphi$$

is given by the cubic spline, [38]. Other ways to penalize the complexity of a function are information based criteria, such as AIC, BIC and MDL, regularization (or ridge penalty) and cross-validation.

2.3 Function Classes

Thus, g_0 is assumed to belong to some quite general family $\mathcal{G}$ of functions. The function estimate $\hat{g}_N^n$ however, is restricted to belong to a possibly more limited class of functions, $\mathcal{G}_n$ say. This family $\mathcal{G}_n$, where n represents the complexity of the class[1], is a member of a sequence of families $\{\mathcal{G}_n\}$ that satisfy $\mathcal{G}_n \to \mathcal{G}$. As explained above, the complexity of $\hat{g}_N^n$ is allowed to depend on Z^N, i.e., n is a function of Z^N. We will indicate this by writing $n(N)$.

In this perspective, an identification method can be seen as a rule to choose the family $\{\mathcal{G}_n\}$ together with a rule to choose $n(N)$ and an estimator that given these provides an estimate $\hat{g}_N^{n(N)}$. Notice that both the selection of $\{\mathcal{G}_n\}$ and $n(N)$ can be driven by data. This possibility is, as we shall see in Section 7., very important.

Typical choices of $\mathcal{G}$ are Hölder Balls which consist of Lipschitz continuous functions:

[1] Typically n is the number of basis functions in the class.

$$\Lambda^\alpha(C) = \{f : |f(x) - f(y)| \leq C \cdot |x - y|^\alpha\} \tag{2.2}$$

and L_p Sobolev Balls which have derivatives of a certain degree which belongs to L_p:

$$W_p^m(C) = \{f : \int |f^{(m)}(t)|^p dt \leq C^p\} \tag{2.3}$$

Recently, Besov classes and Triebel classes, [35] have been employed in wavelet analysis. The advantage with these classes are that they allow for spatial inhomogenity. Functions in these classes can be locally spiky and jumpy.

3. Some General Estimation Results

The basic estimation set-up is what is called *non-linear regression* in statistics. The problem is as follows. We would like to estimate the relationship between a scalar y and $\varphi \in \mathbb{R}^d$. For a particular value $\varphi(t)$ the corresponding $y(t)$ is assumed to be

$$y(t) = g_0(\varphi(t)) + e(t) \tag{3.1}$$

where $\{e(t)\}$ is supposed to be a sequence of independent random vectors, with zero mean values and variance

$$\mathrm{E}\, e(t)e^T(t) = \lambda \tag{3.2}$$

To find the function g_0 in (3.1) we have the following information available:

1. A parameterized family of functions

$$\mathcal{G}_m = \{g(\varphi(t), \theta) | \theta \in D_{\mathcal{M}} \subset \mathbb{R}^m\} \tag{3.3}$$

2. A collection of observed y, φ-pairs:

$$Z^N = \{[y(t), \varphi(t)], t = 1, ..., N\} \tag{3.4}$$

The typical way to estimate g_0 is then to form the scalar valued function

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^{N} |y(t) - g(\varphi(t), \theta)|^2 \tag{3.5}$$

and determine the parameter estimate $\hat{\theta}_N$ as its minimizing argument:

$$\hat{\theta}_N = \arg\min V_N(\theta) \tag{3.6}$$

The estimate of g_0 will then be

$$\hat{g}_N(\varphi) = g(\varphi, \hat{\theta}_N) \tag{3.7}$$

Sometimes a general, non-quadratic, norm is used in (3.4)

$$V_N(\theta) = \frac{1}{N}\sum_{t=1}^{N} \ell(\varepsilon(t,\theta)) \tag{3.8}$$

$$\varepsilon(t,\theta) = y(t) - g(\varphi(t),\theta)$$

Another modification of (3.4) is to add a *regularization term*,

$$W_N(\theta) = V_N(\theta) + \delta|\theta - \theta^{\#}|^2 \tag{3.9}$$

(and minimize W rather than V) either to reflect some prior knowledge that a good θ is close to $\theta^{\#}$ or just to improve numerical and statistical properties of the estimate $\hat{\theta}_N$. Again, the quadratic term in (3.9) could be replaced by a non-quadratic norm.

Now, what are the properties of the estimated relationship $\hat{g}_N$? How close will it be to g_0? Following some quite standard results (see, e.g., [20, 31]), we have the following properties. We will not state the precise assumptions under which the results hold. Generally it is assumed that $\{\varphi(t)\}$ is (quasi)-stationary and has some mixing property (i.e., that $\varphi(t)$ and $\varphi(t+s)$ become less and less dependent as s increases). The estimate $\hat{\theta}_N$ is a random variable that depends on Z^N. Let E denote expectation with respect to both $e(t)$ and $\varphi, t = 1, ..., N$. Let

$$\theta^* = \mathrm{E}\hat{\theta}_N$$

and

$$g^*(\varphi) = g(\varphi, \theta^*)$$

Then $g^*(\varphi)$ will be as close as possible to $g_0(\varphi)$ in the following sense:

$$\arg\min_{g \in \mathcal{G}_m} \mathrm{E}|g(\varphi) - g_0(\varphi)|^2 = g^*(\varphi) \tag{3.10}$$

where expectation E is over the distribution of φ that governed the observed sample Z^N. We shall call

$$g^*(\varphi) - g_0(\varphi)$$

the *bias error*. Moreover, if the bias error is small enough, the variance will be given approximately by

$$\mathrm{E}|\hat{g}_N(\varphi) - g^*(\varphi)|^2 \approx \frac{m}{N}\lambda \tag{3.11}$$

Here m is the dimension of θ (number of estimated parameters), N is the number of observed data pairs and λ is the noise variance. Moreover, expectation both over $\hat{\theta}_N$ and over φ, assuming, the same distribution for φ as in the sample Z^N. The total integrated mean square error (IMSE) will thus be

$$\mathrm{E}|\hat{g}_N(\varphi) - g_0(\varphi)|^2 = ||g^*(\varphi) - g_0(\varphi)||^2 + \frac{m}{N}\lambda \tag{3.12}$$

Here the double bar norm denotes the functional norm, integrating over φ with respect to its distribution function when the data were collected. Now, what happens if we minimize the regularized criterion W_N in (3.9)?

1. The value $g^*(\varphi)$ will change to the function that minimizes

$$\mathrm{E}|g(\varphi,\theta) - g_0(\varphi)|^2 + \delta|\theta - \theta^{\#}|^2 \tag{3.13}$$

2. The variance (3.11) will change to

$$\mathrm{E}|\hat{g}_N(\varphi) - g^*(\varphi)|^2 \approx \frac{r(m,\delta)}{N} \cdot \lambda \tag{3.14}$$

where

$$r(m,\delta) = \sum_{k=1}^{m} \frac{\sigma_i^2}{(\sigma_i + \delta)^2} \tag{3.15}$$

where σ_i are the eigenvalues (singular values) of $\mathrm{E}V_N''(\theta)$, the second derivative matrix (the Hessian) of the criterion.

How to interpret (3.15)? A redundant parameter will lead to a zero eigenvalue of the Hessian. A small eigenvalue of V'' can thus be interpreted as corresponding to a parameter (combination) that is not so essential: "A spurious parameter". The regularization parameter δ is thus a threshold for spurious parameters. Since the eigenvalues σ_i often are widely spread we have

$$r(m,\delta) \simeq m^{\#} = \#\text{ of eigenvalues of } V'' \text{ that are larger than } \delta$$

We can think of $m^{\#}$ as "the efficient number of parameters in the parameterization". Regularization thus decreases the variance, but typically increases the bias contribution to the total error.

4. The Bias/Variance Trade-Off

Consider now a sequence of parameterized function families

$$\mathcal{G}_n = \{g_n(\varphi(t),\theta) | \theta \in D_{\mathcal{M}} \subset \mathbb{R}^m\}$$
$$n = 1, 2, 3\ldots \tag{4.1}$$

where n denotes the number of basis function (1.9).

