Diss. ETH No. 10991

# THE DESIGN OF NEURAL NETWORKS USING A PRIORI KNOWLEDGE

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by
Rico A. Cozzio-Büeler
dipl. Informatik-Ing., ETH, Zurich
born July 6, 1965
citizen of Uster, ZH, Switzerland

accepted on the recommendation of
Prof. Dr. E. Engeler, examiner
Dr. J. Bernasconi, co-examiner

1995

*Sandra in Dankbarkeit und Liebe gewidmet, für die Geduld und Unterstützung, die ich immer erfahren durfte.*

# ACKNOWLEDGMENT

Thank you Prof. Dr. Erwin Engeler for your generous support of this work during the last four years. I am very indebted to you for your continuous support of industrial research and for your confidence in my work. Your patient and competent supervision has given me the chance to learn many new things.

Thank you Dr. Jakob Bernasconi for your support and for your precise supervision of this dissertation. I enjoyed many instructive and inspiring conversations that have given me deeper insight into the field of neural networks. It has been a great pleasure working with you.

Thank you Dr. Georg Maier, Dr. Paolo Conti and Dr. Hiro Sugaya from ABB Corporate Research, Baden, for giving me the opportunity of carrying out this work. The excellent working conditions and computer resources available at ABB have been very valuable to me.

Thank you Erich Scheiben, Dr. Bernhard Bachmann, Dr. Hans-Jürg Wiesmann and Dr. Jakob Rhyner for many stimulating discussions in a productive and open atmosphere. Thank you Jan Fülscher for your helpful comments. Thank you Mrs. Scheuber, Mrs. Schneider and Mrs. Gerber for processing my huge amount of library orders. Your patient and faithful cooperation has saved me a lot of time.

Thank you Mom and Dad for making it possible for me to get a good education. I am very grateful that I was always able to follow my interests. Thank you Maria and Werner Büeler for your encouraging support during the last years, which was very motivating.

Thank you Sandra for your love and patience. You created a warm and encouraging atmosphere, where you always tried to relieve me of everything that wasted my time. I enjoyed your continuous support in times of confidence and in times of doubt. You always motivated me to concentrate on my work, although much of our time together was taken away.

# CONTENTS

**1. Introduction** **1**

**2. Neural Networks for Industrial Applications** **4**

2.1 Basic Architectures ..... 4

2.2 Typical Applications ..... 9

2.2.1 Pattern Recognition and Classification ..... 9

2.2.2 Nonlinear Modeling ..... 10

2.2.3 Filtering ..... 10

2.2.4 Optimization ..... 11

2.3 Problems ..... 11

**3. Neural Network Design using a priori Knowledge** **14**

3.1 Propositional Rules ..... 14

3.2 Fuzzy Rules ..... 15

3.3 Functional Properties ..... 15

3.4 Finite Automata ..... 17

**4. Approximation of Differential Equations using Neural Networks** **18**

4.1 Motivation ..... 18

4.2 Local Approximation by Taylor Series ..... 19

4.3 Generalization of the Taylor Series Method ..... 20

4.4 Choice of Suitable Neural Network Approximators ..... 24

4.4.1 Modified Logistic Networks ..... 25

4.4.2 Radial Basis Function Networks ..... 28

4.5 Transformation of Differential Equations into Approximable Form ..... 34
4.6 Single-Step and Multi-Step Integration Procedures ..... 40
4.6.1 Modified Logistic Networks ..... 43
4.6.2 Radial Basis Function Networks ..... 45
4.7 Summary of the Neural Network Design Algorithm ..... 46
4.7.1 Single-Step Integration Procedure ..... 46
4.7.2 Multi-Step Integration Procedure ..... 48
4.7.3 Examples ..... 50
Example 1: First Order Differential Equation ..... 50
Example 2: Second Order Differential Equation ..... 52
4.8 Training Designed Neural Networks with Observation Data ..... 55
4.8.1 Improving the Modeling Accuracy of a Network ..... 55
4.8.2 Estimating Unknown Parameters of Differential Equations ..... 58

**5. Application to Forecasting of Chaotic Time Series 60**

5.1 Lorenz System ..... 61
5.1.1 One-Step-Ahead Forecasts ..... 65
5.1.2 Repeated Forecasts ..... 68
5.1.3 Learning of Unknown Parameters ..... 72
5.2 Rössler System ..... 76
5.2.1 One-Step-Ahead Forecasts ..... 80
5.2.2 Repeated Forecasts ..... 81
5.2.3 Learning of Unknown Parameters ..... 82
5.3 Circular Pendulum ..... 83
5.3.1 One-Step-Ahead Forecasts ..... 91
5.3.2 Repeated Forecasts ..... 92
5.3.3 Learning of Unknown Parameters ..... 94

**6. Conclusions 96**

**7. Bibliography** **100**

**8. Appendix** **114**

8.1 Example MAPLE Program for the Lorenz System ..... 114

8.2 Description of Implemented Software ..... 119

8.2.1 Odesim ..... 119

8.2.2 Predictnet ..... 120

8.2.3 Trainnet ..... 120

**List of Symbols** **122**

**List of Figures** **123**

**List of Tables** **125**

# ABSTRACT

We investigate methods for the design of neural networks using application specific knowledge. The standard approach of problem solving with neural networks reduces a problem to an approximation or classification task, which is solved by learning from observation data. However, application specific knowledge that is not available as observation data is usually ignored. This inefficient usage of information cannot be afforded for complex problems. An integration of both observation data and a□priori knowledge into neural networks should be tried. We show that using a□priori knowledge for the design of neural networks helps to solve some basic difficulties encountered in practice: Inefficient training and bad generalization of neural networks. For that purpose, different types of a□priori knowledge are identified and corresponding design methods are presented.

An important type of a□priori knowledge are differential equations, which are the main mathematical description tool used in engineering. We develop a method that allows to exploit ordinary differential equations for the design of neural networks. This algorithm determines the structure as well as the weights of a network. It is based on a generalization of the Taylor series method for the approximation of differential equations and it generates a set of equations for the network weights. We study the necessary conditions that enforce solvability of these equations and we show that both the class of Radial Basis Function Networks and the class of Modified Logistic Networks satisfy these conditions.

The architecture of the resulting neural networks depends on the form of the differential equations. A set of transformation rules is given that allows to transform most ordinary differential equations into a form suitable for the design algorithm. If the differential equations are linear, the standard neural network architecture can be maintained, whereas if the differential equations are in general form, a polynomial preprocessing layer has to be added to the network.

Our design algorithm generates neural networks that approximate the given differential equations. We test their approximation quality by forecasting some time series, which are generated by periodic sampling of chaotic differential equations. We examine the Lorenz system, the Rössler system and the circular pendulum system of differential equations. Both one-step-ahead forecasts and repeated forecasts are verified. It turns out that the Lorenz system and the Rössler system can be approximated well, whereas the circular pendulum system proves to be more difficult. In all cases, the Modified Logistic Networks are superior to the Radial Basis Function Networks.

Since the main purpose of our design algorithm is to integrate both a□priori knowledge and observation data into neural networks rather than competing with numerical integration algorithms, we investigate opportunities to exploit the learning capabilities of the networks after their initial design. If a network is constructed on the basis of incomplete differential equations, we can reduce the remaining modeling defects by additional learning. Alternatively, if the differential equations contain unknown parameters, they can be estimated from observation data using neural network training algorithms.

We test the latter aspect by estimating the parameters of the Lorenz system, the Rössler system and the circular pendulum system. A simulated annealing algorithm is applied to minimize the one-step-ahead forecasting error of a neural network. Although we observe good error minimization, the resulting parameter estimations match the correct parameters only if they coincide with the global minimum of the forecasting error. Consequently, if accurate parameter estimations are desired, the neural network must be constructed so that sufficient forecasting quality is guaranteed.

# KURZFASSUNG

Wir untersuchen Methoden für den Entwurf von neuronalen Netzwerken mit Hilfe von Anwendungswissen. Der übliche Lösungsansatz beim Einsatz von neuronalen Netzwerken besteht darin, ein Problem zuerst auf ein Approximations- oder Klassifikationsproblem zu reduzieren und anschließend durch Lernen aus Beispielen zu lösen. Anwendungswissen, das nicht in Form von Lerndaten zur Verfügung steht, wird dabei meistens ignoriert. Dieses Vorgehen ist bei komplexen Problemen unbefriedigend, da vorhandenes Anwendungswissen ineffizient genutzt wird. Es sollte eine Integration von Lerndaten und Anwendungswissen in neuronalen Netzwerken angestrebt werden. Gerade die Berücksichtigung von Anwendungswissen beim Entwurf von neuronalen Netzwerken kann grundlegende Schwierigkeiten beim Einsatz der Netzwerke erleichtern: Ineffizientes Lernverhalten sowie ungenügende Verallgemeinerung. Dies wird anhand verschiedener Klassen von Anwendungswissen und entsprechender Entwurfsmethoden für neuronale Netze vorgeführt.

Eine wichtige Klasse von Anwendungswissen sind Differentialgleichungen, welche das bevorzugte Werkzeug zur Beschreibung von Problemwissen in industriellen Anwendungen darstellen. Wir entwickeln eine Methode für den Entwurf von neuronalen Netzwerken mit Hilfe von gewöhnlichen Differentialgleichungen. Der Algorithmus bestimmt sowohl die Struktur als auch die Gewichte eines Netzwerkes. Er basiert auf einer Verallgemeinerung der Entwicklung einer Taylorreihe zur Approximation von Differentialgleichungen und erzeugt ein Gleichungssystem für die Gewichte des Netzwerkes. Wir untersuchen die Bedingungen, die für die Lösbarkeit des Gleichungssystems notwendig sind, und wir zeigen, daß sowohl die sogenannten "Radial-Basis-Function"-Netzwerke als auch die "Modified-Logistic"-Netzwerke diese Bedingungen erfüllen.

Die Architektur der konstruierten neuronalen Netzwerke hängt von der Form der Differentialgleichungen ab. Mit Hilfe von Transformationsregeln können die meisten gewöhnlichen Differentialgleichungen auf eine Form

gebracht werden, die für den Konstruktionsalgorithmus geeignet ist. Falls es sich um lineare Differentialgleichungen handelt, so kann die Standardarchitektur der Netzwerke beibehalten werden. Haben die Gleichungen jedoch eine allgemeine Form, so wird eine zusätzliche vorverarbeitende Schicht im Netzwerk benötigt.

Unser Entwurfsalgorithmus erzeugt neuronale Netzwerke zur Approximation von gegebenen Differentialgleichungen. Wir testen die Genauigkeit solcher Näherungen anhand der Vorhersage einiger Zeitreihen, die durch periodisches Abtasten chaotischer Differentialgleichungen erzeugt wurden. Die Vorhersagegüte wird am Beispiel des Systems von Lorenz und des Systems von Rössler, sowie an einem Kreispendel geprüft. Dabei werden die Vorhersagen sowohl nach einem einzelnen Zeitschritt als auch nach einer beliebigen Anzahl Schritte betrachtet. Es zeigt sich, daß nicht nur das System von Lorenz sondern auch das System von Rössler gut approximiert werden können. Beim Kreispendel entstehen grössere Schwierigkeiten. In allen Fällen ist aber die Approximation der "Modified-Logistic"-Netzwerke derjenigen der "Radial-Basis-Function"-Netzwerke überlegen.

Da der Hauptzweck unserer Entwurfsmethode darin besteht, eine einfache Integration von Anwendungswissen und Lerndaten in einem neuronalen Netzwerk zu ermöglichen, ist es nicht unser Ziel, bestehende numerische Integrationsalgorithmen zu übertreffen. Wir untersuchen stattdessen Möglichkeiten, wie die Lernfähigkeiten der neuronalen Netzwerke ausgeschöpft werden können, nachdem sie mit unserer Methode konstruiert worden sind. Es bestehen dabei mehrere Möglichkeiten: Falls die Netzwerke aus unvollständigen Differentialgleichungen entstanden sind, so können die verbleibenden Modellierungsfehler durch zusätzliches Lernen verringert werden. Enthalten die Differentialgleichungen hingegen unbekannte Parameter, so können diese mit Hilfe von Lernalgorithmen für neuronale Netze geschätzt werden.

Wir testen die zweite Variante anhand der Schätzung der Parameter des Systems von Lorenz, des Systems von Rössler und des Kreispendels. Dabei verwenden wir eine Variante des "Simulated-Annealing"-Algorithmus zur Minimierung des Vorhersagefehlers der neuronalen Netzwerke. Obwohl der verbleibende Fehler sehr klein wird, stimmen die Parameterschätzungen mit den richtigen Parameterwerten nur dann überein, wenn sie mit dem

globalen Minimum der Fehlerfunktion zusammenfallen. Für eine gute Parameterschätzung muß dies deshalb durch eine ausreichende Vorhersagegüte des eingesetzten Netzwerkes sichergestellt werden.

# 1. INTRODUCTION

During the last decade, artificial neural networks have experienced increasing scientific and technical interest, which has resulted in a vast amount of studies on neural networks. Several achievements have revived research on neural networks, which was practically nonexistent before 1980: First, computer hardware development continues to provide us with increasing computing power for lower prices. This allows a growing scientific community to perform neural network simulations on cheap hardware. Secondly, theoretical progress concerning the training of neural networks has created new interest in "learning machines", which promise solutions to previously unsolvable problems.

The most prominent work which helped restarting the field of neural networks was the introduction of the "backpropagation"-algorithm in [Rume86a]. Research on "artificial neural networks" independent of neurobiology was started, which is mostly interested in abstract "neural" models and their information processing capabilities. Priority is given to engineering aspects, not to the biological plausibility of neural network models. In this thesis, we conform with this approach, treating the terms "neural networks" and "artificial neural networks" as synonyms.

The increased usage of neural networks for many fields of application revealed some success stories, but also some problems. The general euphoria was dampened by the discovery of the fact that neural networks are no "turn-key"-technology which allows an easy solution of complex problems without any application knowledge. Empirical experience, gained from successfully using neural networks for small benchmark problems, does often not scale up to large problems. Choosing the neural network architecture, selecting the best pre- and postprocessing of the data, and picking the optimal training algorithm becomes nontrivial for complex problems. If the wrong choices are taken, inefficient learning and bad generalization of the networks can result.

From the engineering point of view, this is a clear disadvantage, because the usage of engineering tools should be as much application independent as possible. Consequently, neural network research has always had a strong empirical component, which has been backed up by solid theoretical results only recently (e.g., see [Stin89], [Horn90], [Pogg90], [Park91] or [Park93]).

However, the usage of application specific knowledge is often the only way to reduce the complexity of a problem to a level where neural networks can be successfully applied. The traditional try-and-error approach to select the architecture and the size of a neural network should be replaced by a network design based on exploiting all available application knowledge. The main advantages of using a□priori knowledge for the design of neural networks should be smaller needs for learning data, improved training efficiency and better generalization. Since any information that is built directly into a network does not have to be learned any more, less learning data is needed and generalization is improved. Additionally, network training becomes more efficient because smaller networks can be used to learn the remaining learning data.

An important type of a□priori knowledge are differential equations, which are the main mathematical tool used for the description of process knowledge in many industrial applications. In this thesis, we present a new design method that allows to construct a neural network from a set of ordinary differential equations. The resulting network can then be used to forecast the process described by the differential equations. If observation data from the process is available, the remaining modeling defects can be reduced by additional network training. This allows to smoothly integrate both structured knowledge in the form of differential equations and raw observation data into a neural network model of the process.