In the previous section we saw that the integrated mean square error is typically split into two terms the *variance* term and the *bias* term

$$V_2(\hat{g}_N^n, g_0) = V_2(\hat{g}_N^n, g_n^*) + V_2(g_n^*, g_0) \tag{4.2}$$

where, according to (3.11),

$$V_2(\hat{g}_N^n, g_n^*) \sim \frac{m}{N}. \tag{4.3}$$

The bias term, which is entirely deterministic, decreases with n. Thus, for a given family $\{\mathcal{G}_n\}$ there will be an optimal $n = n^*(N)$ that balances the variance and bias terms.

Notice that (4.3) is a very general expression that holds almost regardless of how the sequence $\{\mathcal{G}_n\}$ is chosen. Thus, it is in principle only possible to influence the bias error. In order to have a small integrated mean square error it is therefore of profound importance to choose $\{\mathcal{G}_n\}$ such that the bias is minimized. An interesting possibility is to let the choice of $\{\mathcal{G}_n\}$ be data driven. This may not seem like an easy task but here wavelets have proven to be useful, see Section 7.

When the bias and the variance can be exactly quantified, the integrated mean square error can be minimized with respect to n. This gives the optimal model complexity $n^*(N)$ as a function of N. However, often it is only possible to give the rate with which the bias decreases as a function of n and the rate with which the variance increases with n. Then it is only possible to obtain the rate with which $n^*(N)$ increases with N. Another problem is that if g_0 in reality belongs not to $\mathcal{G}$ but to some other class of functions, the rate will not be optimal. These considerations has lead to the development of methods where the choice of n is based on the observations Z^N. Basically, n is chosen so large that there is no evidence in the data that g_0 is more complex than the estimated model, but not larger than that. Then, as is shown in [14], the bias and the variance are matched. These adaptive methods are discussed in Section 7.

5. Neural Nets

What is meant by the term neural nets depends on the author. Lately neural net has become a word of fashion and today almost all kinds of models can be found by the names neural network somewhere in the literature. Old types of models, known for decades by other names, have been converted to, or reinvented as neural nets. This makes it impossible to cover all types of neural networks and only what is called feedforward and recurrent will be considered, which are the networks most commonly used in system identification. Information about other neural network models can be found in any introductory book in this field, e.g., [18, 29, 15].

In [16, 25, 34] alternative overviews of neural networks in system identification and control can be found. Also the books [41, 42] contain many interesting articles on this topic.

5.1 Feedforward Neural Nets

The step from the general function expansion (1.9) to what is called neural nets is not big. With the choice $g_k(\varphi) = \alpha_k\sigma(\beta_k\varphi + \gamma_k)$ where β_k is a parameter vector of size $\dim\varphi$, and γ_k and α_k are scalar parameters we obtain

$$g(\varphi) = \sum_{k=1}^{n} \alpha_k\sigma(\beta_k\varphi + \gamma_k) + \alpha_0 \tag{5.1}$$

where a mean level parameter α_0 has been added. This model is referred to as a *feedforward network* with one hidden layer and one output unit in the NN literature. In Fig. 5.1 it is displayed in the common NN way. The basis functions, called *hidden units*, *nodes*, or *neurons*, are univariate which makes the NN to an expansion in simple functions. The specific choice of $\sigma(\cdot)$ is the *activation function* of the units which is usually chosen identically for all units.

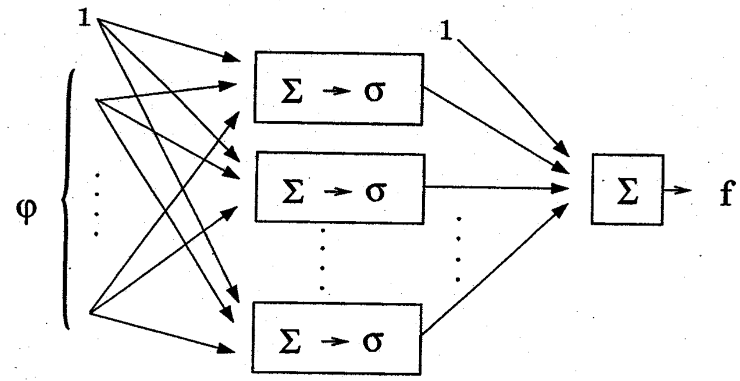

Fig. 5.1. Feedforward network with one hidden layer and one output unit. The arrows symbolize the parameters of the model.

The name *feedforward* is explained by the figure; there is a specific direction of flow in the computations when the output g is computed. First the weighted sums calculated at the input at each unit, then these sums pass the activation function and form the outputs of the hidden units. To form g, a weighted sum of the results from the hidden units is formed. This sum at the output is called the *output unit*. If g is vector function there are several output units forming an *output layer*. The input, φ, is sometimes called the *input layer*. The weights at the different sums are the parameters of the network.

In [6] it was shown that condition [R1], (1.7), holds if the activation function is chosen to be *sigmoidal* which is defined as

Definition 5.1. *Let $\sigma(x)$ be continuous. Then $\sigma(x)$ is called a sigmoid function if it has the following properties*

$$\sigma(x) \to \begin{cases} a & \text{as } x \to +\infty \\ b & \text{as } x \to -\infty \end{cases} \tag{5.2}$$

where $a, b,\ b < a$ are any real values.

The most common choice is

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{5.3}$$

which gives a smooth, differentiable, model with the advantage that gradient based parameter estimate methods can be used, see Section 6.. However, in [19] it is shown that (5.1) is a universal approximator, i.e., [R1] holds for all non-polynomial $\sigma(\cdot)$ which are continuous except at most in a set of measure zero.

The one hidden layer NN is related to the Projecting Pursuit (PP) model, see Section 7. In each unit a direction is estimated (β_k) but, in difference to PP, the function in this direction is fixed except for scaling and translation.

The one hidden layer network (5.1) can be generalized into a multi-layer network with several layers of hidden units. The outputs from the first hidden layer then feeds in to another hidden layer which feeds to the output layer - or another hidden layer. This is best shown with a picture; in Fig. 5.2 such a net with two hidden layers and several outputs is shown. The formula for a NN with two hidden layer and one output becomes

$$g(\varphi) = \sum_i \theta^3_{1,i}\sigma\left(\sum_j \theta^2_{i,j}\sigma(\sum_m \theta^1_{r,m}\varphi_m)\right) \tag{5.4}$$

The parameters have three indexes. $\theta^M_{j,i}$, is the parameter between the unit i in one layer and unit j in the following layer. M denotes which layer the parameter belongs to. The translation parameters corresponding to γ_k in (5.1) has not been written out.

At first, because of the general approximation ability of the NN with one hidden layer, there seems to be no reason to add more hidden layers. However, the rate of convergence might be very slow for some functions and it might be possible with a much faster convergence with two hidden layers (i.e., condition [R2], (1.8) might favor two layers). Also, in [33] it is shown that in certain control applications a two hidden layer NN can stabilize systems which cannot possibly be stabilized by NN with only one hidden layer.

5.2 Recurrent Neural Nets

If some of the inputs of a feedforward network consist of delayed outputs from the network, or some delayed internal state, then the network is called a *recurrent network*, or sometimes a dynamic network. In Fig. 5.3 an example of a recurrent net with two past outputs fed back into the network.

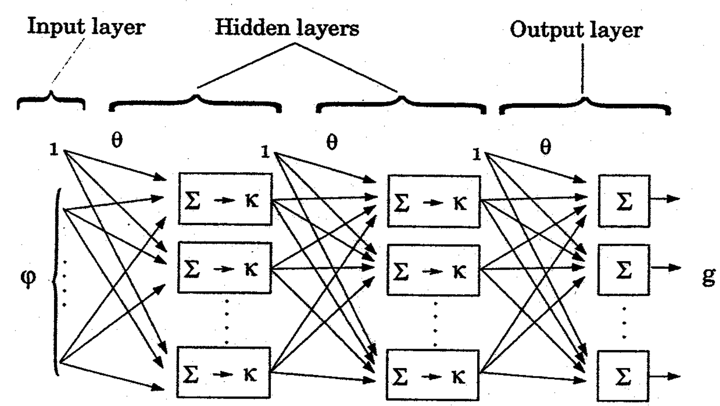

Fig. 5.2. Feedforward network with two hidden layers.

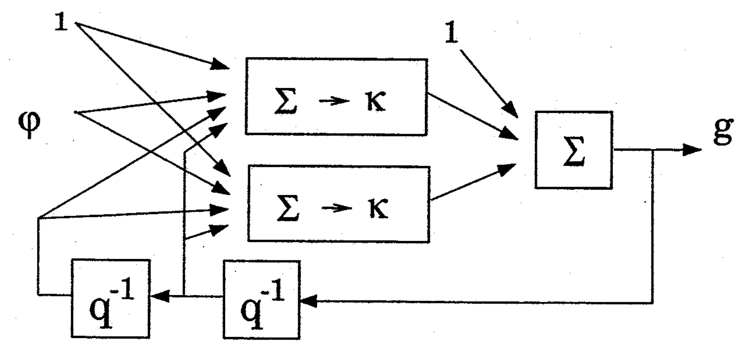

Fig. 5.3. Recurrent network. q^{-1} delays the signal one time sample.

The dynamic recurrent networks are especially interesting for identification, and in Section 9. two black-box models introduced based on recurrent networks.

Recurrent networks can also be used as a non-linear state-space model. This is investigated in [23].

6. Algorithmic Aspects

In this section we shall discuss how to achieve the best fit between observed data and the model, *i.e.* how to carry out the minimization of (1.10).

$$V_N(\theta) = \frac{1}{2N}\sum_{t=1}^{N} |y(t) - g(\varphi(t),\theta)|^2 \tag{6.1}$$

No analytic solution to this problem is possible, so the minimization has to be done by some numerical search procedure. A classical treatment of the problem of how to minimize sum of squares is given in [7]. A survey of methods for the NN application is given in [18] and in [36]. Most efficient search routines are based on iterative local search in a "downhill" direction from the current point. We then have an iterative scheme of the following kind

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} - \mu_i R_i^{-1} \nabla \hat{g}_i \tag{6.2}$$

Here $\hat{\theta}^{(i)}$ is the parameter estimate after iteration number i. The search scheme is thus made up from the three entities

- μ_i step size
- $\nabla \hat{g}_i$ an estimate of the gradient $V_N'(\hat{\theta}^{(i)})$
- R_i a matrix that modifies the search direction

It is useful to distinguish between two different minimization situations

(i) *Off-line* or *batch*: The update $\mu_i R_i^{-1} \nabla \hat{g}_i$ is based on the whole available data record Z^N.

(ii) *On-line* or *recursive*: The update is based only on data up to sample i (Z^i), (typically done so that the gradient estimate $\nabla \hat{g}_i$ is based only on data just before sample i.)

We shall discuss these two modes separately below. First some general aspects will be treated.

6.1 Search Directions

The basis for the local search is the gradient

$$V_N'(\theta) = -\frac{1}{N}\sum_{t=1}^{N}(y(t) - g(\varphi(t),\theta))\psi(\varphi(t),\theta) \tag{6.3}$$

where

$$\psi(\varphi(t),\theta) = \frac{\partial}{\partial\theta} g(\varphi(t),\theta) \quad (d|1 - \text{vector}) \tag{6.4}$$

It is well known that gradient search for the minimum is inefficient, especially close to the minimum. Then it is optimal to use the *Newton search direction*

$$R^{-1}(\theta)V_N'(\theta) \tag{6.5}$$

where

$$R(\theta) = V_N''(\theta) = \frac{1}{N}\sum_{t=1}^{N}\psi(\varphi(t),\theta)\psi^T(\varphi(t),\theta) + \frac{1}{N}\sum_{t=1}^{N}(y(t) - g(\varphi(t),\theta))\frac{\partial^2}{\partial\theta^2} g(\varphi(t),\theta) \tag{6.6}$$

The true Newton direction will thus require that the second derivative

$$\frac{\partial^2}{\partial\theta^2} g(\varphi(t),\theta)$$

be computed. Also, far from the minimum, $R(\theta)$ need not be positive semidefinite. Therefore alternative search directions are more common in practice:

- *Gradient direction.* Simply take

$$R_i = I \tag{6.7}$$

- *Gauss-Newton direction.* Use

$$R_i = H_i = \frac{1}{N}\sum_{t=1}^{N}\psi(\varphi(t),\hat{\theta}^{(i)})\psi^T(\varphi(t),\hat{\theta}^{(i)}) \tag{6.8}$$

- *Levenberg-Maquard direction.* Use

$$R_i = H_i + \delta I \tag{6.9}$$

  where H_i is defined by (6.8).
- *Conjugate gradient direction.* Construct the Newton direction from a sequence of gradient estimates. Loosely, think of V_N'' as constructed by difference approximation of d gradients. The direction (6.5) is however constructed directly, without explicitly forming and inverting V''.

It is generally considered, [7], that the Gauss-Newton search direction is to be preferred. For ill-conditioned problems the Levenberg-Maquard modification is recommended. However, good results with conjugate gradient methods have also been reported in NN applications ([36]).

6.2 Back-Propagation: Calculation of the Gradient

The only model-structure dependent quantity in the general scheme (6.2) is the gradient of the model structure (6.4). For a one-hidden-layer structure (5.1) this is entirely straightforward, since

$$\begin{aligned} \frac{d}{d\alpha}\alpha\sigma(\beta\varphi+\gamma) &= \sigma(\beta\varphi+\gamma) \\ \frac{d}{d\gamma}\alpha\sigma(\beta\varphi+\gamma) &= \alpha\sigma'(\beta\varphi+\gamma) \\ \frac{d}{d\beta}\alpha\sigma(\beta\varphi+\gamma) &= \alpha\sigma'(\beta\varphi+\gamma)\varphi \end{aligned}$$

For multi-layer NNs the gradient is calculated by the well known *Back-Propagation* (BP) method which can be described as the chain rule for differentiation applied to the expression (5.4). It also makes sure to re-use intermediate results which are needed at several places in the algorithm. Actually, the only complicated with the algorithm is to keep track of all indexes.

Backpropagation has been "rediscovered" several times (see, e.g., [40, 28]).

Here the algorithm will be derived for the case where the network has two hidden layers and one output unit. For multi output models and with less- or more hidden layers only minor changes have to be done.

Consider the NN model (5.4). Denote by x_b^M and f_b^M the result at unit b in layer M before and after the activation function, respectively. That is

$$f_b^M = \sigma(x_b^M)$$

We can then write $g(\varphi) = x_1^3 = \sum_i \theta_{1,i}^3 f_i^2$ and the derivative with respect to one of the parameters in the output layer becomes

$$\psi(\varphi)_{3,1,b} = \frac{\partial g}{\partial \theta_{1,b}^3} = f_b^2$$

In the same way $x_a^2 = \sum_m \theta_{a,m}^2 f_m^1$ and the derivative of a parameter in the middle layer becomes

$$\psi(\varphi)_{2,a,b} = \frac{\partial g}{\partial \theta_{a,b}^2} = \delta_a^2 f_b^1$$

where

$$\delta_a^2 = \theta_{1,a}^3 \sigma'(x_a^2)$$

For the first layer we can write $x_a^1 = \sum_m \theta_{a,m}^1 \varphi_m$ and the derivative of a parameter in this layer becomes

$$\psi(\varphi)_{1,a,b} = \frac{\partial g}{\partial \theta_{a,b}^1} = \delta_a^1 \varphi_b$$

where

$$\delta_a^1 = \sum_j \delta_j^2 \theta_{j,a}^2 \sigma'(x_a^1)$$

The nice feature is that $\{f_b^M\}$ and $\{x_b^M\}$ are obtained as intermediate results when $g(\varphi)$ is calculated (forward propagation in the network). Calculating $\{\delta_a^M\}$ can be viewed as propagating $g(\varphi)$ backwards through the net, and this is the origin of the name of the algorithm.