This thesis is organized as follows: Chapter 2 introduces some neural network background. After a short description of neural network architectures, some typical applications and some practical problems of neural networks are presented. In chapter 3, different types of a□priori knowledge are identified and corresponding design methods for neural networks are discussed.

Chapter 4 is dedicated to our neural network design method based on ordinary differential equations: First, the method is derived by generalizing

the Taylor series method for the approximation of differential equations. The algorithm determines the structure of a network and it generates a set of equations for the network weights. Then, we study the necessary conditions that enforce solvability of these equations and we show that both the class of Radial Basis Function Networks and the class of Modified Logistic Networks satisfy these conditions. Afterwards, we investigate the influence of the differential equations onto the neural network architecture. We demonstrate that every ordinary differential equation that can be transformed into polynomial form can be approximated with neural networks designed by our method. Finally, we summarize the design algorithm and give some examples.

In chapter 5, we test the approximation quality of constructed neural networks by forecasting some time series, generated by chaotic differential equations. We discuss the Lorenz system, the Rössler system and a circular pendulum system of differential equations. For each system, the quality of one-step-ahead forecasts and of repeated forecasts is evaluated.

Since the main purpose of our design algorithm is to integrate both a□priori knowledge and observation data into neural networks, we investigate opportunities to exploit the learning capabilities of the networks after their initial design. One possibility is the estimation of unknown parameters of a differential equation using neural network training algorithms. The feasibility of this approach is tested by estimating the parameters of the Lorenz system, the Rössler system and the circular pendulum system.

In chapter 6, we draw some conclusions about the results of our experiments. Finally, chapter 7 contains the bibliography of referenced papers, which is followed by the appendix in chapter 8.

# 2. NEURAL NETWORKS FOR INDUSTRIAL APPLICATIONS

## 2.1 Basic Architectures

The brain is composed of about $10^{11}$ biological neurons of different types. Figure 2.1 describes the parts of a typical neuron. A number of tree-like fibers called *dendrites* are connected to a *cell body*, which contains the *nucleus*. A single long fiber called the *axon* is extending from the cell body and branches into several strands. These strands connect to other neurons via their *synapses*. Signals are transmitted from one cell to the other by releasing chemical substances from the sender, which raise or lower the electric potential relative to the receiving cell. After some threshold is reached, a short electric pulse is sent over the axon. The activation level of the sender neuron determines the frequency of electric pulses sent to all connected neurons. For an extended description, see [Hert91] or [Hayk94].

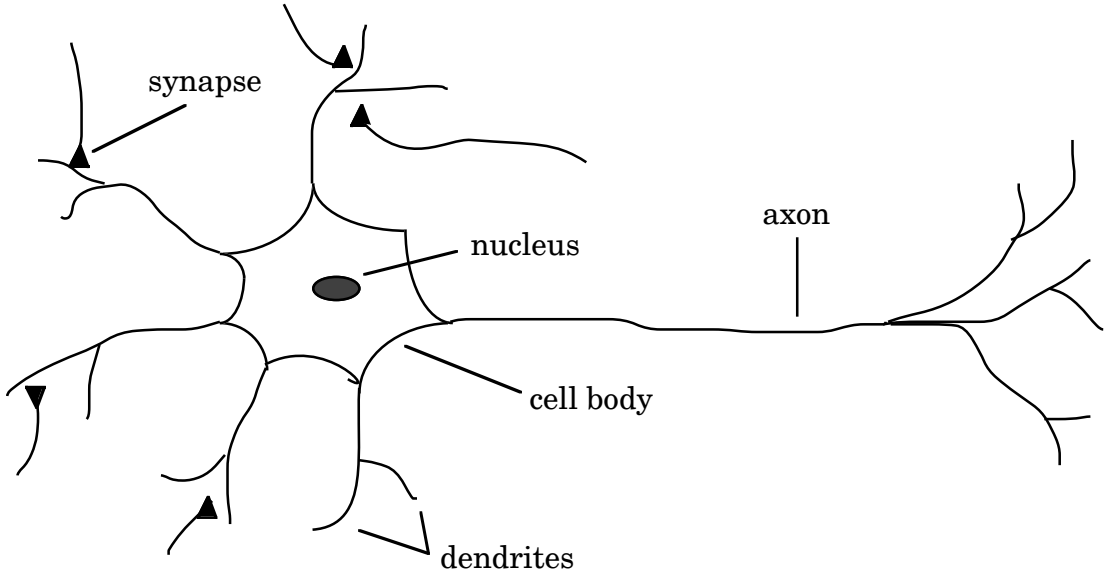



Fig. 2.1 : Biological neuron

Figure 2.2 shows an artificial neuron called *unit* or *node*, which is the basic processing element of artificial neural networks. Artificial neurons are simple mathematical models of biological neurons. In an artificial neuron, synapses correspond to *inputs* and synaptic connection strengths are modeled by *weights*.

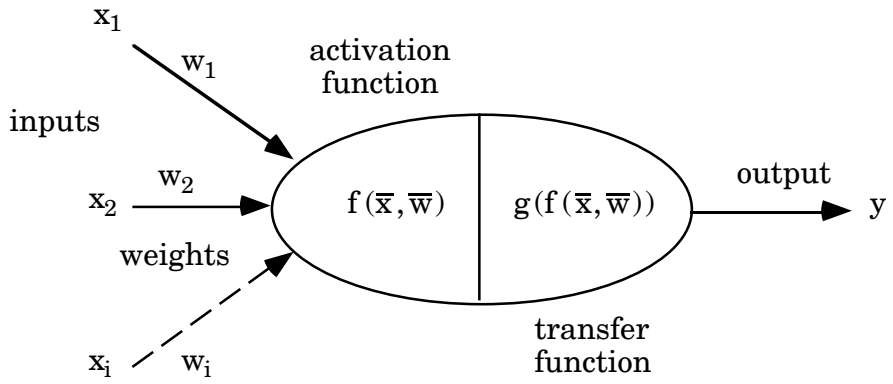



Fig. 2.2 : Artificial neuron

A network unit computes a function of its input signals and their associated weights. The computation proceeds in two steps: First, the *activation function* determines the unit's activation level. This is an approximation of the chemical processes that take place at the neuron's synapses. Then, the *transfer function* computes the *output* of the unit, which models the frequency of the pulses sent to all connected neurons. Here, the analogy between biological and artificial neurons usually stops. Since the properties of artificial neural networks are similar to biological networks with respect to their information processing capabilities, research concentrates on the study of these properties. Thus, artificial neural network research often does not care about biological plausibility.

The most important activation functions are listed in table 2.1. Networks using a polynomial activation function are called *higher-order networks*. In most networks, however, linear activation functions are used, which compute a weighted sum of the unit's inputs. Since linear activation functions are so dominant, there is often no explicit distinction between activation and transfer functions in the literature. In that case, a linear activation function is implicitly assumed.

| Type | Activation Function |
|---|---|
| Linear function | $f(\overline{x},\overline{w}) = \sum_i w_i \cdot x_i$ |
| Polynomial function | $f(\overline{x},\overline{w}) = \sum_i \sum_{j<i} \sum_k \sum_l x_i^{\,k} \cdot x_j^{\,l} \cdot w_{i,j,k,l}$ |

Table 2.1 : Typical activation functions

Table 2.2 lists the most frequently used transfer functions. For binary neural networks, the threshold function is common, whereas for networks with continuous outputs, sigmoid or Gaussian functions are usually applied.

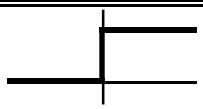
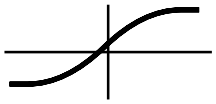
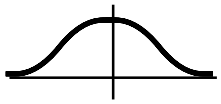
| Type | Transfer Function | Graph |
|---|---|---|
| Threshold function | $g(s) = \mathrm{sign}(s)$ |  |
| Sigmoid function | $g(s) = \tanh(s)$ |  |
| Gaussian function | $g(s) = e^{-s^2}$ |  |

Table 2.2 : Typical transfer functions

Artificial neurons are connected together in large networks, where many different architectures are possible. Figure 2.3 shows a *multilayer feedforward network*, which contains several layers of units. The units in one layer are connected to all units of the next layer, and no feedback connections are allowed. There is a layer of input units and a layer of output units. All other layers are called "*hidden*" *layers*, because they are invisible to the outside environment. These networks are also called *multilayer perceptrons*. They compute a static mapping from input to output, which depends on the network weights. Such networks are used for function approximation and classification problems.

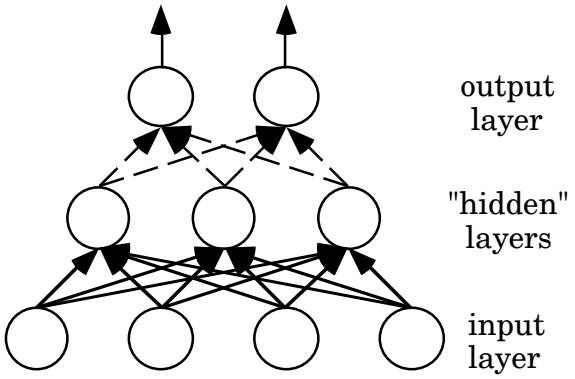



Fig. 2.3 : Multilayer feedforward network

Figure 2.4 shows a recurrent network, which contains feedback connections between units. The dynamics of such networks can be described by update rules or by differential equations. If the network operates in discrete time, the standard approach is to read its outputs only after it has converged to a stable state[1]. Such networks are mostly used for pattern recognition or optimization problems. If the network runs in real-time, however, outputs are generated continuously. These networks can be found in signal processing applications.

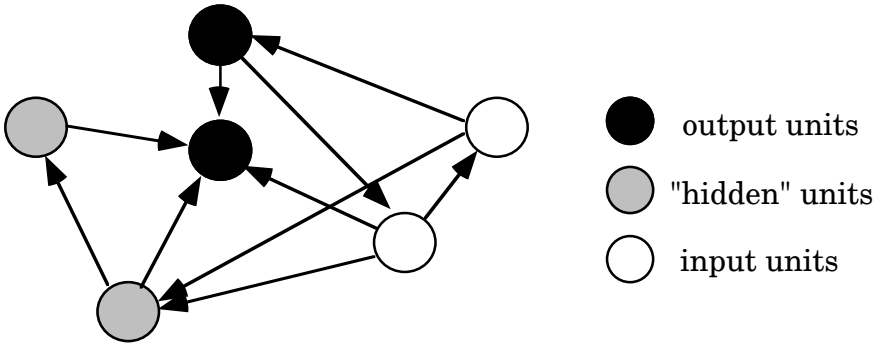



Fig. 2.4 : Recurrent network

The connection weights of the network units are the only place where neural networks can store information. If a network is supposed to perform a certain task, a set of weights has to be found which allows to implement the desired function. Algorithms for finding good network weights are called *learning* or *training algorithms*. They allow the network to approximate a set of target input-output mappings or to optimize some performance criterion.

---

[1] Convergence of dynamic networks can be enforced by constraining the network weights. In Hopfield networks, e.g., this is achieved by requiring symmetric connections (see [Hert91]).

Basically, there are three approaches to learning in neural networks:

1. A *teacher* provides examples of the desired input-output mapping of a neural network, which is called the *learning set*. A training algorithm tries to reduce the output error of a network by minimizing an error function like (2-1). This is called *supervised learning*. The most widely used supervised training algorithm is the error backpropagation algorithm described in [Rume86a].

$$E = \frac{1}{2}\sum_{i=1}^{N}\left\|\bar{d}_i - NN(\bar{x}_i, \bar{w})\right\|_2^2$$
$$\bar{x}_i = \text{input vector of example i}$$
$$\bar{d}_i = \text{desired output vector for example i} \qquad (2\text{-}1)$$
$$\bar{w} = \text{weights of neural network}$$
$$N = \text{number of learning examples}$$
$$NN(\bar{x}, \bar{w}) = \text{neural network output}$$

2. No teacher does exist who can tell the desired output for a given input to a neural network, but there is a critic who gives a *performance criterion* quantifying the quality of its current behavior (e.g., good or bad). This kind of learning problem is called *reinforcement learning* (see [Bart85] or [Bart91]).

3. There is neither a teacher nor a critic guiding the neural network, but only raw input data is given. The network should look for regularities in the data and provide some automatic classification. The purpose of these networks is to analyze the data and to provide some filtering or preprocessing for other neural networks. This kind of learning is called *unsupervised learning* (see [Rume86b], [Barl89] or [Koho89]).

From a mathematical perspective, most types of learning in neural networks can be reduced to optimization problems, where connection weights have to be found that optimize a given performance criterion. Nonetheless, what distinguishes neural network training from methods developed in operations research or optimization theory is the problem of *generalization*. Since for nontrivial problems, a learning set never specifies a function to be learned completely, the behavior of a network for inputs it has never seen is very important. The difference between learning and memorizing is the capability to extract relevant properties of the problem and to induce rules for generalizing solutions to similar but not equal

situations. In principle, a learning set can be approximated arbitrarily well by a sufficiently large neural network and with enough training. However, the art of neural network training manifests itself in good generalization to unknown inputs. This cannot be achieved by optimizing the performance on the learning set only. A standard approach to estimate the generalization of a network is the separation of the available data into a *training set* and a *test set*. Training takes place on the training set only, and the error on the test set shows the quality of the network's generalization. For an elaborated discussion on how to estimate the real performance of a learning system, see [Weis91].

There are many different artificial neural network architectures and learning algorithms we have not presented here. In addition to learning, other attractive properties of neural networks are the ability to process noisy or incomplete inputs, the slow performance degradation after network connections are damaged, and the high processing speed that can be achieved by massive parallelization of the networks. For a general discussion see [Lipp87], [Hint89], [Hech90], [Hert91] or [Hayk94].

## 2.2 Typical Applications

Many researchers regard neural networks as one step further towards the goals of Artificial Intelligence (AI). Consequently, they are used for nearly every possible application where traditional approaches (even symbolic AI methods) fail or better solutions are expected. Lists of typical applications can be found in the DARPA (Defense Advanced Research Projects Agency) neural network study [DARP88], in text books like [Hech90] or [Hert91], or in paper collections such as [Mood92] or [Dagl94]. The next sections give a short overview of the characteristics of neural network applications.

### 2.2.1 Pattern Recognition and Classification

Pattern recognition and classification problems are a traditional field of Artificial Intelligence, because those tasks are considered simple for humans but difficult for computers. Although many different techniques exist, neural networks are competitive in pattern recognition tasks like *speech recognition* (see [Lipp89], [Cole90]), *optical character recognition* (see [Jack88], [Guyo89], [Keel92]), *computer vision* (see [Mann90], [Intr92]), or

*signal classification* (see [Vcrk90]), where perfect classification is often impossible.

Other types of classification problems are typically covered by special software packages called *expert systems*. They include the knowledge of experts in a specific field in the form of a knowledge base, and it is possible to ask questions to the system and to get answers similar in quality to the advice of an expert. The first generation of these systems was based on purely symbolic knowledge such as "if-then-else" rules. Newer approaches like those presented by Gallant in [Gall93] propose the combination with approximate reasoning based on fuzzy logic or neural networks, where learning from data is also supported. Conventionally, problems like *fault diagnosis* have been attacked with expert systems. Today, they are often replaced by neural networks (see [Yama90]), where a prominent example can be found in medicine (see [Jabr92]).

### 2.2.2 Nonlinear Modeling

A generalization of classification problems to continuous domains leads to nonlinear modeling problems. Since most neural networks are universal function approximators (e.g., see [Cybe89], [Funa89]], [Stin89, [Giro90], [Hart90], [Horn90], [Park91], [Ito92], [Cott93], [Park93]), they are often preferred to alternative approximation functions like polynoms or splines.

Example applications can be found in *robotics* (see [DeMe92], [Ritt91]), *forecasting* of time series (see [Kimo90], [Weig90], [Wong91]) or *control* (see [Bart83], [Mill90], [Wang90] or [Whit92]).

### 2.2.3 Filtering

There are a few nonstandard applications of neural networks which can be described as filtering problems. In these applications, neural networks are trained to transform a given signal into a filtered version of the same signal according to a given goal. Most approaches train the networks in autoassociative mode, where the input equals the desired output. The network is then forced to extract only the most relevant features, because artificial bottlenecks created in the hidden layers prevent the network to implement the full input-output relation. Representative examples of this special type of applications are *noise reduction* in signal processing (see

[Cich93]) or *data compression* (see [Nasr88] or [Cott87], which is discussed in [Hech90], pp. 325).

### 2.2.4 Optimization

Another nonstandard type of neural network application is the optimization of a given cost function. Here, not the learning capabilities of a neural network are of interest, but the possibility to massively parallelize the execution of a network makes it attractive for the solution of optimization problems. First, the problem is mapped into an energy function of a recurrent network, which corresponds directly to the weights and the architecture of the network. Afterwards, the network is iterated from an initial configuration until it converges to a stable state, where the solution of the optimization problem is extracted.

Optimization problems typically attacked with neural networks share the following characteristics:

- The cost function can be transformed into an energy function of a recurrent neural network.
- Constraints to the solutions can be coded as penalty functions.
- Suboptimal solutions are acceptable (because the network may not find a global optimum).
- Quick results are desired.

Examples of this type of problems can be found in [Mats90], [Mjol90] or [Cich93].

## 2.3 Problems

Although neural networks have attractive properties for many applications, they show some basic difficulties as well:

1. In order to allow robust learning of a neural network, the number of training examples must be at least of the order of unknown weights in the network. If the data is noisy, however, the size of the training set must be significantly larger. Consequently, the

amount of data needed can become quite large already for a moderate complexity of a neural network model. This is a clear disadvantage for problems where collecting more experimental data is difficult or expensive.

2. As the size of a neural network grows and the number of training examples increases, the optimization of the network error becomes more difficult. Since the error surface generated by a neural network typically shows long, flat and narrow valleys (see [Hech90], pp. 115), training algorithms need many iteration steps to find good solutions. At the same time, the numerous local minima increase the probability to get stuck with gradient-based algorithms (see [Blum89], [Brad89], [Sont89] or [Gori92]), which is illustrated in figure 2.5. Consequently, large neural networks often suffer from inefficient learning.

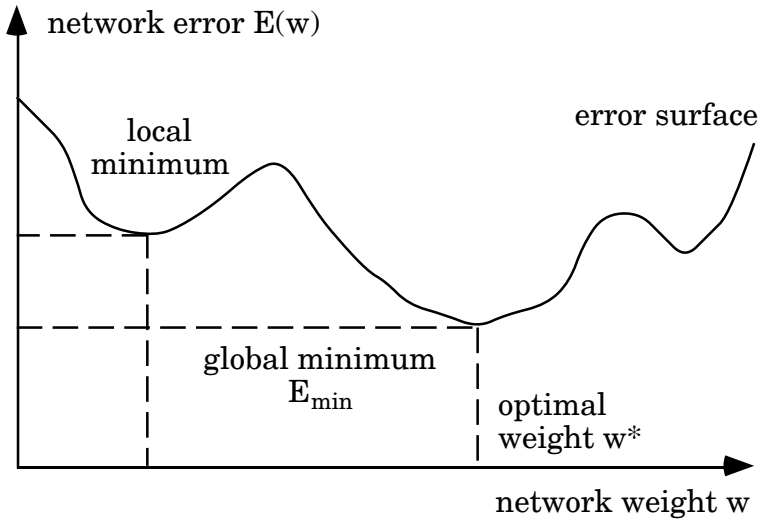



Fig. 2.5 : Typical error surface with local minima (one-dimensional case)

3. The problem of inefficient learning becomes less severe as soon as more powerful computers are available. However, the generalization of a neural network to inputs it has not seen before is much more important than optimal training. Since the network was trained to approximate the training examples only, generalization is directly related to its interpolation and extrapolation properties. Figure 2.6 shows the typical generalization behavior of a neural network: The approximation errors for the test set are bigger than for the training set, and the

interpolation quality of the network is better than its extrapolation quality.

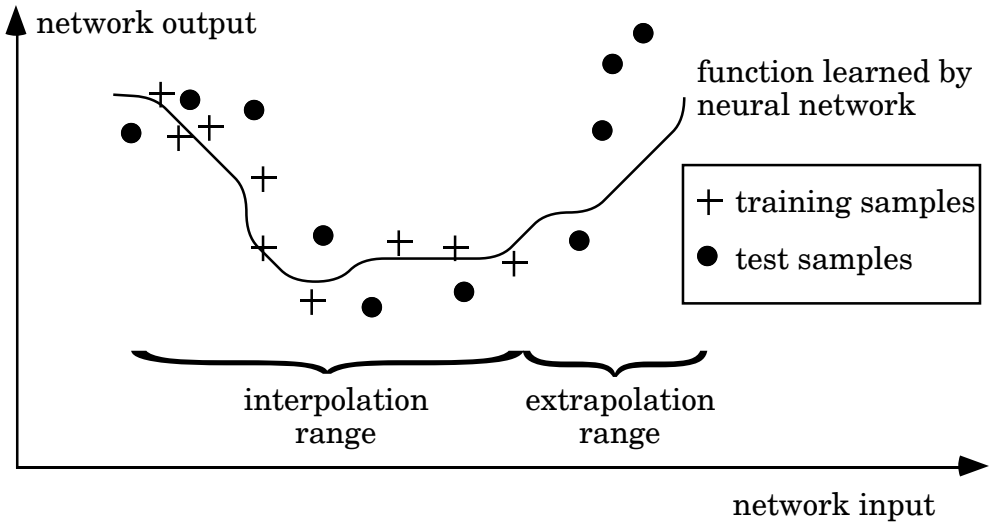



Fig. 2.6 : Generalization properties of a neural network

Although it is possible to reduce the training error of a neural network by increasing the number of network units, this does not necessarily improve the error on the test set. There is usually a tradeoff between network size and generalization error. After a certain optimal size of the network is exceeded, the network adapts too closely to the training examples, which causes deteriorating generalization. This phenomenon is called *overtraining*. The optimal network size and the generalization error can be estimated with statistical resampling techniques like cross-validation (see [Weis91], pp. 30, or [Twom93]).

Some of the problems mentioned above are strongly influenced by factors like the network architecture, the choice of data representation or the selection of the error function. Since good heuristics are often missing, results are very application specific, which explains the dominance of empirical research.

One possibility to overcome these problems is the usage of a□priori knowledge. We investigate its potential for improving the learning efficiency and the generalization of neural networks in the next chapter.

# 3. NEURAL NETWORK DESIGN USING A PRIORI KNOWLEDGE

The usage of application specific knowledge for the design or training of neural networks has been investigated by several authors. In this chapter, we distinguish different types of a□priori knowledge and we present some of the methods exploiting that knowledge in the context of neural networks.

## 3.1 Propositional Rules

When the connectionist computing paradigm became popular, researchers started looking for explanations of the brain's logical inference capabilities. Since neural networks claim to be a plausible (though simple) model of the brain, logical reasoning should be possible to implement. Several proposals for connectionist inference engines originated from this work (see [Tour85], [Ball86], [Shas87], [Dyer89a], [Dyer89b], [Kurf89], [Anan89], [Ajja89], [Ajja90], [Pink91], [Pina91], [Nara92]). They are based on energy minimization in Hopfield-style networks or Boltzmann machines, whose architecture and network weights are constructed from a knowledge base. In these studies, the knowledge base consists of propositional rules that are optionally enhanced with frames and inheritance. Inference is performed by relaxing the energy of the neural network until a local minimum is reached. Since the energy function of the network is constructed so that its minima correspond to legal inferences, the network works as a parallel inference engine. Priority is given to the power of the knowledge description language, to the problems of knowledge representation and to the efficiency of the inference engine. The learning capabilities of neural networks are not yet exploited in these studies.

In [Towe90], Towell, Shavlik and Noordewier employ a knowledge base used in explanation-based learning to determine the structure and weights of a neural network. Consequently, the knowledge base implemented by the

network becomes accessible to modification by neural learning. Although the knowledge is restricted to hierarchical non-recursive propositional (variable-free) domain theories, the authors show that the initialization of the network using a small knowledge base is able to improve learning and generalization. In [Towe91], Towell, Shavlik and Craven even extend their approach to extracting rules from the network after training (a similar approach is proposed in [Fu94]). This allows validation of the knowledge learned by the neural network, which is an attractive property not offered by most neural network architectures. Their approach allows easy integration of symbolic knowledge into neural networks, which is shared by other studies such as [Good92] or [Tsch93]. An extension from propositional rules to more general rules would be desirable. Some works on neural unification algorithms (see [Stol89], [H90a], [H90b] or [H90c]) show promising steps in these directions.

## 3.2 Fuzzy Rules

Since neural networks are approximate models, the step from exact propositional rules to approximate fuzzy rules is natural. In [Jang93], Jang proved the functional equivalence of Radial Basis Function Networks and fuzzy inference systems. If we combine this result with the universal approximation property of fuzzy basis functions demonstrated by Wang in [Wang92], the translation from fuzzy rules to neural networks becomes straightforward. Consequently, training of such networks can be done with standard learning algorithms, and often the networks can be transformed back into a fuzzy knowledge base after training. Variations of fuzzy neural network models differ only in the fuzzy membership functions and in the inference rules that are implemented in the network (see [Hori92] or [Taka92]). Sometimes, learning is performed by a clustering algorithm instead of a parameter estimation algorithm (e.g., see [Sulz93]). A general discussion of the relation between fuzzy systems and neural networks can be found in [Kosk91] and [Cox92].

## 3.3 Functional Properties

In [Abu-90], Abu-Mostafa suggests to use all available information about a function f to be learned during the training of a neural network. He calls specific information about f a *hint*. In his experiments, basic functional

properties like invariance (3-1) with respect to different input sets A are treated as hints:

$$\forall x_1, x_2 \in A \subset X: f(x_1) = f(x_2) \qquad (3\text{-}1)$$

Hints are integrated into a training algorithm by generating additional learning examples from the specification of the hint and by using a modified error function. In [Abu-93], he evaluates the effectiveness of hints by means of the Vapnik-Chervonenkis dimension (VC), which is used for estimating the number of examples needed for learning a given function. Although Abu-Mostafa's approach can help to improve generalization when there is insufficient data for proper network training, inappropriate usage of hints may slow or even cripple learning (e.g., by introducing local minima). This is pointed out by Al-Mashouq and Reed in [Al-M91], where they suggest to use the minimum Hamming distance between training examples as a hint in pattern recognition applications.

In [Sudd91], Suddarth investigates functional decomposition of the target function until only strictly monotonic functions remain to be learned by a network. This turns out to be much easier than learning general functions. He proposes to use a monotonic version of the target function as a hint, which should structure the hidden layer of the network in such a way that learning becomes easier. Rule injection into networks is also investigated, which is similar to the methods discussed in section 3.1. For both approaches, he reports dramatic improvement of learning time.

If the function to be learned is invariant with respect to some input transformations, there are other approaches to take this information into account than simply using it for the generation of additional learning examples. In [Barn91], Barnard and Casasent present a technique to build invariance into the structure of a network by constraining the network weights (e.g., by weight sharing). A network can be forced to satisfy symmetry or asymmetry of functions in similar ways. These methods are common in image recognition applications, where translation, rotation and scaling of objects have to be taken into account.

## 3.4 Finite Automata

Since the computational power of recurrent neural networks is equivalent to a Turing machine[2] (for a discussion, see [Poll87] or [Fran91]), they can easily implement finite automata. It is therefore natural to use such networks for the induction of unknown finite-state languages. In [Watr92]), Watrous and Kuhn apply second-order recurrent networks to learn a grammar from example sentences. After successful training, the grammar of the induced language can be extracted from the neural network, which is demonstrated in [Gile92]. If the grammar is partially known a□priori, it is possible to construct a neural network that incorporates the given grammar. In [Mins67]), Minsky proves that every finite-state machine can be simulated by a neural network, which is an attractive way to build a□priori knowledge into recurrent networks.

[2] Of course, this requires either infinite precision (e.g., rational) node activations or a network of infinite size to implement the tape of the Turing machine.

# 4. APPROXIMATION OF DIFFERENTIAL EQUATIONS USING NEURAL NETWORKS

## 4.1 Motivation

Today, differential equations are the main mathematical tool used in engineering. They are the prevalent technique for modeling dynamic systems, because the causal relations between physical variables present in engineering problems are often optimally described by differential equations.

However, if the dynamics of a physical system are unknown, we face the problem of identifying the process driving the system. This nonlinear modeling problem can be approached using a general model, whose parameters are estimated from observations of the system. These *system identification* and *parameter estimation* tasks can be attacked with neural networks, where the network corresponds to the model and the network weights are the free parameters. A learning algorithm searches for the optimal network weights fitting the network output to the actual system observations. Nevertheless, finding optimal weights for large neural networks is difficult (see section 2.3). An overview of the usage of neural networks for system identification and parameter estimation tasks typically found in control applications is given in [Nare90] and [Whit92].

Many engineering problems share some characteristics of both problem specifications: It is common that partial knowledge about the dynamics of a system is available as differential equations, but they are insufficient for modeling purposes. Assume for example, only the structure of the differential equations describing a system is known, but some physical constants are missing. The standard approach of applying neural networks to such problems uses only available system observations for modeling and the differential equations are ignored. However, an optimal usage of

information tries to integrate both the observation data and the structured knowledge into the neural networks. If we manage to combine both aspects of modeling in the networks, the accuracy, stability and efficiency of neural modeling can be improved, and the networks generalize better from observed data.

In this chapter, a method to exploit ordinary differential equations for the design of neural networks is presented. Both the architecture and the weights of the networks are determined from the differential equations. Single-step and multi-step numerical integration procedures for the approximation of the differential equations are derived using the resulting neural networks as basic building blocks. After initial design, the neural networks can be trained from observation data in order to account for the modeling deficiencies of the differential equations.