The calculations are further simplified by the relation of the derivative of the sigmoid which follows from (5.3).

$$\sigma'(\cdot) = \sigma(\cdot)\,(1-\sigma(\cdot)) \tag{6.10}$$

6.3 Implicit Regularization

Recall the discussion about regularization in Section 3.. We pointed out that the parameter δ in (3.9) acts like a knob that affects the "efficient number of parameters used". It thus plays a similar role as the model size:

- Large δ: small model structure, small variance, large bias
- Small δ: large model structure large variance, small bias

It is quite important for NN applications to realize that there is a direct link between the iterative process (6.2) and regularization in the sense that *aborting the iterations before the minimum has been found, has a quite similar effect as regularization.* This was noted in [37] and pointed out as the cause of "overtraining" in [30]. More precisely, the link is as follows (when quadratic approximations are applicable)

$$(I - \mu R^{-1} V'')^i \sim \delta(\delta I + V'')^{-1}$$

so, as the iteration number increases, this corresponds to a regularization parameter that decreases to zero as

$$\log \delta \sim -i \tag{6.11}$$

How to know when to stop the iterations? As $i \to \infty$ the value of the criterion V_N will of course continue to decrease, but as a certain point the corresponding regularization parameter becomes so small that increased variance starts to dominate over decreased bias. This should be visible when the model is tested on a fresh set – the *Validation data* (often called *generalization data* in the NN context). We thus evaluate the criterion function on this fresh data set, and plot the fit as a function of the iteration number. A typical such plot is shown in Fig. 9.3. The point where the fit starts to be worse for the validation data is the iteration number (the degree of regularization or the effective model flexibility) where we are likely to strike the optimal balance between bias and variance error. Experience with NN applications has shown that this often is a very good way of limiting the actual model flexibility by effectively eliminating spurious parameters, i.e., dealing with requirement [R3], mentioned in Section 1.

6.4 Off-line and On-line Algorithms

The expressions (6.3) and (6.6) for the Gauss-Newton search clearly assume that the whole data set Z^N is available during the iterations. If the application is of an off-line character, i.e., the model $\hat{g}_N$ is not required during the data acquisition, this is also the most natural approach.

However, in the NN context there has been a considerable interest in on-line (or recursive) algorithms, where the data are processed as they are measured. Such algorithms are in NN contexts often also used in off-line situations. Then the measured data record is concatenated with itself several times to create a (very) long record that is fed into the on-line algorithm. We may refer to [21] as a general reference for recursive parameter estimation algorithm. In [32] the use of such algorithms is the off-line case is discussed.

It is natural to consider the following algorithm as the basic one:

$$\hat{\theta}(t)=\hat{\theta}(t-1)+\mu_t R_t^{-1}\psi(\varphi(t),\hat{\theta}(t-1))\varepsilon(t,\hat{\theta}(t-1)) \tag{6.12}$$

$$\varepsilon(t,\theta)=y(t)-g(\varphi(t),\theta) \tag{6.13}$$

$$R_t = R_{t-1}+ \mu_t[\psi(\varphi(t),\hat{\theta}(t-1))\psi^T(\varphi(t),\hat{\theta}(t-1))-R_{t-1}] \tag{6.14}$$

The reason is that if $g(\varphi(t),\theta)$ is linear in θ, then (6.12) – (6.14), with $\mu_t = 1/t$, provides the analytical solution to the minimization problem (6.1). This also means that this is a natural algorithm close to the minimum, where a second order expansion of the criterion is a good approximation. In fact, it is shown in [21], that (6.12)–(6.14) in general gives an estimate $\hat{\theta}(t)$ with the same ("optimal") statistical, asymptotic properties as the true minimum to (6.1).

In the NN literature, often some averaged variants of (6.12)–(6.14) are discussed:

$$\hat{\hat{\theta}}(t)=\hat{\hat{\theta}}(t-1)+\mu_t R_t^{-1}\nabla\hat{g}_t \tag{6.15}$$

$$\nabla\hat{g}_t=\nabla\hat{g}_{t-1}+ \gamma_t[\psi(\varphi(t),\hat{\theta}(t-1))\varepsilon(t,\hat{\theta}(t-1))-\nabla\hat{g}_{t-1}] \tag{6.16}$$

$$\hat{\theta}(t)=\hat{\theta}(t-1)+\rho_t[\hat{\hat{\theta}}(t)-\hat{\theta}(t-1)] \tag{6.17}$$

The basic algorithm (6.12)–(6.14) then corresponds to $\gamma_t = \rho_t = 1$. Now, when do the different averages accomplish?

Let us first discuss (6.17) (and take $\gamma_t \equiv 1$). This is what has been called "accelerated convergence". It was introduced by [27] and has been extensively discussed by Kushner and others. The remarkable thing with this averaging is that we achieve the same asymptotic statistical properties of $\hat{\theta}(t)$ by (6.15)–(6.17) with $R_t = I$ (gradient search) as by (6.12)–(6.14) if

$$\gamma_1 = 1$$

$$\rho_t = 1/t$$

$$\mu_t >> \rho_t \qquad \mu_t \to 0$$

It is thus an interesting alternative to (6.12)–(6.14), in particular if $\dim\theta$ is large so R_t is a big matrix.

We now turn to the averaging in (6.16).For $\gamma < 1$ this gives what is known as a *momentum* term. Despite its frequent use in NN applications, it is more debatable. An immediate argument for (6.16) is that the averaging makes the gradient $\nabla\hat{g}_t$ more reliable (less noisy) so that we can take larger steps in (6.15). It is, however, immediate to verify that exactly the same averaging takes place in (6.15) if smaller steps are taken. A second argument is that (6.16) lends a "momentum" effect to the gradient estimate $\nabla\hat{g}_t$. That is, due to the low pass filter, $\nabla\hat{g}_t$ will reflect not only the gradient at $\hat{\theta}(t-1)$, but also at several previous values of $\hat{\theta}(k)$. This means that the update push in (6.15) will not stop immediately at value θ where the gradient is zero. This could of course help to push away from a non-global, local minimum, which is claimed to be a useful feature. However, there seems to be no systematic investigation of whether this possible advantage is counter balanced by the fact that more iterations will be necessary for convergence.

6.5 Local Minima

A fundamental problem with minimization tasks like (6.1) is that $V_N(\theta)$ may have several or many local (non-global) minima, where local search algorithms may get caught. There is no easy solution to this problem. It is usually well used effort to spend some time to come up with a good initial value $\theta^{(0)}$ where to start the iterations. Other than that, only various global search strategies are left, such as random search, random restarts, simulated annealing, the genetic algorithm and whathaveyou.

7. Adaptive Methods

The use of data to select the basis functions characterize adaptive methods. The adaptation can be more or less sophisticated. In its simplest form, only the number of basis functions is selected. The merits and limitations of this procedure are explained in the first subsection while the second subsection deals with more advanced methods where also the basis functions themselves are adapted to the data.

7.1 Adaptive Basis Function Expansion

Suppose that we have a set of basis functions $\{b_k\}$ that span $\mathcal{G}$. Each set of n basis functions would generate a function class $\mathcal{G}_n$ and a good idea would be to select these n basis function such that the approximation error is minimized among all possible choices of these sets of n basis functions. The problem of finding an n-dimensional subspace that minimizes the worst approximation error is is known as Kolmogorovs n-width problem, [26]. Depending on $\mathcal{G}$, the problem can be more or less complicated. For example, for the functions $\sum_{k=0}^{\infty} a_k z^k$that are analytic inside the disc of radius $r \leq 1$ satisfying $\sum_{k=0}^{\infty} |a_k|^2 r^{2k} < 1$, the optimal subspace is given by $\text{span}\{1, z, \dots, z^{n-1}\}$.

Wavelets. For orthonormal basis functions, the basis functions that correspond to the largest coefficients in the expansion of g_0 give the best approximation. Thus, an idea is to estimate a large number of coefficients and to select the n largest ones. It is interesting to note that with this procedure one get adaptation of the $\{\mathcal{G}_n\}$ to the smoothness of $\mathcal{G}$ for free; if the basis functions span several (or a scale of) spaces of functions, the approach will be optimal for all these spaces.