## 4.2 Local Approximation by Taylor Series

Assume we have a first order ordinary differential equation of the form (4-1) with initial condition (4-2).

$$x'(t) = f(t, x(t)) \tag{4-1}$$

$$x(t_0) = x_0 \tag{4-2}$$

If $x(t)$ and the functional $f$ are analytic in the neighborhood of the initial value $t_0$, we can successively differentiate the differential equation and we obtain the equations (4-3).

$$x^{(k)}(t_0) = f(t, x(t))^{(k-1)}\Big|_{t=t_0}, k = 1..K \tag{4-3}$$

Since the solution $x(t)$ is assumed to be analytic at $t_0$, it can be approximated near $t_0$ by a power series (4-4), whose derivatives are given in (4-5).

$$x(t) \approx \sum_{k=0}^{\infty} a_k \cdot (t - t_0)^k \tag{4-4}$$

$$x^{(k)}(t_0) = a_k \cdot k! \tag{4-5}$$

Solving (4-5) for the coefficients $a_k$ gives the solution (4-6), which turns (4-4) into the well known Taylor series (4-7) with convergence radius $\rho$.

$$a_k = \frac{x^{(k)}(t_0)}{k!} = \frac{1}{k!} \cdot f(t, x(t))^{(k-1)}\Big|_{t=t_0} \tag{4-6}$$

$$x(t) = \sum_{k=0}^{\infty} \frac{x^{(k)}(t_0)}{k!} \cdot (t - t_0)^k, t \in [t_0 - \rho, t_0 + \rho] \tag{4-7}$$

A local approximation (4-8) near $t_0$ is obtained by truncating the Taylor series after K terms.

$$x(t_0 + \Delta t) \approx \sum_{k=0}^{K} a_k \cdot \Delta t^k, \text{ where } a_k = \frac{x^{(k)}(t_0)}{k!} \tag{4-8}$$

This straightforward method to compute polynomial approximations (4-8) of differential equations is very old. Already Isaac Newton used infinite power series for the approximation of differential equations (see [Hair91], p. 4).

The result of the Taylor series method is a local approximation (4-8) of the differential equation (4-1), given as a polynom of the step size $\Delta t$. The coefficients $a_k$ are functions of the initial conditions (4-2), and they are determined by the solution of equation (4-5).

Provided that the step size $\Delta t$ is smaller than the convergence radius of the series and enough terms of the truncated series are used, the local approximation (4-8) of the solution of the differential equation allows to construct a single-step integrator, which is sufficiently accurate to compute good approximations over a large interval.

Taylor coefficients can be computed recursively and methods to choose a good step size do exist (see [Henr56], [Camp61], [Leav66], [Bart71], [Fair88], or for a short summary [Hair91], p. 47). Hence, many computer programs for the automatic approximation of differential equations based on long truncated power series have been implemented (see [Chan74], [Norm76], [Corl77], [Bart80], [Corl82]).

## 4.3 Generalization of the Taylor Series Method

In this section, we keep the basic idea of the Taylor series method, but we extend the power series approximation to more general approximation

functions. The power series (4-4) is replaced by a parametrized local approximation function (4-9), with finitely many parameters $a_k$.

$$x(t) \approx \Phi_K(t;t_0;a_0,a_1,\ldots,a_K), t \in [t_0 - \rho, t_0 + \rho] \tag{4-9}$$

The method of successive differentiation can be generalized to approximation function $\Phi_K$, if $\Phi_K$ satisfies the following conditions:

*Solvability*:
After substituting the approximation $\Phi_K$ for x(t), the equations generated by the successive differentiation method (4-3) must be solvable for the parameters $a_k$. This can be achieved by the following requirements:

*Triangulation condition*:

$$\Phi_K(t;t_0;a_0,a_1,\ldots,a_K)^{(k)}\Big|_{t=t_0} = \Theta_k^{\Phi}(a_0 \ldots a_k), \text{ where } k \le K \tag{4-10}$$

The k-th derivative $\Theta_k^{\Phi}$ of the approximation function $\Phi_K$ at initial value $t_0$ must depend only on the parameters $a_0$ ... $a_k$.

*Invertibility*:

$$\begin{aligned} y &= \Theta_k^{\Phi}(a_0 \ldots a_k) \\ a_k &= \Theta_k^{\Phi^{-1}}(y;a_0 \ldots a_{k-1}) \\ &= \Theta_k^{\Phi^{-1}}(\Theta_k^{\Phi}(a_0 \ldots a_k);a_0 \ldots a_{k-1}) \end{aligned} \tag{4-11}$$

The k-th derivative $\Theta_k^{\Phi}$ of the approximation function $\Phi_K$ at initial value $t_0$ must be invertible for the parameter $a_k$.

*Convergence*:

$$\lim_{K\to\infty} \left| x(t) - \Phi_K(t;t_0;a_0,\ldots,a_K) \right| = 0, t \in [t_0 - \rho, t_0 + \rho] \quad \text{for some norm } |\ldots| \tag{4-12}$$

With increasing number of parameters $a_k$, the sequence of truncated approximation functions $\Phi_K$ must converge to the real solution x(t) of the differential equation in a region near the initial value $t_0$.

If the approximation function $\Phi_K$ and the functional f are analytic at initial value $t_0$, and if $\Phi_K$ obeys the condition (4-10), the successive differentiation method (4-3) generates the following equations (4-13):

$$\begin{aligned}\Theta_0^{\Phi}(a_0) &= x_0 \\ \Theta_k^{\Phi}(a_0 \ldots a_k) &= f(t, \Phi_K(t;t_0;a_0,\ldots,a_K))^{(k-1)}\Big|_{t=t_0} \\ &= g_{k-1}(t_0;\Theta_0^{\Phi}(a_0),\ldots,\Theta_{k-1}^{\Phi}(a_0 \ldots a_{k-1})), k = 1 \ldots K\end{aligned} \tag{4-13}$$

The solution of (4-13) for parameter $a_k$ is given in equation (4-14):

$$a_k = \Theta_k^{\Phi^{-1}}(g_{k-1}(t_0;\Theta_0^{\Phi}(a_0),\ldots,\Theta_{k-1}^{\Phi}(a_0 \ldots a_{k-1}));a_0 \ldots a_{k-1}) \tag{4-14}$$

After having solved the first k equations of (4-13) for the parameters $a_0..a_k$, the k+1-th equation contains only the unknown parameter $a_k$. Consequently, equations (4-13) can be solved iteratively, generating one parameter $a_k$ after the other.

If both the derivatives $g_k$ of f and $\Theta_k^{\Phi}$ are linear, (4-13) turns into a linear equation system, whose matrix is triangular[3] because of condition (4-10). For complex functions $\Theta_k^{\Phi}$, more sophisticated methods may be needed (e.g., see [Wu84]).

The following theorem demonstrates that the method of successive differentiation suffices to guarantee the convergence of the sequence of approximations $\Phi_K$ to x(t):

*Theorem* (4.1): Let x(t) be an analytic function defined by the differential equation (4-15), and let $\Phi_K$ be an analytic local approximation function of the form (4-16) with K parameters $a_k$.

$$\begin{aligned}x'(t) &= f(t, x(t)) \\ x(t_0) &= x_0\end{aligned} \tag{4-15}$$

$$x(t) \approx \Phi_K(t;t_0;a_0,\ldots,a_K), t \in [t_0 - \rho, t_0 + \rho] \tag{4-16}$$

$$\begin{aligned}\Phi_K(t_0;t_0;a_0,\ldots,a_K) &= x_0 \\ \Phi_K(t;t_0;a_0,\ldots,a_K)^{(k)}\Big|_{t=t_0} &= f(t, \Phi_K(t;t_0;a_0,\ldots,a_K))^{(k-1)}\Big|_{t=t_0}, k = 1..K\end{aligned} \tag{4-17}$$

---

[3] Since condition (4-10) generates triangular matrices for linear systems, it is called the *triangulation condition*.

If the parameters $a_k$ satisfy equations (4-17), then for any t within a radius $\rho$ of $t_0$, the approximation $\Phi_K$ converges to x(t) with increasing K, where $\rho$ equals the radius of the Taylor series expansion of x(t):

$$\lim_{K\to\infty}\left|x(t) - \Phi_K(t;t_0;a_0,\ldots,a_K)\right| = 0, t \in [t_0 - \rho, t_0 + \rho] \tag{4-18}$$

Proof:

The functions $\Phi_K$ and x(t) can both be developed into the Taylor series (4-19) and (4-20) with convergence radius $\eta$ and $\rho$.

$$\begin{aligned}\Phi_K(t;t_0;a_0,\ldots,a_K) &= \sum_{k=0}^{\infty}\frac{\Phi_K{}^{(k)}(t_0;t_0;a_0,\ldots,a_K)}{k!}\cdot(t-t_0)^k \\ &= \sum_{k=0}^{\infty} b_k \cdot \Delta t^k, t \in [t_0 - \eta, t_0 + \eta]\end{aligned} \tag{4-19}$$

$$\begin{aligned}x(t) &= \sum_{k=0}^{\infty}\frac{x^{(k)}(t_0)}{k!}\cdot(t-t_0)^k \\ &= \sum_{k=0}^{\infty} c_k \cdot \Delta t^k, t \in [t_0 - \rho, t_0 + \rho]\end{aligned} \tag{4-20}$$

Since the parameters $a_k$ are solutions of equations (4-17), the derivatives $\Phi_K{}^{(k)}(t_0;t_0;a_0,\ldots,a_K)$ coincide with the derivatives $x^{(k)}(t_0)$. This can be shown with a proof by induction: The case (4-21) for k=0 is trivial, whereas the general case is covered by the induction step (4-22).

$$\Phi_K(t_0;t_0;a_0,\ldots,a_K) = \Theta_0 = x(t_0) = x_0 \tag{4-21}$$

$$\begin{aligned}\Phi_K(t;t_0;a_0,\ldots,a_K)^{(k)}\big|_{t=t_0} &= \Theta_k \\ &= f(t,\Phi_K(t;t_0;a_0,\ldots,a_K))^{(k-1)}\big|_{t=t_0} \\ &= g_{k-1}(t_0;\Theta_0,\ldots,\Theta_{k-1}) \\ &= g_{k-1}(t_0;x(t_0),\ldots,x^{(k-1)}(t_0)) \\ &= f(t,x(t))^{(k-1)}\big|_{t=t_0} \\ &= x^{(k)}(t_0)\end{aligned} \tag{4-22}$$

Since we know that the derivatives $\Phi_K^{(k)}(t_0;t_0;a_0,...,a_K)$ equal $x^{(k)}(t_0)$, the coefficients $b_k$ and $c_k$ of the Taylor series' are equal as well for k=0..K. Consequently, for K going to infinity, the Taylor series of $\Phi_K$ converges to the Taylor series of x(t) and $\eta$ converges to $\rho$. Since two power series with equal coefficients converge to the same function (see the identity theorem for power series in [Burg92], p. 414), $\Phi_K$ is proven to approach x(t) with increasing K.

❑

The result of theorem (4.1) is independent of the specific form of approximation function $\Phi_K$ that is used. Consequently, if we select a specific function $\Phi_K$, only conditions (4-10), (4-11) and (4-17) have to be checked and the convergence (4-12) of $\Phi_K$ to x(t) is implied.

## 4.4 Choice of Suitable Neural Network Approximators

In section 4.3, the Taylor series method for generating polynomial approximations has been generalized to nonpolynomial functions $\Phi_K$. This method allows to construct neural networks for the approximation of differential equations, if neural network architectures can be found for $\Phi_K$ that satisfy the conditions (4-10) and (4-11).

Although many standard neural network models have been proven to be universal function approximators (e.g., see [Cybe89], [Funa89]], [Stin89, [Giro90], [Hart90], [Horn90], [Park91], [Ito92], [Cott93], [Park93]), most of them do not satisfy the property (4-10), which requires that the network parameters $a_{i>k}$ vanish when the k-th derivative is taken at the initial value $t_0$. This strong requirement is necessary for solving equations (4-13) iteratively, but it is satisfied trivially only by polynoms.

This attractive property of power series suggests that we should look for neural network architectures which can be rewritten as variants of power series. The derivation of power series of some basis function can take advantage of the fact that the concatenation of two functions turns into multiplication of their derivatives. Thus, if the derivatives of the underlying power series vanish, also the derivatives of the modified series vanish.

There exist classes of neural networks that satisfy additive and multiplicative closure, which means that power series of network functions

can again be implemented by a neural network of the same class. The classes of neural networks satisfying the Stone-Weierstrass theorem share this property:

*Theorem* (4.2) (*Stone-Weierstrass*, see [Cott90]): Let domain D be a compact space of N dimensions, and let F be a set of continuous real-valued functions on D, satisfying the following criteria:

1) *Identity function*: The constant function f(x)=1 is in F.

2) *Separability*: For any two points $x_1 \neq x_2$ in D, there is an f in F such that $f(x_1) \neq f(x_2)$.

3) *Algebraic closure*: If f and g are any two functions in F, f·g and a·f+b·g are in F for any two real numbers a and b.

Then F is dense in C(D), the set of continuous real-valued functions on D. In other words, for any $\varepsilon>0$ and any function in C(D), there is a function f in F such that $|g(x) - f(x)| < \varepsilon$ for all $x \in D$.

Consequently, classes of neural networks satisfying the Stone-Weierstrass theorem are guaranteed to be universal function approximators for continuous real-valued functions. In the following sections 4.4.1 and 4.4.2, two such neural network architectures are investigated: the *Modified Logistic Network* and the *Radial Basis Function Network* with Gaussian units.

### 4.4.1 Modified Logistic Networks

The application of neural networks to systems of differential equations requires only networks with a single output unit, because every variable of a differential equation is approximated by a different network. A traditional three-layered neural network model with sigmoidal units in the hidden layer and a linear output unit is represented by function (4-23).

$$y(\bar{x}; \bar{z}_1 .. \bar{z}_K, \theta_1 .. \theta_K, w_1 .. w_K) = \sum_{k=1}^{K} w_k \cdot \text{sigmoid}(\bar{z}_k \cdot \bar{x} - \theta_k), \quad \text{where e.g., } \text{sigmoid}(x) = \tanh(x) \tag{4-23}$$

Unfortunately, networks (4-23) using sigmoidal functions like *tanh* do not satisfy algebraic closure. Consequently, they are not able to implement polynoms of their activation functions exactly.

In [Cott90], Cotter proposes a variant of this network, the *Modified Logistic Network* (4-24), which satisfies the Stone-Weierstrass theorem and which compensates this defect.

$$y(\bar{x};\bar{z}_{1,1}..\bar{z}_{I,K},w_1..w_K)=\sum_{k=1}^{K} w_k \cdot \frac{1}{1+\sum_{i=1}^{I} e^{-\bar{z}_{i,k}\cdot\bar{x}}} \tag{4-24}$$

Note that the traditional networks (4-23) using the function *tanh* are a subset of the class of Modified Logistic Networks, because *tanh* satisfies the identity (4-25). Polynoms of *tanh* functions, however, do not belong to the class of *tanh* networks any more.

$$\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\frac{e^{2x}-1}{e^{2x}+1}=1-\frac{2}{1+e^{2x}} \tag{4-25}$$

We now propose the neural network (4-26) for approximation function $\Phi_K$. $\Phi_K^{MLG}$ is a polynom of networks (4-23),.which can be represented as a Modified Logistic Network (4-24) after transforming *tanh* according to identity (4-25) and after expanding all multiplications.

$$\Phi_K^{MLG}(t;t_0;a_0,...,a_K)=$$
$$a_0+\sum_{k=1}^{K}\left[\tanh(t-t_0)^{k-1}\cdot\sum_{i=0}^{\left\lfloor\frac{K}{2}\right\rfloor}\frac{\tanh(a_k\cdot(t-t_0))^{2\cdot i+1}}{2\cdot i+1}\right] \tag{4-26}$$

The derivatives of the approximation function $\Phi_K^{MLG}$ at the initial value $t_0$ are given in (4-27). Its coefficients $c_{k,i}$ are shown in table 4.1, where empty entries denote 0.

$$\Phi_K^{MLG}(t;t_0;a_0,...,a_K)^{(k\le K)}\Big|_{t=t_0}=\Theta_k^{MLG}(a_0..a_k)=\sum_{i=0}^{k} c_{k,i}\cdot a_i \tag{4-27}$$

| **k** | $c_{k,0}$ | $c_{k,1}$ | $c_{k,2}$ | $c_{k,3}$ | $c_{k,4}$ | $c_{k,5}$ | **...** |
|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | |
| 1 | | 1 | | | | | |
| 2 | | | 2 | | | | |
| 3 | | | | 6 | | | |
| 4 | | | -8 | | 24 | | |
| 5 | | | | -80 | | 120 | |
| ... | ... | ... | ... | ... | ... | ... | ... |

Table 4.1 : Coefficients $c_{k,i}$

*Theorem* (4.3): A Modified Logistic Network of the form (4-26) allows the application of the method of successive differentiation (4-13) for the local approximation of differential equations. The equations generated by (4-13) can be recursively solved for parameters $a_k$ with equation (4-14).

Proof:

1) *Solvability*: The derivatives (4-27) of the approximation function $\Phi_K^{MLG}$ are linear in the parameters $a_k$ and satisfy both the triangulation condition (4-10) and the invertibility condition (4-11). Therefore, equations (4-13) are solvable for the parameters $a_k$ using (4-14).

2) *Convergence*: According to theorem (4.1), convergence of $\Phi_K^{MLG}$ to the solution x(t) is a consequence of the application of the method of successive differentiation for computing the parameters $a_k$. The convergence radius of $\Phi_K^{MLG}$ is identical to the radius $\rho$ of a Taylor series expansion of the solution x(t) of the differential equation.

❑

Since the derivatives of the approximation function $\Phi_K^{MLG}$ are linear expressions of the parameters $a_k$, equation system (4-13) to be solved is not significantly more complex than for polynomial approximation functions. This makes the Modified Logistic Network $\Phi_K^{MLG}$ a good choice for the local approximation of differential equations.

### 4.4.2 Radial Basis Function Networks

Radial basis functions are well known in approximation theory. Some of their properties relevant to multidimensional approximation are documented in [Powe87] and [Pogg90]. The following summary is taken from [Pogg90], pp. 1494-1495:

*Interpolation problem: Given N different points $\bar{x}_i \in R^n$ and N real numbers $y_i$, find a function F from $R^n$ to R satisfying the interpolation conditions:*

$$F(\bar{x}_i) = y_i, i = 1..N \tag{4-28}$$

*The Radial Basis Function approach consists in choosing a function F of the following form:*

$$F(\bar{x}) = \sum_{i=1}^{N} c_i \cdot h(\left\|\bar{x} - \bar{x}_i\right\|_2) + \sum_{i=1}^{m} d_i \cdot p_i(\bar{x}), m \le N \tag{4-29}$$

*where h is a continuous function from $R^+$ to R, usually called the radial basis function. $\|...\|_2$ is the Euclidean norm on $R^n$, $p_i$ is a basis of the linear space of algebraic polynomials of degree at most k-1, and k is given. The interpolation conditions give N linear equations for the coefficients of (4-29), so that the remaining degrees of freedom are fixed by the following constraints:*

$$\sum_{i=1}^{N} c_i \cdot p_j(\bar{x}_i) = 0, j = 1..m \tag{4-30}$$

*In case k=0, the radial basis function expansion (4-29) is simplified to (4-31) and the equations (4-28) turn into the linear system (4-32):*

$$F(\bar{x}) = \sum_{i=1}^{N} c_i \cdot h(\left\|\bar{x} - \bar{x}_i\right\|_2) \tag{4-31}$$

$$H \cdot \bar{c} = \bar{y} \tag{4-32}$$

*A sufficient condition for the existence of a solution of the system (4-32) is given by the following theorem:*

*Theorem (4.4) (Micchelli, see [Pogg90]): Let h be a continuous function on [0,∞) and positive on (0,∞). Suppose its first derivative is completely*

*monotonic but not constant on (0,∞). Then, for any distinct vectors* $\bar{x}_1 \ldots \bar{x}_N \in R^n$*:*

$$(-1)^{N-1} \cdot \det\ h(\left\|\bar{x}_i - \bar{x}_j\right\|_2^2) > 0 \qquad (4\text{-}33)$$

*The essence of this theorem is that if the first derivative of a function is completely monotonic, this function can be used as Radial Basis Function, since the matrix H associated to it can be inverted. A list of functions that can be used in practice for data interpolation by means of the Radial Basis Function expansion (4-31) is given in table 4.2, and their use is justified by the results of Micchelli:*

$$h(r) = e^{-\left(\frac{r}{c}\right)^2}$$
$$h(r) = \frac{1}{(c^2 + r^2)^\alpha}, \alpha > 0$$
$$h(r) = (c^2 + r^2)^\beta, 0 < \beta < 1$$
$$h(r) = r$$

*Table 4.2 : Different radial basis functions*

For our purposes, we use three-layered neural networks with one input layer, one hidden layer with Gaussian radial basis function units and one output layer with a single linear unit. Since every variable of the system of differential equations is approximated by a different neural network, we need only one output unit, which finally computes the function (4-34).

$$y(\bar{x}; \bar{z}_1 .. \bar{z}_K, \sigma_1 .. \sigma_K, w_1 .. w_K) = \sum_{k=1}^{K} w_k \cdot e^{-\left(\frac{\bar{x} - \bar{z}_k}{\sigma_k}\right)^2}, \text{where } \sigma_k > 0 \qquad (4\text{-}34)$$

*Theorem* (4.5): Let the class RBF of functions of the form (4-34) be called *Radial Basis Function Networks*. Then, all functions f∈RBF satisfy the Stone-Weierstrass theorem (4.2).

Proof:

1) *Identity function*: The constant function f(x)=1 is in RBF if we allow $\sigma \to \infty$:

$$f(x) = \lim_{\sigma \to \infty} e^{-\left(\frac{x}{\sigma}\right)^2} = 1 \tag{4-35}$$

2) *Separability*: For any two points $x_1 \neq x_2$ in D, there is an f in RBF such that $f(x_1) \neq f(x_2)$ :

$$f(x) = e^{-(x-x_1)^2} \tag{4-36}$$

3) *Algebraic closure*: If f and g are any two functions in RBF, f·g and a·f+b·g are in RBF for any two real numbers a and b:

Since the output layer is linear, the sum of two networks can be trivially implemented by one network. The product of two networks is transformed into a sum of products of the hidden layer activation functions, which again can be represented by Gaussian units. This transformation is justified by the identity (4-37).

$$w_1 \cdot e^{-\left(\frac{\bar{x}-\bar{z}_1}{\sigma_1}\right)^2} \cdot w_2 \cdot e^{-\left(\frac{\bar{x}-\bar{z}_2}{\sigma_2}\right)^2} = w_3 \cdot e^{-\left(\frac{\bar{x}-\bar{z}_3}{\sigma_3}\right)^2}$$
$$\bar{z}_3 = \frac{\sigma_1^2 \cdot \bar{z}_2 + \sigma_2^2 \cdot \bar{z}_1}{\sigma_1^2 + \sigma_2^2}, \sigma_3^2 = \frac{\sigma_1^2 \cdot \sigma_2^2}{\sigma_1^2 + \sigma_2^2}, w_3 = w_1 \cdot w_2 \cdot e^{\frac{-(\bar{z}_1 - \bar{z}_2)^2}{\sigma_1^2 + \sigma_2^2}} \tag{4-37}$$

The generalization of (4-37) to multiple products is given in (4-38).

$$\prod_{k=1}^{N} w_k \cdot e^{-\left(\frac{\bar{x}-\bar{z}_k}{\sigma_k}\right)^2} = w \cdot e^{-\left(\frac{\bar{x}-\bar{z}}{\sigma}\right)^2}$$

$$\bar{z} = \frac{\sum_{k=1}^{N} \bar{z}_k \cdot \prod_{\substack{i=1 \\ i \neq k}}^{N} \sigma_i^2}{\tilde{\sigma}}, \sigma^2 = \frac{\prod_{i=1}^{N} \sigma_i^2}{\tilde{\sigma}}, \tilde{\sigma} = \sum_{k=1}^{N} \prod_{\substack{i=1 \\ i \neq k}}^{N} \sigma_i^2$$

$$w = \exp\left(\frac{-\sum_{k=1}^{N} \bar{z}_k \cdot \left( \bar{z}_k \cdot \sum_{\substack{i=1 \\ i \neq k}}^{N} \prod_{\substack{j=1 \\ j \neq k \\ j \neq i}}^{N} \sigma_j^2 - \sum_{\substack{i=1 \\ i \neq k}}^{N} \bar{z}_i \cdot \prod_{\substack{j=1 \\ j \neq k \\ j \neq i}}^{N} \sigma_j^2 \right)}{\tilde{\sigma}}\right) \cdot \prod_{k=1}^{N} w_k \qquad (4\text{-}38)$$

Therefore, Radial Basis Function Networks (4-34) satisfy algebraic closure.

❑

Note that even the division of two Gaussian functions can be implemented by a single Gaussian unit, which is shown in (4-39).

$$\frac{w_1 \cdot e^{-\left(\frac{\bar{x}-\bar{z}_1}{\sigma_1}\right)^2}}{w_2 \cdot e^{-\left(\frac{\bar{x}-\bar{z}_2}{\sigma_2}\right)^2}} = w_3 \cdot e^{-\left(\frac{\bar{x}-\bar{z}_3}{\sigma_3}\right)^2}, \text{where } \sigma_1^2 \neq \sigma_2^2$$

$$\bar{z}_3 = \frac{\sigma_2^2 \cdot \bar{z}_1 - \sigma_1^2 \cdot \bar{z}_2}{\sigma_2^2 - \sigma_1^2}, \sigma_3^2 = \frac{\sigma_1^2 \cdot \sigma_2^2}{\sigma_2^2 - \sigma_1^2}, w_3 = \frac{w_1}{w_2} \cdot e^{\frac{-(\bar{z}_1 - \bar{z}_2)^2}{\sigma_2^2 - \sigma_1^2}} \qquad (4\text{-}39)$$

We now present the neural network (4-40) for approximation function $\Phi_K$, which can be converted into a Radial Basis Function Network of the form (4-34) for constant $\Delta t = t - t_0$.

$$\Phi_K^{RBF}(t; t_0; a_0, \ldots, a_K) =$$

$$a_0 + \sum_{k=1}^{K} \left[ \left( \frac{1}{e} - e^{-((t-t_0)-1)^2} \right)^{k-1} \cdot \sum_{i=1}^{K} \frac{b_i \cdot e^{i-1}}{2^i \cdot i!} \cdot \left( \frac{1}{e} - e^{-(a_k \cdot (t-t_0)-1)^2} \right)^i \right] \qquad (4\text{-}40)$$

The transformation (4-41) translates the arguments of the Gaussian units in (4-40) into their equivalent in (4-34), where the parameters $a_k$ correspond to the inputs into the Gaussian layer.

$$x_k = a_k, \sigma_k = \frac{1}{t - t_0} = \frac{1}{\Delta t}, z_k = \sigma_k \tag{4-41}$$

After expanding all multiplications in (4-40), the approximation function $\Phi_K^{RBF}$ can be rewritten as a polynom of Gaussian functions. Identity (4-38) for multiple products transforms $\Phi_K^{RBF}$ into a Radial Basis Function Network (4-34). The coefficients $b_i$ of the neural network (4-40) are given in table 4.3. They are determined in order to satisfy condition (4-10).

Although we have not found an explicit mathematical formula for the coefficients $b_i$, they can be computed easily by the recursive equation (4-43), which is derived from the simplified approximation function (4-42).

$$\tilde{\Phi}_K^{RBF}(t;t_0;a_0,a_1) = a_0 + \sum_{i=1}^{K} \frac{b_i \cdot e^{i-1}}{2^i \cdot i!} \cdot \left( \frac{1}{e} - e^{-(a_1 \cdot (t-t_0)-1)^2} \right)^i \tag{4-42}$$

$$b_1 = 1, b_k = \frac{e \cdot (-1)^{k-1}}{a_1^{\,k}} \cdot \tilde{\Phi}_{k-1}^{RBF}(t;t_0;a_0,a_1)^{(k)}\Big|_{t=t_0} \tag{4-43}$$

| **Coefficients $b_i$** |
|---|
| $b_1 = 1$ |
| $b_2 = 1$ |
| $b_3 = 5$ |
| $b_4 = 25$ |
| $b_5 = 209$ |
| $b_6 = 1961$ |
| $b_7 = 23589$ |
| ... |

Table 4.3 : Coefficients $b_i$

The derivatives of the approximation function $\Phi_K^{RBF}$ at the initial value $t_0$ are given in (4-44). Their coefficients $c_{k,i}$ are shown in table 4.4, where empty entries denote 0.

$$\Phi_K^{RBF}(t;t_0;a_0,a_1..a_K)^{(k\le K)}\Big|_{t=t_0} = \Theta_k^{RBF}(a_0..a_k) = \sum_{i=0}^{k} \frac{c_{k,i}}{e^i} \cdot a_i \tag{4-44}$$

| **k** | $c_{k,0}$ | $c_{k,1}$ | $c_{k,2}$ | $c_{k,3}$ | $c_{k,4}$ | $c_{k,5}$ | **...** |
|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | |
| 1 | | -1 | | | | | |
| 2 | | | 4 | | | | |
| 3 | | | 6 | -24 | | | |
| 4 | | | -16 | -96 | 192 | | |
| 5 | | | -100 | 200 | 1440 | -1920 | |
| ... | ... | ... | ... | ... | ... | ... | ... |

Table 4.4 : Coefficients $c_{k,i}$

*Theorem* (4.6): A Radial Basis Function Network of the form (4-40) allows the application of the method of successive differentiation (4-13) for the local approximation of differential equations. The equations generated by (4-13) can be recursively solved for parameters $a_k$ with equation (4-14).

Proof:

1) *Solvability*: The derivatives (4-44) of the approximation function $\Phi_K^{RBF}$ are linear in the parameters $a_k$ and satisfy both the triangulation condition (4-10) and the invertibility condition (4-11). Therefore, equations (4-13) are solvable for the parameters $a_k$ using (4-14).

2) *Convergence*: According to theorem (4.1), convergence of $\Phi_K^{RBF}$ to the solution x(t) is a consequence of the application of the method of successive differentiation for computing the parameters $a_k$. The convergence radius of $\Phi_K^{RBF}$ is identical to the radius ρ of a Taylor series expansion of the solution x(t) of the differential equation.

❑

According to theorem (4.6), the approximation function (4-40) describes a Radial Basis Function Network which allows the application of the method of successive differentiation (4-13) for the local approximation of differential equations. Since the derivatives of the approximation function $\Phi_K^{RBF}$ are

linear expressions of the parameters $a_k$, the equation system (4-13) to be solved is only slightly more complex than for polynomial approximation functions. This makes the Radial Basis Function Network $\Phi_K^{RBF}$ a good choice as an approximation function.