This approach has been exploited in the wavelet theory. Wavelet theory is based on orthonormal bases of L_2 that also span a wide scale of function spaces with a varying degree of smoothness, Besov and Triebel spaces, [35].

The basic problem with such a method is to determine which parameters are small and which are large, respectively. [10] has shown that the use of *shrinkage* gives (near) minimax rates in these spaces. Shrinkage essentially means that a threshold is determined that depends on the number of data. Parameter estimates less than this threshold are set to zero. Often, for technical reasons, a soft threshold is used instead. In that case, every wavelet coefficient is "pulled" towards zero by a certain non-linear function. This is conceptually closely related to the *regularization* procedure outlined in Section 3. Then, parameters are attracted towards the nominal value $\theta^{\#}$. However, so far explicit regularization does not seem to have been exploited in wavelet theory.

Neural Networks. Neural networks is an example of a structure where the basis functions appear more implicit. Consider the expression (5.1). This is an expansion with $\{\sigma(\varphi, \eta_k)\}$ as basis functions. The fact that the η_ks are estimated from data means that the basis functions are chosen adaptively. In other words, the basis functions are selected from data. Below we shall see that they have an important property when it comes to high-dimensional systems.

7.2 The "Curse" of Dimensionality

Almost all useful approximation theorems are asymptotic, i.e., they require the number of data to approach infinity, $N \to \infty$. In practical situations this cannot be done and it is of crucial importance how fast the convergence is. A general estimation of a function $\mathbb{R}^d \to \mathbb{R}$ becomes slower in N when d is larger and in most practical situations it becomes impossible to do general estimation of functions for d larger than, say, 3 or 4. For higher dimensions the number of data required becomes so large that it is in most cases not realistic. This is the curse of dimensionality. This can be shown with the following example

Example 7.1. Approximate a function $\mathbb{R}^d \to \mathbb{R}$ within the unit cube with the resolution 0.1. This requires that the distance between data is not larger than 0.1 in every direction, requiring $N = 10^d$ data. This is hardly realistic for $d > 4$. When there are noisy measurements the demand of data increase further.

7.3 Methods to Avoid the "Curse"

From the discussion in the preceding subsection it should be clear that general nonlinear estimation is not possible. Nevertheless, a number of methods have been developed to deal with the problems occurring for high-dimensional functions. The idea is to be able to estimate functions that in some sense have a low-dimensional character. *Projection pursuit regression*, [11], uses an approximation of the form

$$\hat{g}(\varphi) = \sum_{k=1}^{n} g_k\left(\varphi^T \theta(k)\right)$$

where the g_ks are smooth univariate functions. The method thus expands the function in n different directions. These directions are selected to be the most important ones and, for each of these, the functions g_k are optimized. Thus, it is a joint optimization over the directions $\{\theta(k)\}$ and the functions g_k. The claim is that for small n a wide class of functions can be well approximated by this expansion, [9]. The claim is supported by the fact that any smooth function in d variables can be written in this way, [8]. It is supposed to be useful for moderate dimensions, $d < 20$.

Projection pursuit regression is closely related to *neural networks* where the same function, any sigmoid function σ satisfying Definition 5.1, is used in all directions. The effectiveness of such methods has been illustrated in [1]: Consider the class of functions $\{g\}$ on $\mathbb{R}^d$ for which there is a Fourier representation $\tilde{g}$ which satisfies

$$C_f = \int |\omega||\tilde{g}(\omega)|d\omega < \infty.$$

Then there is a linear combination of sigmoidal functions such that

$$\int_{B_r} |g(x) - g_n(x)|^2 dx \leq \frac{(2rC_f)^2}{n} \tag{7.1}$$

where B_r is a ball with radius r. The important thing to notice here is that the degree of approximation as a function of n does not depend on the dimension d.

This work originated with the result in [17] where sinusoidal functions where used to prove a similar result. The above result is not limited to sinusoidal and sigmoidal functions and the same idea has been applied to projection pursuit regression, [43], *hinging hyperplanes*, [3], and *radial basis functions*, [12].

Notice, however, that the result is only an approximation result and a stochastic counterpart still awaits its proof.

[1] also showed that $1/n^{(2/d)}$ is a lower bound for the minimiax rate for linear methods. For large d, this rate is exceedingly slow compared with $1/n$. Thus, this is a serious disadvantage for the methods described in the previous section. In higher dimensional spaces, the convergence rate of linear method is much slower compared with certain non-linear methods.

8. Specific Properties of NN Structures

So, what are the special features of the Neural Net structure that motivate the strong interest? Based on the discussion so far, we may point to the following list of properties:

- The NN expansion is a basis, even for just one hidden layer, i.e., Requirement [R1] is satisfied.
- The NN structure does extrapolation in certain, adaptively chosen, directions and is localized across these directions. Like Projection Pursuit it can thus handle larger regression vectors, if the data pattern $[y(t), \varphi(t)]$ cluster along subspaces.
- The NN structure uses adaptive bases functions, whose shape and location are adjusted by the observed data.
- The approximation capability (Requirement [R2]) is good as manifested in (7.1).
- Regularization, implicit (stopped iterations) or explicit (penalty for parameter deviations, usually from zero) is a useful tool to effectively include only those basis functions that are essential for the approximation, without increasing the variance. (Requirement [R3]).
- In addition, NNs have certain advantages in implementation, both in hardware and software, due to the repetitive structure. The basis functions are built up from only one core function, σ. This also means that the structure is resilient to failures, since any node can play any other node's role, by adjusting its weights.

9. Models of Dynamical Systems Based on Neural Networks

We are now ready to take the step from general "curve fitting" to system identification. The choice of a model structure for dynamical systems contains two questions

1. What variables, constructed from observed past data, should be chosen as regressors, *i.e.*, as components of $\varphi(t)$?
2. What non-linear mapping should $\varphi(t)$ be subjected to, i.e., How many hidden layers in (5.1) should be used, and how many nodes should each layer have?

The second question is related to more general NN considerations, as discussed in Section 5.. The first one is more specific for identification applications. To get some guidance about the choice of regressors φ, let us first review the linear case.

9.1 A Review of Linear Black Box Models

The simplest dynamical model is the Finite Impulse Response model (FIR):

$$y(t) = B(q)u(t) + e(t) = b_1 u(t-1) + \ldots + b_n u(t-n) + e(t) \tag{9.1}$$

Here we have used q to denote the shift operator, so $B(q)$ is a polynomial in q^{-1}. The corresponding predictor is $\hat{y}(t|\theta) = B(q)u(t)$ is thus based on a regression vector

$$\varphi(t) = [u(t-1), u(t-2), \ldots, u(t-n)]$$

As n tends to infinity we may describe the dynamics of all ("nice") linear systems. However, the character of the noise term $e(t)$ will not be modeled in this way.

A variant of the FIR model is the Output Error model (OE):

$$y(t) = \frac{B(q)}{F(q)}u(t) + e(t) \tag{9.2}$$

where

$$F(q) = 1 + f_1 q^{-1} + \ldots + f_{n_f} q^{-n_f}$$

The predictor is

$$\hat{y}(t|\theta) = \frac{B(q)}{F(q)}u(t) \tag{9.3}$$

Also this predictor is based on past inputs only. It can be rewritten

$$\hat{y}(t|\theta) = b_1 u(t-1) + \ldots + b_{n_b} u(t-n_b) -$$

$$- f_1\hat{y}(t-1|\theta) - \ldots - f_{n_f}\hat{y}(t-n_f|\theta) \tag{9.4}$$

It is thus based on the regression variables

$$[u(t-1),\ldots,u(t-n_b),\hat{y}(t-1|\theta),\ldots,\hat{y}(t-n_f|\theta)] \tag{9.5}$$

Note that these regressors are partly constructed from the data, using a current model. As n_b and n_f tend to infinity, also this model is capable of describing all reasonable linear dynamic systems, but not the character of the additive noise $e(t)$. The advantage of (9.2) over (9.1) is that fewer regressors are normally required to get a good approximation. The disadvantage is that the minimization over θ becomes more complicated. Also, the stability of the predictor (9.3) depends on $F(q)$, and thus has to be monitored during the minimization.