## 4.5 Transformation of Differential Equations into Approximable Form

In sections 4.4.1 and 4.4.2, the Modified Logistic Network (4-26) and the Radial Basis Function Network (4-40) have been proposed for approximation function $\Phi_K$. They both satisfy the convergence and solvability conditions described in section 4.3. Fortunately, their derivatives $\Theta_k$ are linear expressions of the unknown parameters $a_k$, which generates particularly simple solutions to equations (4-13). Consequently, the parameters $a_k$ depend only on the functional f, respectively on its derivatives $g_k$.

The architecture of the resulting neural networks looks as follows: The functional dependency of the parameters $a_k$ on the initial conditions (4-2) of the differential equation determines the structure of a preprocessing layer. It is followed by a middle and an output layer, whose network weights are independent of the differential equation (see figure 4.1).

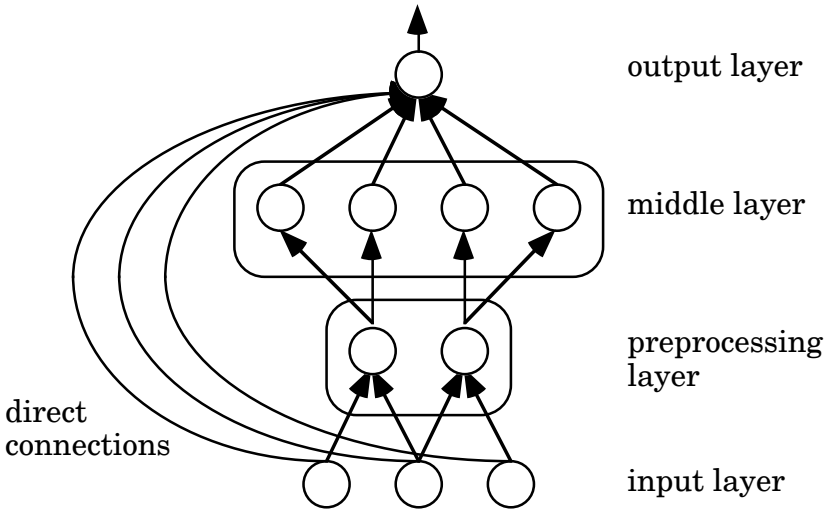



Fig. 4.1 : Network architecture

In case the functional f is linear and independent of time t, equations (4-13) are also linear and can be solved by backsubstitution. In this case, the parameters $a_k$ depend linearly on the initial conditions of the differential

equation and both types of neural networks do not need an intermediate preprocessing layer.

If the functional f is nonlinear, however, a nonlinear preprocessing layer between the input and the middle layer is needed, which is not regarded as standard neural network architecture any more.

Since we want to keep standard architectures if possible, we try to solve this problem by a suitable transformation of the differential equation, turning nonlinear functionals f into polynoms. In [Kern81] and [Fair88], a set of transformation rules is offered, which cover a very general class of functionals f. Actually, Kerner demonstrates in [Kern81] that polynomial systems can be even further simplified to quadratic systems, which are called *Riccati-systems*. Consequently, the quadratic representation can be regarded as a *normal form* for differential equations, and a very general class of differential equations can be reduced to this form. The neural networks that result from such quadratic or polynomial functionals f use polynomial preprocessing layers. In the literature, these networks are known as *higher-order neural networks* (see [Sejn86], [Lee86] or [Pine87]).

The transformation method is simple: Complex expressions are replaced by new variables, which are then defined by additional differential equations and their corresponding initial conditions. Since most "interesting" functions are themselves generated by rational differential equations, this transformation allows to reduce these functions to their associated rational derivatives. After repeated application of the transformation rules, polynomial differential equations are obtained. This method covers rational, irrational and most transcedental functions usually encountered in physical modeling. Basically, all functions controlled by differential equations of finite order can be treated this way (a counterexample is the gamma function, which does not obey a differential equation of finite order).

In table 4.5, some transformation rules are listed which serve to eliminate nonlinear functions.

| **Replaced Functions** | **Added Differential Equations** | **Added Initial Conditions** |
|---|---|---|
| $c \cdot t \rightarrow a(t)$ | $a'(t) = c$ | $a(t_0) = c \cdot t_0$ |
| $\frac{1}{x(t)} \rightarrow a(t)$ | $a'(t) = -a(t)^2 \cdot x'(t)$ | $a(t_0) = \frac{1}{x(t_0)}$ |
| $\sin(x(t)) \rightarrow a(t)$<br>$\cos(x(t)) \rightarrow b(t)$ | $a'(t) = b(t) \cdot x'(t)$<br>$b'(t) = -a(t) \cdot x'(t)$ | $a(t_0) = \sin(x(t_0))$<br>$b(t_0) = \cos(x(t_0))$ |
| $\ln(x(t)) \rightarrow a(t)$ | $a'(t) = \frac{x'(t)}{x(t)}$ | $a(t_0) = \ln(x(t_0))$ |
| $e^{x(t)} \rightarrow a(t)$ | $a'(t) = a(t) \cdot x'(t)$ | $a(t_0) = e^{x(t_0)}$ |
| $\tanh(x(t)) \rightarrow a(t)$ | $a'(t) = (1 - a(t)^2) \cdot x'(t)$ | $a(t_0) = \tanh(x(t_0))$ |

Table 4.5 : Example of transformation rules

Note that the first transformation rule in table 4.5 formally erases the distinction between autonomous and nonautonomous differential equations by the near-trivial introduction of a variable.

*Theorem* (4.7): Let differential equations of polynomial form be called being of *approximable form*. Then, if a differential equation of the form (4-1) can be transformed into a system of differential equations of approximable form by repeated transformation of the functional f, the differential equations can be locally approximated within the convergence radius to any desired accuracy by higher-order neural networks of type (4-26) or (4-40) with polynomial preprocessing layers.

Proof:
If the functional f can be transformed into polynomial form by introducing new variables, each variable can be approximated by a neural network. The parameters of the networks (4-26) or (4-40) are the solutions of equations (4-14), which result in polynomial expressions of the initial conditions of the differential equations. Therefore, the preprocessing layer of the network consists in polynoms determined by the differential equations, whereas the weights of the middle and the output layer are independent of the differential equations and remain fixed. The accuracy of the approximation is controlled by the number of parameters $a_k$ that are included, which

also determines the size of the neural networks. According to theorems (4.3) and (4.6), the differential equations can be approximated to any desired accuracy within the convergence radius ρ. ❑

The following example illustrates the transformation process (individual transformation steps are marked by arrows that are labeled with the transformation rule applied) :

1. Using the transformation rules of table 4.5, the rational differential equation (4-45) with initial condition (4-46) is first transformed into equations (4-47) and (4-48), which introduces the new variable a(t). Differential equations (4-47) are now in polynomial form.

$$x'(t) = \frac{1}{x(t)^2} \tag{4-45}$$

$$x(t_0) = x_0 \tag{4-46}$$

$$\downarrow \quad \frac{1}{x(t)^2} \rightarrow a(t)$$

$$\begin{aligned} x'(t) &= a(t) \\ a'(t) &= -a(t)^2 \cdot 2 \cdot x(t) \cdot x'(t) \\ &= -2 \cdot a(t)^3 \cdot x(t) \end{aligned} \tag{4-47}$$

$$\begin{aligned} x(t_0) &= x_0 \\ a(t_0) &= \frac{1}{x_0^2} \end{aligned} \tag{4-48}$$

2. Then, equations (4-47) are transformed successively into quadratic form, resulting in equations (4-55) and (4-56), which introduce the new variables b(t) to f(t).

$$\begin{aligned} x'(t) &= a(t) \\ a'(t) &= -2 \cdot a(t)^3 \cdot x(t) \end{aligned} \tag{4-49}$$

$$\downarrow \quad a(t)^3 \rightarrow b(t)$$

$$\begin{aligned} a'(t) &= -2 \cdot b(t) \cdot x(t) \\ b'(t) &= -6 \cdot a(t)^2 \cdot b(t) \cdot x(t) \end{aligned} \tag{4-50}$$

$$\downarrow \quad a(t)^2 \rightarrow c(t)$$

$$\begin{aligned} b'(t) &= -6 \cdot b(t) \cdot c(t) \cdot x(t) \\ c'(t) &= -4 \cdot a(t) \cdot b(t) \cdot x(t) \end{aligned} \tag{4-51}$$

$$\downarrow \quad x(t) \cdot b(t) \rightarrow d(t)$$

$$\begin{aligned} b'(t) &= -6 \cdot c(t) \cdot d(t) \\ c'(t) &= -4 \cdot a(t) \cdot d(t) \\ d'(t) &= a(t) \cdot b(t) - 6 \cdot c(t) \cdot d(t) \cdot x(t) \end{aligned} \tag{4-52}$$

$$\downarrow \quad x(t) \cdot c(t) \rightarrow e(t)$$

$$\begin{aligned} d'(t) &= a(t) \cdot b(t) - 6 \cdot d(t) \cdot e(t) \\ e'(t) &= a(t) \cdot c(t) - 4 \cdot a(t) \cdot d(t) \cdot x(t) \end{aligned} \tag{4-53}$$

$$\downarrow \quad x(t) \cdot a(t) \rightarrow f(t)$$

$$\begin{aligned} e'(t) &= a(t) \cdot c(t) - 4 \cdot d(t) \cdot f(t) \\ f'(t) &= c(t) - 2 \cdot d(t) \cdot x(t) \end{aligned} \tag{4-54}$$

$$\downarrow$$

$$\begin{aligned} x'(t) &= a(t) \\ a'(t) &= -2 \cdot d(t) \\ b'(t) &= -6 \cdot c(t) \cdot d(t) \\ c'(t) &= -4 \cdot a(t) \cdot d(t) \\ d'(t) &= a(t) \cdot b(t) - 6 \cdot d(t) \cdot e(t) \\ e'(t) &= a(t) \cdot c(t) - 4 \cdot d(t) \cdot f(t) \\ f'(t) &= c(t) - 2 \cdot d(t) \cdot x(t) \end{aligned} \tag{4-55}$$

$$
\begin{aligned}
&x(t_0) = x_0 \\
&a(t_0) = \frac{1}{x_0^{\,2}}, b(t_0) = \frac{1}{x_0^{\,6}} \\
&c(t_0) = \frac{1}{x_0^{\,4}}, d(t_0) = \frac{1}{x_0^{\,5}} \\
&e(t_0) = \frac{1}{x_0^{\,3}}, f(t_0) = \frac{1}{x_0}
\end{aligned}
\tag{4-56}
$$

The substitution of compound expressions by new variables has a serious drawback, however: Since for each new variable another differential equation is introduced, the size of the system of differential equations may be substantially increased. The worst case occurs if full transformation to Riccati-form is desired, which our example (4-55) demonstrates (see [Kern81] for an estimation of the increase in size). We also encounter a difference in numerical precision between the approximations of the original and the transformed differential equations: After transforming the differential equation by differentiating away the nonlinear functions, we actually reverse this transformation by numerical integration. In most cases this gives worse results. Thus, approximating the transformed differential equations in general requires smaller integration steps than approximating the original equations.

On the other hand, if we transform a system of differential equations into quadratic form, and if we apply the method of successive differentiation and solve the generated equations, we still get polynomial solutions, not quadratic ones. This is already the case for very simple differential equations. Equations (4-58) generated by successive differentiation of example (4-57) demonstrate this phenomenon.

$$
\begin{aligned}
&x'(t) = x(t)^2 \\
&x(t_0) = x_0
\end{aligned}
\tag{4-57}
$$

$$
\begin{aligned}
&x'(t_0) = x(t_0)^2 \\
&x''(t_0) = \left(x(t)^2\right)'\Big|_{t=t_0} = 2 \cdot x(t) \cdot x'(t)\big|_{t=t_0} = 2 \cdot x(t_0)^3 \\
&x'''(t_0) = \left(2 \cdot x(t)^3\right)'\Big|_{t=t_0} = 6 \cdot x(t)^2 \cdot x'(t)\big|_{t=t_0} = 6 \cdot x(t_0)^4 \\
&\ldots
\end{aligned}
\tag{4-58}
$$

Since quadratic differential equations still generate preprocessing layers with polynoms of unrestricted degree, the transformation from polynomial to quadratic form is not worth the effort. It only increases the complexity of the created neural networks. We therefore recommend to transform the differential equations only into polynomial form, not into full normal form.

## 4.6 Single-Step and Multi-Step Integration Procedures

So far, we have discussed the construction of neural networks for the local approximation of differential equations in the neighborhood of $t_0$. Using the approximation function $\Phi_K$, we can construct a single-step numerical integration method (4-59).

$$x(t_0 + \Delta t) \approx \Phi_K(t_0 + \Delta t; t_0; a_0, \ldots, a_K), \Delta t < \rho \tag{4-59}$$

If the function $\Phi_K(t_0+\Delta t;t_0;a_0,\ldots,a_K)$ is independent[4] of $t_0$ and if the parameters $a_k$ are resolved using the method of successive differentiation from section 4.3, the single-step method corresponds to an iterated map (4-60).

$$\begin{aligned} x(t_0) &= x_0 \\ x_{k+1} &= \varphi_\Phi(\Delta t; x_k) \end{aligned} \tag{4-60}$$

If the step size $\Delta t$ is chosen smaller than the convergence radius $\rho$, the successive values $x_k$ at the points $t_k = t_0 + k \cdot \Delta t$ provide good approximations of the solution of the differential equation. This is a standard method used in numerical algorithms, where truncated Taylor series approximations $\Phi_K$ are reduced to linear functions $\varphi_\Phi$ (e.g., see [Hair91]).

Up to now, we exclusively treated first order differential equations. In order to handle higher-order differential equations, there are two approaches:

1) We can transform every higher order differential equation into a system of first order differential equations, which can be treated as usual. This approach is useful only if we know the derivatives at the initial value $t_0$, which is the case in example (4-61).

---

4 Note that general differential equations can be easily transformed into autonomous systems using the first transformation rule of table 4.5 shown in section 4.5.

$$\begin{aligned} x''(t) &= -x(t) \\ x(t_0) &= x_0 \\ x'(t_0) &= x_0' \end{aligned} \tag{4-61}$$

$$\downarrow \quad x'(t) \rightarrow a(t)$$

$$\begin{aligned} x'(t) &= a(t) \\ a'(t) &= -x(t) \\ x(t_0) &= x_0 \\ a(t_0) &= x'(t_0) = x_0' \end{aligned} \tag{4-62}$$

2) In example (4-63), however, there are two boundary conditions given for x(t) instead of the initial conditions for x(t) and x'(t). Therefore, the first approach is not applicable to this case.

$$\begin{aligned} x''(t) &= -x(t) \\ x(t_0) &= x_0 \\ x(t_1) &= x_1 \end{aligned} \tag{4-63}$$

There is a basic problem with systems like (4-63). This is revealed when we look at equations (4-64) generated by successive differentiation:

$$\begin{aligned} x(t_0) &= x_0 \\ x(t_1) &= x_1 \\ x''(t) &= -x(t) \\ x'''(t) &= -x'(t) \\ x^{(k)}(t) &= \begin{cases} (-1)^{\frac{k}{2}} \cdot x(t), & \text{for even } k \\ (-1)^{\frac{k-1}{2}} \cdot x'(t), & \text{for odd } k \end{cases} \end{aligned} \tag{4-64}$$

For first order systems, the derivatives of the approximation function $\Phi_K$ satisfy the differential equation exactly up to order K at initial value $t_0$. This is possible because the equations generated by successive differentiation specify precise conditions for all derivatives of $\Phi_K$ up to order K. For the second order differential equation (4-64), however, there is no condition for the first derivative $\Phi'_K$, because it is replaced by a boundary condition in the differential equation. Therefore, any approximation error of the derivative x'(t) will be duplicated in all higher derivatives of odd order.

Since the exact derivative is not available, it is very important that a good approximation is used instead. The following solution provides good estimates of the derivative: If the variable described by the differential

equation is sampled periodically, the time series of the past evolution of the process is available as in figure 4.2.

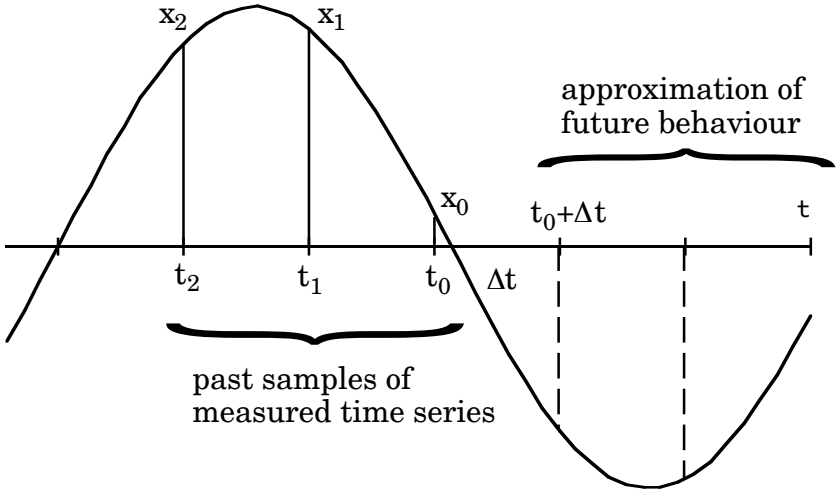



Fig. 4.2 : Time series of a process variable

A natural way to approximate the derivative numerically is to take the derivative of a Lagrange interpolation (4-65) of the past samples $x_0..x_I$.

$$x(t) \approx \Psi(t; t_0..t_M; x_0..x_M) = \sum_{i=0}^{M} x_i \cdot \prod_{\substack{j=0 \\ j \neq i}}^{M} \frac{t - t_j}{t_i - t_j}, \qquad (4\text{-}65)$$

where $M - 1 \geq$ the order m of the differential equation

The derivatives $\Psi^{(k)}$ at initial value $t_0$ of the Lagrange interpolation (4-65) are linear in the samples $x_0..x_M$. They are substituted for the unknown derivatives of $\Phi_K$, which finally results in a multi-step numerical integration procedure using several past values for the next integration step. In case the points $x_k$ are distributed on the grid $t_k = t_0 + k \cdot \Delta t, \Delta t < \rho$, an integration step is performed by executing one iteration of the map (4-66). Note that the step size $\Delta t$ must be smaller than the convergence radius $\rho$.

$$\begin{aligned} x(t_0 - k \cdot \Delta t) &= x_{-k}, k = 0..M \\ x_{k+1} &= \varphi_\Phi(\Delta t; x_{k-M}..x_k) \end{aligned} \qquad (4\text{-}66)$$

In the next two sections, we substitute the neural networks $\Phi_K^{MLN}$ and $\Phi_K^{RBF}$ for the local approximation function $\Phi_K$, and we discuss the resulting neural network architectures.

### 4.6.1 Modified Logistic Networks

It is not quite clear, which parts of a Modified Logistic Network should actually be called a *unit* or *neuron*. We differ from [Cott90], who decomposes a Modified Logistic Network into a network of specialized units, and we call a unit with activation function (4-67) a *Modified Logistic Unit*. This terminology is consistent with the generalization of the logistic function (4-68) to function (4-67).

$$y(\bar{x};\bar{z}_1..\bar{z}_I) = \frac{1}{1+\sum_{i=1}^{I} e^{-\bar{z}_i\cdot\bar{x}}} \tag{4-67}$$

$$y(\bar{x};\bar{z}) = \frac{1}{1+e^{-\bar{z}\cdot\bar{x}}} \tag{4-68}$$

It has been shown in section 4.5 that the transformation of differential equations into normal form may generate equations with polynomial solutions for the parameters $a_k$ of a Modified Logistic Network. Therefore, the most general architecture for a Modified Logistic Network of the form (4-26) needs a polynomial preprocessing layer. It feeds its outputs into a layer of Modified Logistic Units, whose outputs are summed up in a linear output unit (see figure 4.3).

Additionally, there are direct connections from the input to the output layer. All weights in the network are determined by the differential equation to be approximated, and they are computed by the method described in section 4.3. If we approximate a system of several differential equations, there is one such neural network for each variable, and the past values of all variables are used as inputs to the networks.

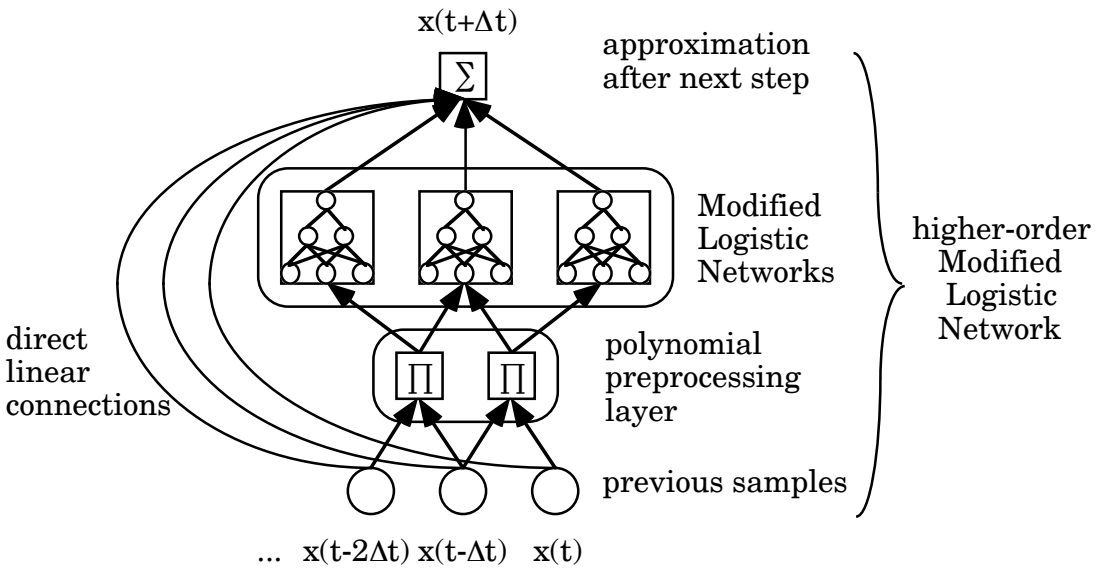



Fig. 4.3 : Architecture of Modified Logistic Network

Please recall the form of the Modified Logistic Network (4-69) described in section 4.4.1:

$$\Phi_K^{MLG}(t;t_0;a_0,\ldots,a_K) = a_0 + \sum_{k=1}^{K}\left[\tanh(t-t_0)^{k-1}\cdot\sum_{i=0}^{\left\lfloor\frac{K}{2}\right\rfloor}\frac{\tanh(a_k\cdot(t-t_0))^{2\cdot i+1}}{2\cdot i+1}\right] \qquad (4\text{-}69)$$

*Theorem* (4.8): According to the definition of a Modified Logistic Unit (4-67), a network (4-69) with K parameters $a_k$ and constant $\Delta t = t - t_0$ needs $K\cdot\left(\left\lfloor\frac{K}{2}\right\rfloor+1\right)\cdot\left(2\cdot\left\lfloor\frac{K}{2}\right\rfloor+1\right)$ units in the middle layer, which increases with $O(K^3)$ relative to the number of parameters $a_k$.

Proof:

The function *tanh* can be represented by one Modified Logistic Unit because of identity $\tanh(x) = 1 - \frac{2}{1+e^{2x}}$. Since the product of two Modified Logistic Units is again a Modified Logistic Unit, the power $\tanh(x)^i$ can be represented by $\sum_{j=1}^{i} j = \frac{i\cdot(i+1)}{2}$ units. The sum of powers of $\tanh(x)^{2\cdot i+1}$ in (4-69) needs only as many units as the highest power of *tanh*, because the lower powers are contained therein and their

units can be merged. Consequently, the total number of needed Modified Logistic Units is $K \cdot \left( \left\lfloor \frac{K}{2} \right\rfloor + 1 \right) \cdot \left( 2 \cdot \left\lfloor \frac{K}{2} \right\rfloor + 1 \right)$. ❑

### 4.6.2 Radial Basis Function Networks

If we use the Radial Basis Function Network (4-70) presented in section 4.4.2 for the local approximation function $\Phi_K$, the resulting architecture in figure 4.4 looks similar to the higher-order Modified Logistic Network, except for the layer of Modified Logistic Units, which is replaced by a layer of Gaussian units of the form (4-34).

$$\Phi_K^{RBF}(t;t_0;a_0,\ldots,a_K) = a_0 + \sum_{k=1}^{K} \left[ \left( \frac{1}{e} - e^{-((t-t_0)-1)^2} \right)^{k-1} \cdot \sum_{i=1}^{K} \frac{b_i \cdot e^{i-1}}{2^i \cdot i!} \cdot \left( \frac{1}{e} - e^{-(a_k \cdot (t-t_0)-1)^2} \right)^i \right] \qquad (4\text{-}70)$$

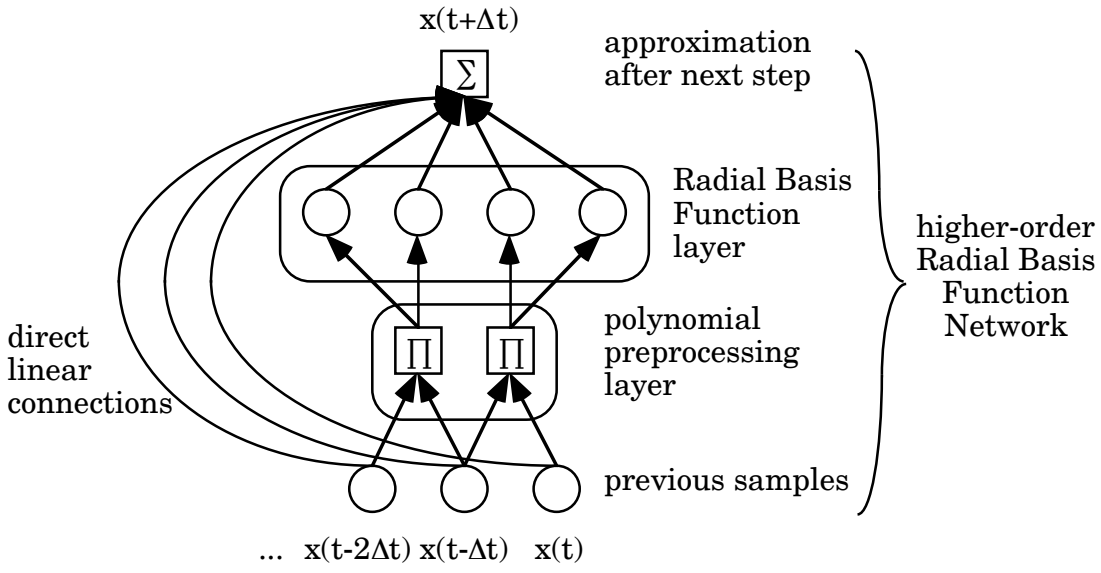



Fig. 4.4 : Architecture of Radial Basis Function Network

*Theorem* (4.9): A Radial Basis Function Network (4-70) with K parameters $a_k$ and constant $\Delta t = t - t_0$ needs $K \cdot \frac{K \cdot (K+1)}{2} = \frac{K^3 + K^2}{2}$ Gaussian units in the middle layer. Therefore, the network size increases with $O(K^3)$ relative to the number of parameters $a_k$.

Proof:
Since the product of two Gaussian units is again a Gaussian unit, the power $\left(\frac{1}{e} - e^{-(a_k \cdot (t-t_0)-1)^2}\right)^i$ can be represented by $\sum_{j=1}^{i} j = \frac{i \cdot (i+1)}{2}$ units. The sum of such powers in (4-70) needs only as many units as the highest power in (4-70), because the lower powers are contained therein and their units can be merged. Consequently, the total number of needed Gaussian units is $K \cdot \frac{K \cdot (K+1)}{2}$. ❑

## 4.7 Summary of the Neural Network Design Algorithm

Our method to design neural networks for the approximation of differential equations can be summarized as follows:

### 4.7.1 Single-Step Integration Procedure

1. In order to construct a *single-step numerical integration procedure* using neural networks, a system of differential equations first has to be transformed into a system of *autonomous first order ordinary differential equations of polynomial form* using the transformation rules described in section 4.5. We get a system of equations of the form (4-71), with initial conditions (4-72).

$$x_i'(t) = P_i(x_1(t), \ldots, x_N(t)),\ i = 1..N \quad \text{where } P_i \text{ is a polynom in its arguments} \tag{4-71}$$

$$x_i(t_0) = x_{i,0},\ i = 1..N \tag{4-72}$$

2. For each function $x_i(t)$, we select an approximation function (4-73) satisfying the conditions discussed in section 4.3. For the construction of single-step integration procedures, either Modified Logistic Networks of the form (4-26) or Radial Basis Function Networks of the form (4-40) can be used (see sections 4.4.1 or 4.4.2). Note that we could even mix different neural network architectures for different functions $x_i(t)$.

$$x_i(t) \approx \Phi_K^i(t; t_0; a_{i,0}, a_{i,1}, \ldots, a_{i,K}), i = 1..N \tag{4-73}$$

3. We successively differentiate the differential equations (4-71) according to equations (4-74).

$$x_i^{(k)}(t_0) = P_i(x_1(t),\ldots,x_N(t))^{(k-1)}\Big|_{t=t_0}, i = 1..N, k = 1..K \tag{4-74}$$

Since the approximation functions satisfy the triangulation condition (4-75), the combination of (4-74) with initial conditions (4-72) results in the full equation system (4-76) that has to be solved for the $N\cdot(K+1)$ parameters $a_{i,k}$.

$$\Phi_K^i(t;t_0;a_{i,0},a_{i,1},\ldots,a_{i,K})^{(k)}\Big|_{t=t_0} = \Theta_k^{\Phi_i}(a_{i,0}\ldots a_{i,k}), k \le K \tag{4-75}$$

$$\begin{aligned}\Theta_0^{\Phi_i}(a_{i,0}) &= x_{i,0}, i = 1..N\\ \Theta_k^{\Phi_i}(a_{i,0}\ldots a_{i,k}) &= P_i(\Phi_K^1(t;t_0;a_{1,0},\ldots,a_{1,K}),\ldots,\Phi_K^N(t;t_0;a_{N,0},\ldots,a_{N,K}))^{(k-1)}\Big|_{t=t_0}\\ &= g_{i,k-1}(t_0;\Theta_0^{\Phi_1}(a_{1,0}),\ldots,\Theta_{k-1}^{\Phi_N}(a_{N,0}\ldots a_{N,k-1})), i = 1..N, k = 1..K\end{aligned} \tag{4-76}$$

4. The resulting equations (4-76) are iteratively solved for the coefficients $a_{i,k}$ by (4-77). They generate the truncated local approximations (4-78) of the solution of the differential equations in the neighborhood of the initial value $t_0$.

$$a_{i,k} = \Theta_k^{\Phi_i\,-1}(g_{i,k-1}(t_0;\Theta_0^{\Phi_1}(a_{1,0}),\ldots,\Theta_{k-1}^{\Phi_N}(a_{N,0}\ldots a_{N,k-1}));a_{i,0}\ldots a_{i,k-1}) \tag{4-77}$$

$$x_i(t) \approx \Phi_K^i(t;t_0;a_{i,0},\ldots,a_{i,K}), i = 1..N \tag{4-78}$$

5. The local approximation functions (4-78) are used for the single-step integration procedure (4-79), which approximates the differential equations on the grid $t_k=t_0+k\cdot\Delta t$ for $\Delta t$ within the convergence radius $\rho$.

$$x_i(t_0 + \Delta t) \approx \Phi_K^i(t_0 + \Delta t;t_0;a_{i,0},\ldots,a_{i,K}), i = 1..N, \Delta t < \rho \tag{4-79}$$

Since functions $\Phi_K^i(t_0 + \Delta t;t_0;a_{i,0},\ldots,a_{i,K})$ are independent of $t_0$ for both types of neural networks, the single-step method can be simplified to an iterated map (4-80).

$$\begin{aligned}x_i(t_0) &= x_{i,0}, i = 1..N\\ x_{i,k+1} &= \varphi_{\Phi i}(\Delta t;x_{1,k},\ldots,x_{N,k})\\ &\approx x_i(t_0 + (k+1)\cdot\Delta t), i = 1..N\end{aligned} \tag{4-80}$$

After expanding the polynoms in $\Phi_K^i$, the resulting neural networks correspond to the architectures described in sections 4.6.1 and 4.6.2.