A very common variant is the ARX model

$$A(q)y(t) = B(q)u(t) + e(t) \tag{9.6}$$

with the predictor

$$\begin{aligned} \hat{y}(t|\theta) \quad &= \quad -a_1y(t-1) - \ldots - a_{n_a}y(t-n_a) \\ &\quad\quad +b_1u(t-1) + \ldots + b_{n_b}u(t-n_b) \end{aligned} \tag{9.7}$$

thus using the regressors

$$[y(t-1),\ldots,y(t-n_a),u(t-1),\ldots,u(t-n_b)] \tag{9.8}$$

As shown, e.g., in [22] this structure is capable of describing all (reasonable) linear systems, including their noise characteristics, as n_a and n_b tend to infinity. The ARX model is thus a "complete" linear model from the black box perspective. The only disadvantage is that n_a and n_b may have to be chosen larger than the dynamics require, in order to accomodate the noise description. Therefore, a number of variants of (9.6) have been suggested, where the noise model is given "parameters of its own". The best known of these is probably the ARMAX model

$$A(q)y(t) = B(q)u(t) + C(q)e(t) \tag{9.9}$$

Its predictor is given by

$$\begin{aligned} \hat{y}(t|\theta) \quad &= \quad (c_1 - a_1)y(t-1) + \ldots \\ &+ \quad (c_n - a_n)y(t-n) \\ &+ \quad b_1u(t-1) + \ldots + b_nu(t-n) \\ &+ \quad c_1\hat{y}(t-1|\theta) + \ldots + c_n\hat{y}(t-n|\theta) \end{aligned} \tag{9.10}$$

It thus uses the regression vector (9.8) complemented with past predictors, just as in (9.5) (although the predictors are caclulated in a different way).

A large family of black box linear models is treated, e.g., in [21]. It has the form

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t) + \frac{C(q)}{D(q)}e(t) \tag{9.11}$$

The special case $A(q) = 1$ gives the well known Box-Jenkins (BJ) model. The regressors used for the corresponding predictor are given, e.g., by equation (3.114) in [21]. These regressors are based on $y(t-k)$, $u(t-k)$, the predicted outputs $\hat{y}(t-k|\theta)$ using the current model, as well as the simulated outputs $\hat{y}_u(t-k|\theta)$, which are predicted outputs based on an output error model (9.2).

Let us repeat that from a black box perspective, most variants of (9.11) are equivalent, in the sense that they can be transformed into each other, at the expense of changing the orders of the polynomials. The ARX model (C=D=F=1) covers it all. The rationale for the other variants is that we may come closer to the true system using fewer regressors.

9.2 Choice of Regressors for Neural Network Models

The discussion on linear systems clearly points to the possible regressors:

- Past Inputs $u(t-k)$
- Past Measured Outputs $y(t-k)$
- Past Predicted Outputs, using current model, $\hat{y}(t-k|\theta)$
- Past Simulated Outputs, using past inputs only and current model, $\hat{y}_u(t-k|\theta)$

A rational question to ask would be: Given that I am prepared to use m regressors (the size of the input layer is m), how should I distribute these over the four possible choices? There is no easy and quantitative answer to this question, but we may point to the following general aspects:

- Including $u(t-k)$ only, requires that the whole dynamic response time is covered by past inputs. That is, if the maximum response time to any change in the input is Υ, and the sampling time is T, then the number of regressors should be Υ/T. This could be a large number. On the other hand, models based on a finite number of past inputs cannot be unstable in simulation, which often is an advantage.
  A variant of this approach is to form other regressors from u^t, e.g., by Laguerre filtering, (e.g., [39]). This retains the advantages of the FIR-approach, at the same time as making it possible to use fewer regressors. It does not seem to have been discussed in the NN-context yet.
- Adding $y(t-k)$ to the list of regressors makes it possible to cover slow responses with fewer regressors. A disadvantage is that past outputs bring in past disturbances into the model. The model is thus given an additional task to also sort out noise properties. A model based on past outputs may also be unstable in simulation from input only. This is caused by the fact that the past measured outputs are then replaced by past model outputs.

- Bringing in past predicted or simulated outputs $\hat{y}(t-k|\theta)$ typically increases the model flexibility, but also leads to non-trivial difficulties. For neural networks, using past outputs at the input layer gives *recurrent* networks. See Section 5. Two problems must be handled:
  - It may lead to instability of the network, and since it is a non-linear model, this problem is not easy to monitor.
  - The simulated/predicted output depends on θ. In order to do updates in (6.2) in the true gradient direction, this dependence must be taken into account, which is not straightforward. If the dependence is neglected, convergence to local minima of the criterion function cannot be guaranteed.

The balance of this discussion is probably that the regressors (9.8) should be the first ones to test.

9.3 Neural Network Dynamic Models

Following the nomencalture for linear models it is natural to coin similar names for Neural Network models. This is well in line with, e.g., [5, 4]. We could thus distinguish between

- *NNFIR*-models, which use only $u(t-k)$ as regressors
- *NNARX*-models, which use $u(t-k)$ and $y(t-k)$ as regressors
- *NNOE*-models, which use $u(t-k)$ and $\hat{y}_u(t-k|\theta)$
- *NNARMAX*-models, which use $u(t-k), y(t-k)$ and $\hat{y}(t-k|\theta)$
- *NNBJ*-models, which use all the four regressor types.

In [25] another notation is used for the same models. The NNARX model is called Series-Parallel model and the NNOE is called Parallel model.

From a structural point of view, these black-box models are just slightly more troublesome to handle than their linear counterparts. When the regressor has been decided upon, it only remains to decide how many hidden units which should be used. The linear ARX model is entirely specified by three structural parameters $[n_a\ n_b\ n_k]$. [n_k is here the delay, which we have taken as 1 so far. In general we would work with the input regressors $u(t-n_k), \ldots, u(t-n_k-n_b+1)$.] The NNARX model has just one index more, $[n_a\ n_b\ n_k\ n_h]$, where n_h is the number of units in the hidden layer which in some way corresponds to "how non-linear" the system is. The notation for NNOE and NNARMAX models follow the same simple rule.

If more then one hidden layer is used there will be one additional structural parameter for each layer.

It follows from Section 5.2 that NNOE, NNBJ, and NNARMAX correspond to recurrent neural nets because parts of the input to the net (the regressor) consist of past outputs from the net. As pointed out before, it is in general harder to work with recurrent nets. Among other things, it becomes difficult to check under what conditions the obtained model is stable, and it takes an extra effort to calculate the correct gradients for the iterative search.

9.4 Some Other Structural Questions

The actual way that the regressors are combined clearly reflect structural assumptions about the system. Let us, for example, consider the assumption that the system disturbances are additive, but not necessarily white noise:

$$y(t) = g(u^t) + v(t) \tag{9.12}$$

Here u^t denotes all past inputs, and $v(t)$ is a disturbance, for which we only need a spectral description. It can thus be described by

$$v(t) = H(q)e(t)$$

for some white sequence $\{e(t)\}$. The predictor for (9.12) then is

$$\hat{y}(t) = (1 - H^{-1}(q))y(t) + H^{-1}(q)g(u^t) \tag{9.13}$$

In the last term, the filter H^{-1} can equally well be subsumed in the general mapping $g(u^t)$. The structure (9.12) thus leads to a NNFIR or NNOE structure, complemented by a *linear* term containing past y.

In [25] a related Neural Network based model is suggested. It can be described by

$$\hat{y}(t) = f(\theta_1, \varphi_1(t)) + g(\theta_2, \varphi_2(t)) \tag{9.14}$$

where $\varphi_1(t)$ consists of delayed outputs and $\varphi_2(t)$ of delayed inputs. The parameterized functions f and g can be chosen to be linear or non-linear by a neural net. A further motivation for this model is that it becomes easier to develop controllers from (9.14) than from the models discussed earlier.

In [24], it is suggested first to build a linear model for the system. The residuals from this model will then contain all unmodelled non-linear effects. The Neural Net model could then be applied to the residuals (treating inputs and residuals as input and output), to pick up the non-linearities. This is attractive, since the first step to obtain a linear model is robust and often leads to reasonable models. By the second Neural Net step, we are then assured to obtain at least as good a model as the linear one.

The question of how many layers to use is not easy. The paper [34] contains many useful and interesting insights into the importance of second hidden layers in the NN structure. See also the comments on this in Section 5.1.

9.5 The Identification Procedure

A main principle in identification is the rule *try simple things first*. The idea is to start with the simplest model which has a possibility to describe the system and only to continue to more complex ones if the simple model does not pass validation tests.

When a new more complex model is investigated the results with the simpler model give some guidelines how the structural parameters should be chosen in the new model. For example, it is common to start with an ARX model. The delay and number of delayed inputs and outputs give a good initial guess how the structure parameters should be chosen for the more complex ARMAX model. In this way less combinations of structural parameters have to be tested and computer time is saved.

Many non-linear systems can be described fairly well by linear models and for such systems it is a good idea to use insights from the best linear model how to select the regressors for the NN model. To begin with, only the number of hidden units the needs to be varied. Also, there might be more problems with local minima for the non-linear than for the linear models which makes it necessary to do several parameter estimates with different initial guesses. This further limits the number of candidate models which can be tested.

In the following example a hydraulic actuator is identified. First a linear model is proposed which does not capture all the fundamental dynamical behavior and then a NNARX model is tried. The same problem is considered in [2] using wavelets as model structure.

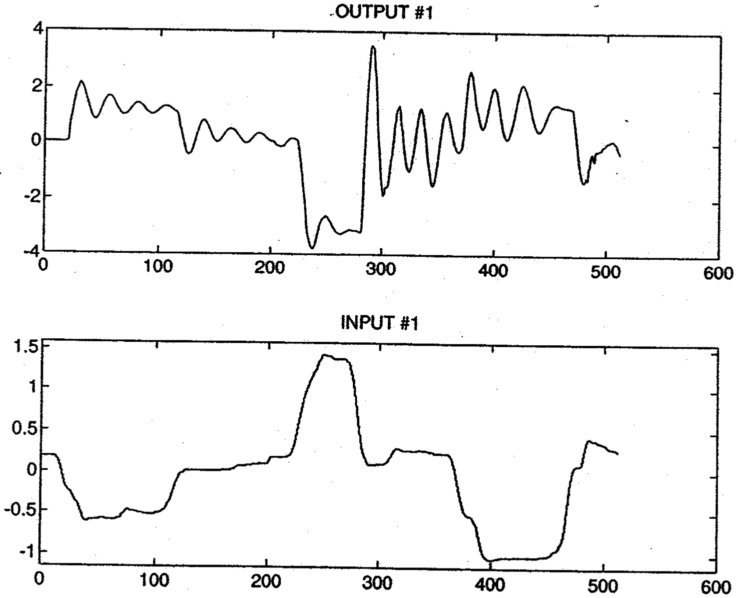

Fig. 9.1. Measured values of oil pressure (top) and valve position (bottom).

Example 9.1. Modeling a Hydraulic Actuator. The position of a robot arm is controlled by a hydraulic actuator. The oil pressure in the actuator is controlled by the size of the valve opening through which the oil flows into the actuator. The position of the robot arm is then a function of the oil pressure. In [13] a thorough description of this particular hydraulic system is given. Fig. 9.1 shows measured values of the valve size and the oil pressure, which are input- and output signals, respectively. As seen in the oil pressure, we have a very oscillative settling period after a step change of the valve size. These oscillations are caused by mechanical resonances in the robot arm.

Following the principle "try simple things first" gives an ARX model with structural parameters $[n_a\ n_b\ n_k] = [3\ 2\ 1]$. In Fig. 9.2 the result of a simulation with the obtained linear model on validation data is shown. The result is not very impressive.

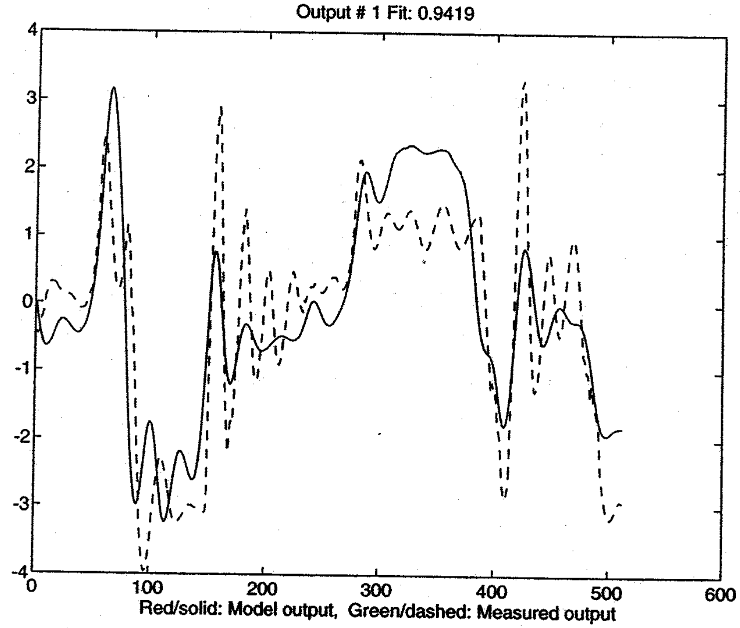

Fig. 9.2. Simulation of the linear model on validation data. Solid line: simulated signal. Dashed line: true oil pressure.

Instead a NNARX model is considered with the same regressor as the linear model, i.e., with the same first three structural indexes, and with 10 hidden units, $n_h = 10$. In Fig. 9.3 it is shown how the quadratic criterion develops during the estimation for estimation and validation data, respectively. For the validation data the criterion first decrease and then it starts to increase again. This is the overtraining which was described in Section 6.3. The best model is obtained at the minimum and this means that not all parameters in the non-linear model have converged and, hence, the "efficient number of parameters" is smaller than the dimension of θ.

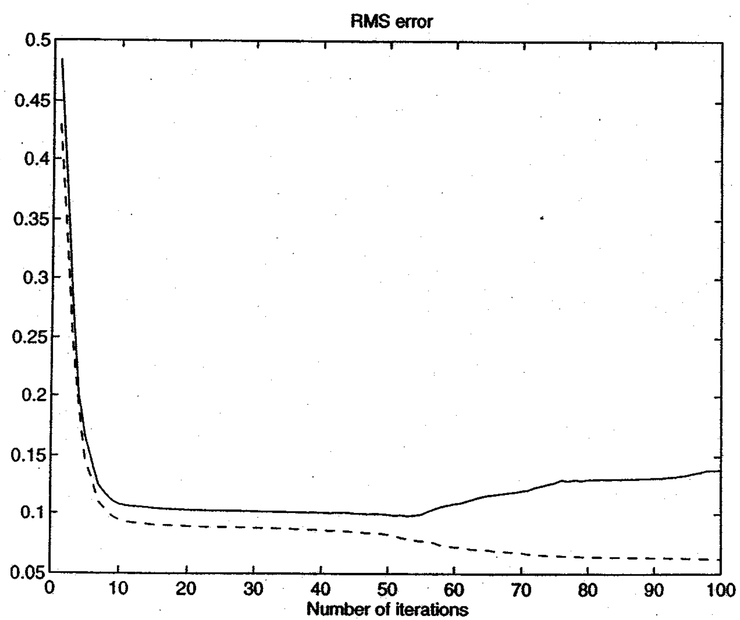

Fig. 9.3. Sum of squared error during the training of the NNARX model. Solid line: Validation data. Dashed line: estimation data.

The parameters which give the minimum are then used in the non-linear model and in Fig. 9.4 this NNARX model is used for simulation on the validation data.

This model performs much better than the linear model and it is compatible to the result obtained with a wavelet model in [2].

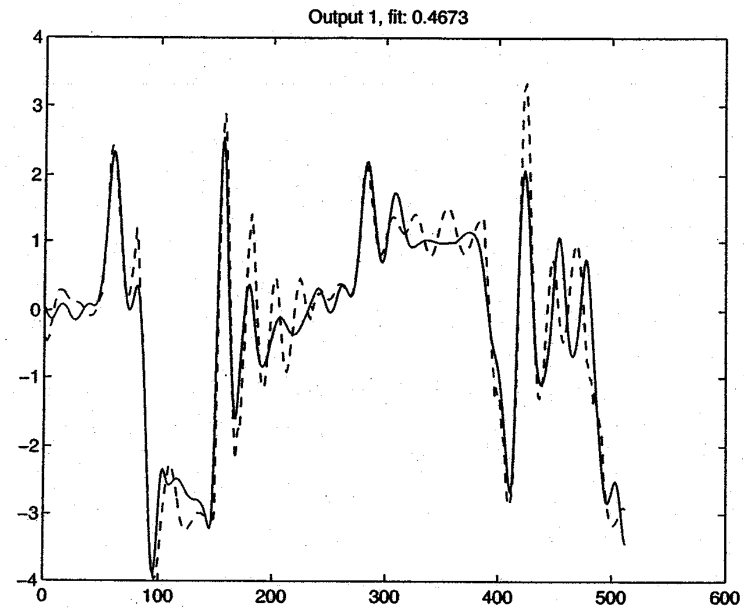

Fig. 9.4. Simulation of the non-linear model on validation data. Solid line: simulated signal. Dashed line: true oil pressure.

References

1. Barron A.R., "Universal approximation bounds for superpositions of a sigmoidal function", *IEEE Trns. Information Theory.*, IT-39:930–945, May 1993.
2. Benveniste A., A. Juditsky, B. Delyon, Q. Zhang, and P-Y. Glorennec, "Wavelets in identification", in *Preprint of the 10th IFAC Symposium on Identification*, 1994. (Copenhagen, 4–6 July).
3. Breiman L., "Hinging hyperplanes for regression, classification and function approximation", *IEEE Trns. Information Theory.*, IT-39:999–1013, May 1993.
4. Chen S., and S.A. Billings, "Neural networks for nonlinear dynamic system modelling and identification", *Int. J. Control*, 56(2):319–346, 1992.
5. Chen S., S.A. Billings, and P.M. Grant, "Non-linear system identification using neural networks", *Int. J. Control*, 51(6):1191–1214, 1990.
6. Cybenko G., "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.
7. Dennis J.E., and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
8. Diaconis P., and M. Shahshahani, "On nonlinear functions of linear combinations", *SIAM J. Sci. Statist. Comput.*, 5:175–191, 1984.
9. Donoho D.L., and I.M. Johnstone, "Projection-based approximation and a duality with kernel methods", *Ann. Statist.*, 17:58–106, 1989.
10. Donoho D.L., and I.M. Johnstone, "Minimax estimation via wavelet shrinkage", Technical report, Dept. of Statistics, Stanford University, 1992.
11. Friedman J.H., and W. Stuetzel, "Projection pursuit regression", *J. Amer. Statist. Assoc.*, 76:817–823, 1981.
12. Girosi F., and G. Anzellotti, "Convergence rates of approximation by translates", Technical report, Dept. Math. Art. Intell. Lab, MIT, 1992.
13. Gunnarsson S. and P. Krus, "Modelling of a flexible mechanical system containing hydralic actuators", Technical report, Dep. of Electrical Engineering, Linköping University, S-581 83 Linköping, Sweden, April 1990.
14. Guo L., and L. Ljung, "The role of model validation for assessing the size of the unmodelled dynamics", Technical report, Report LiTH-ISY, Department of Electrical Engineering, Linköping University, Sweden, 1994.
15. Hertz J., A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.
16. Hunt K.J., D. Sbarbaro, R. Żbikowski, and P.J. Gawthrop, "Neural networks for control systems - a survey", *Automatica*, 28(6):1083–1112, Nov. 1992.
17. Jones L.K., "A simple lemma on greedy approximations in Hilbert space and convergence rates for projection pursuit regression and neural network training", *The Annals of Statistics*, 20:608–613, 1992.
18. Kung S.Y., *Digital Neural Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
19. Leshno M., V.Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward netwoks with a non-polynomial activation function can approximate any function", *Neural Networks*, 6:861–867, 1993.
20. Ljung L., *System Identification: Theory for the User*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
21. Ljung L., and T. Söderström, *Theory and Practice of Recursive Identification*. MIT Press, Cambridge, MA, 1983.
22. Ljung L., and B. Wahlberg, "Asymptotic properties of the least-squares method for estimating transferfunctions and disturbance spectra", *Adv. Appl. Prob.*, 24:412–440, 1992.

23. Matthews M.B., "On the Uniform Approximation of Nonlinear Discrete-Time Fading-Memory Systems using Neural Network Models", PhD thesis, ETH, Zürich, Switzerland, 1992.
24. McAvoy T.J., 1992. Personal communication.
25. Narendra K.S., and K. Parthasarathy, "Identification and control of dynamical systems using neural networks", *IEEE Trans. Neural Networks*, 1:4–27, 1990.
26. Pinkus A., *n-Widths in Approximation Theory*. Springer, Berlin, 1985.
27. Polyak B.T., and A. Juditsky, "Acceleration of stochastic approximation by averaging", *SIAM J. Control*, 30(4):838–855, 1992.
28. Rumelhart D.E., G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors", *Nature*, 323(9):533–536, October 1986.
29. Rumelhart D.E., and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge MA, 1986.
30. Sjöberg J., and L. Ljung, "Overtraining, regularization, and searching for minimum in neural networks", in *Preprint IFAC Symposium on Adaptive Systems in Control and Signal Processing*, pages 669–674, Grenoble, France, 1992.
31. Söderström T., and P. Stoica, *System Identification*. Prentice-Hall International, Hemel Hempstead, UK, 1989.
32. Solbrand G., A. Ahlén, and L. Ljung, "Recursive methods for off-line identification", *Int. J. Control*, 41:177–191, 1985.
33. Sontag E.D., "Feedback stabilization using two-hidden-layer nets", technical report, Report SYCON-90-11, Rutgers Center for Systems and Control, Dept. of Mathematics, Rutgers University, New Brunswick, NJ, October 1990.
34. Sontag E.D., "Neural networks for control", in H.L. Trentelman and J.C. Willems, editors, *Essays on Control: Perspectives in the Theory and its Applications*, volume 14 of *Progress in Systems and Control Theory*, pages 339–380. Birkhäuser, Basel, 1993.
35. Triebel H., *Theory of Function Spaces*. Birkhäuser, Basel, 1983.
36. van der Smagt P.P., "Minimisation methods for training feedforward neural networks", *Neural Networks*, 7(1):1–11, 1994.
37. Wahba G., "Three topics in ill-posed problems", in H.W. Engl and C.W. Groetsch, editors, *Inverse and Ill-posed Problems*. Academic Press, 1987.
38. Wahba G., *Spline Models for Observational Data*. SIAM, University City Science Center, Philadelphia, PA, 1990.
39. Wahlberg B., "System identification using laguerre models", *IEEE Tabx. AC*, 36(5):551–562, May 1991.
40. Werbos P., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science*. Ph.D. thesis, Harvard University, 1974.
41. White D.A., and D.A. Sofge, editors, *Handbook of Intelligent Control, Neural, Fuzzy, and Adaptive Approaches*. Van Nostrand Reinhold, New York, 1992.
42. Miller III W.T., R.S. Sutton, and P.J. Werbos, editors, *Neural Networks for Control*. Neural Network Modeling and Connectionism. MIT Press, 1992. Series editor: J.L. Elman.
43. Zhao Y., *On projection pursuit learning*. Ph.D. thesis, Dept. Math., AI Lab. MIT, 1992.