### 4.7.2 Multi-Step Integration Procedure

Alternatively, if the differential equations are of higher order and boundary conditions are given instead of initial conditions, we can construct a *multi-step numerical integration procedure* without a preceding transformation into first order equations:

1. The system of differential equations first has to be transformed into a system of *autonomous ordinary differential equations of polynomial form* using the transformation rules described in section 4.5. The equations should then be of the form (4-81), with initial conditions (4-82).

$$x_i^{(m_i)}(t) = P_i(x_1(t),\ldots,x_N(t)),\ i = 1..N \quad \text{where } P_i \text{ is a polynom in its arguments} \tag{4-81}$$

$$x_i(t_0 - k\cdot\Delta t) = x_{i,-k},\ i = 1..N, k = 1..M_i, \text{where } M_i \geq m_i \tag{4-82}$$

2. For each function $x_i(t)$, we select an approximation function (4-83) satisfying the conditions discussed in section 4.3. For the construction of multi-step integration procedures, either Modified Logistic Networks of the form (4-26) or Radial Basis Function Networks of the form (4-40) can be used (see sections 4.4.1 or 4.4.2).

$$x_i(t) \approx \Phi_K^i(t;t_0;a_{i,0},a_{i,1},\ldots,a_{i,K}), i = 1..N \tag{4-83}$$

3. We successively differentiate the differential equations (4-81) according to equations (4-84).

$$x_i^{(m_i+k)}(t_0) = P_i(x_1(t),\ldots,x_N(t))^{(k)}\Big|_{t=t_0}, i = 1..N, k = 0..K - m_i \tag{4-84}$$

The derivatives $x_i^{(k)}(t_0)$ for $k=1..m_i\text{-}1$ are approximated by the derivatives (4-86) of the Lagrange interpolation functions (4-85).

$$\Psi_i(t;t_{-M_i}..t_0;x_{i,-M_i}..x_{i,0}) = \sum_{j=-M_i}^{0} x_{i,j}\cdot\prod_{\substack{k=-M_i\\k\neq j}}^{0}\frac{t-t_k}{t_j-t_k}, \tag{4-85}$$

where $M_i - 1 \geq$ the order $m_i$ of the differential equation for $x_i(t)$

$$x_i^{(k)}(t_0) \approx \Psi_i(t;t_{-M_i}..t_0;x_{i,-M_i}..x_{i,0})^{(k)}\Big|_{t=t_0}, k < m_i \tag{4-86}$$

Since the approximation functions $\Phi_K^i$ satisfy the triangulation condition (4-87), the combination of (4-84) with initial conditions (4-82) results in the full equation system (4-88) which must be solved for the N·(K+1) parameters $a_{i,k}$.

$$\Phi_K^i(t;t_0;a_{i,0},a_{i,1},\ldots,a_{i,K})^{(k)}\Big|_{t=t_0} = \Theta_k^{\Phi_i}(a_{i,0}\ldots a_{i,k}) \tag{4-87}$$

$$\begin{aligned}
\Theta_0^{\Phi_i}(a_{i,0}) &= x_{i,0}, i = 1..N \\
\Theta_k^{\Phi_i}(a_{i,0}\ldots a_{i,k}) &= \Psi_i(t;t_{-M_i}..t_0;x_{i,-M_i}..x_{i,0})^{(k)}\Big|_{t=t_0} \\
&= \sum_{j=-M_i}^{0} d_{i,j}\cdot x_{i,j}, \text{for } i = 1..N, k = 1..m_i - 1 \\
\Theta_k^{\Phi_i}(a_{i,0}\ldots a_{i,k}) &= P_i(\Phi_K^1(t;t_0;a_{1,0},..,a_{1,K}),..,\Phi_K^N(t;t_0;a_{N,0},..,a_{N,K}))^{(k-1)}\Big|_{t=t_0} \\
&= g_{i,k-1}(t_0;\Theta_0^{\Phi_1}(a_{1,0}),\ldots,\Theta_{k-1}^{\Phi_N}(a_{N,0}\ldots a_{N,k-1})), \\
&\text{for } i = 1..N, k = m_i..K
\end{aligned} \tag{4-88}$$

4. The resulting equations (4-88) are iteratively solved for the coefficients $a_{i,k}$ by (4-89). They generate the local approximations (4-90) of the solution of the differential equations in the neighborhood of the initial value $t_0$.

$$a_{i,k} = \Theta_k^{\Phi_i\,-1}(g_{i,k-1}(t_0;\Theta_0^{\Phi_1}(a_{1,0}),\ldots,\Theta_{k-1}^{\Phi_N}(a_{N,0}\ldots a_{N,k-1}));a_{i,0}\ldots a_{i,k-1}) \tag{4-89}$$

$$x_i(t) \approx \Phi_K^i(t;t_0;a_{i,0},\ldots,a_{i,K}), i = 1..N \tag{4-90}$$

5. The local approximation functions (4-90) are used for the multi-step integration procedure (4-91), which approximates the differential equations on the grid $t_0+k\cdot\Delta t$ for $\Delta t$ within the convergence radius $\rho$.

$$x_i(t_0+\Delta t) \approx \Phi_K^i(t_0+\Delta t;t_0;a_{i,0},\ldots,a_{i,K}), i = 1..N, \Delta t < \rho \tag{4-91}$$

Since the functions $\Phi_K^i(t_0+\Delta t;t_0;a_{i,0},\ldots,a_{i,K})$ are independent of $t_0$ and $\Delta t$ remains constant, the multi-step method can be simplified to the iterated map (4-92).

$$\begin{aligned}
x_i(t_0 - k\cdot\Delta t) &= x_{i,-k}, i = 1..N, k = 0..M_i \\
x_{i,k+1} &= \varphi_{\Phi_i}(\Delta t;x_{i,k-M_i}..x_{i,k}) \\
&\approx x_i(t_0+(k+1)\cdot\Delta t), i = 1..N
\end{aligned} \tag{4-92}$$

After expanding the polynoms in $\Phi_K^i$, the resulting neural networks correspond to the architectures described in sections 4.6.1 and 4.6.2.

### 4.7.3 Examples

The following examples illustrate the individual steps presented in the previous sections 4.7.1 and 4.7.2 that are necessary to construct neural networks for the approximation of differential equations.

**Example 1: First Order Differential Equation**

The first example shows how to construct a *single-step numerical integration procedure* for the *first order differential equation* (4-93):

$$\begin{aligned} x'(t) &= -x(t) \\ x(t_0) &= x_0 \end{aligned} \tag{4-93}$$

1. Since the differential equation is already in polynomial form, no transformation is necessary.

2. We have to choose an approximation function of type (4-73) for the differential equation (4-93). We select the Radial Basis Function Network (4-94) with four parameters $a_0..a_3$ for that purpose (see section 4.4.2).

$$\Phi_3^{RBF}(t;t_0;a_0,\ldots,a_3) = a_0 + \sum_{k=1}^{3}\left[\left(\frac{1}{e} - e^{-((t-t_0)-1)^2}\right)^{k-1} \cdot \sum_{i=1}^{3}\frac{b_i \cdot e^{i-1}}{2^i \cdot i!} \cdot \left(\frac{1}{e} - e^{-(a_k\cdot(t-t_0)-1)^2}\right)^i\right] \tag{4-94}$$

3. If we substitute the approximation function $\Phi_3^{RBF}$ for $x(t)$ in equations (4-95) generated by successive differentiation, we get the equation system (4-96) for the parameters $a_k$.

$$\begin{aligned} x(t_0) &= x_0 \\ x'(t_0) &= -x(t_0) \\ x''(t_0) &= -x'(t_0) \\ x'''(t_0) &= -x''(t_0) \end{aligned} \tag{4-95}$$

$$\begin{aligned} a_0 &= x_0 \\ \frac{-a_1}{e} &= -a_0 \\ \frac{4 \cdot a_2}{e^2} &= \frac{a_1}{e} \\ \frac{6 \cdot a_2}{e^2} - \frac{24 \cdot a_3}{e^3} &= \frac{-4 \cdot a_2}{e^2} \end{aligned} \tag{4-96}$$

4. Equations (4-96) are solved for the parameters $a_k$, resulting in the values (4-97).

$$\begin{aligned} a_0 &= x_0 \\ a_1 &= x_0 \cdot e \\ a_2 &= \frac{x_0 \cdot e^2}{4} \\ a_3 &= \frac{5 \cdot x_0 \cdot e^3}{48} \end{aligned} \tag{4-97}$$

5. The single-step integration procedure (4-98) is completed by inserting the resolved parameters $a_k$ into $\Phi_3^{RBF}$.

$$x(t_0 + \Delta t) \approx \Phi_3^{RBF}(t_0 + \Delta t; t_0; a_0, \ldots, a_3), \Delta t < \rho \tag{4-98}$$

6. Figure 4.5 compares the results computed by the single-step procedure with the exact solution $x(t)=e^{-t}$ using the parameters $t_0=0$, $x_0=1$, $\Delta t=0.1$. The approximation error is shown in figure 4.6.

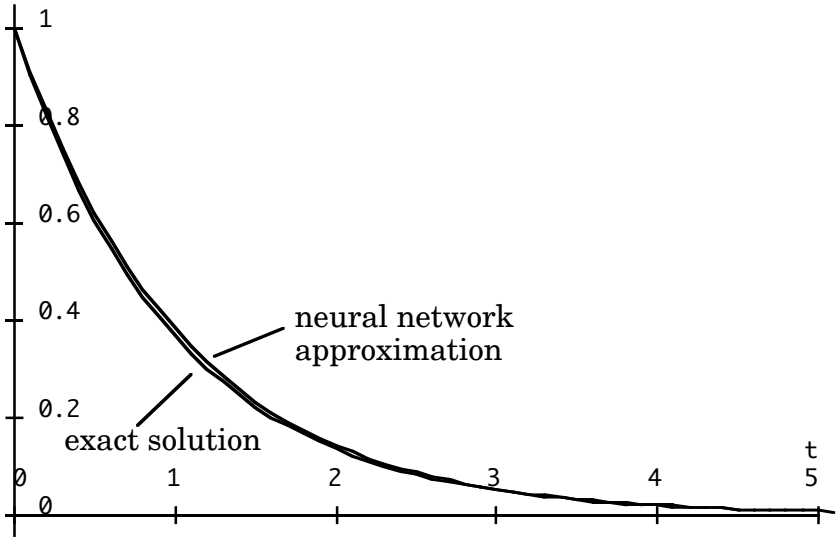



Fig. 4.5 : Approximation and exact solution

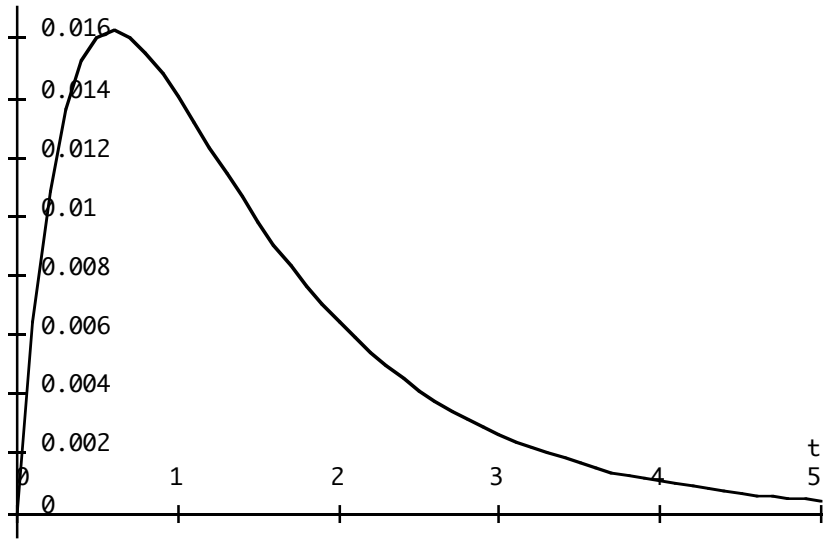



Fig. 4.6 : Approximation error

The approximation by the neural network is already good for $\Delta t=0.1$. It can be improved by increasing the number of parameters $a_k$ or by reducing the step size $\Delta t$.

### Example 2: Second Order Differential Equation

The second example shows how to construct a *multi-step numerical integration procedure* for the *second order differential equation* (4-99):

$$\begin{aligned} x''(t) &= -x(t) \\ x(t_0) &= x_0 \\ x(t_1) &= x_1 \end{aligned} \tag{4-99}$$

1. The differential equation is already in polynomial form, therefore, no transformation is necessary.

2. We have to choose an approximation function of type (4-83) for the second order differential equation (4-99). We select a Modified Logistic Network (4-100) with four parameters $a_0..a_3$ for that purpose (see section 4.4.1).

$$\Phi_3^{MLG}(t;t_0;a_0,\ldots,a_3) = a_0 + \sum_{k=1}^{3}\left[\tanh(t-t_0)^{k-1}\cdot\sum_{i=0}^{1}\frac{\tanh(a_k\cdot(t-t_0))^{2\cdot i+1}}{2\cdot i+1}\right] \tag{4-100}$$

3. The equations (4-101) generated by successive differentiation of (4-99) require the first derivative of x(t). It is approximated by the derivative of the Lagrange interpolation function (4-102), using the samples $x_0$ and $x_1$.

$$\begin{aligned} x''(t_0) &= -x(t_0) \\ x'''(t_0) &= -x'(t_0) \\ x(t_0) &= x_0 \\ x(t_1) &= x_1 \end{aligned} \tag{4-101}$$

$$\Psi(t;t_0,t_1;x_0,x_1) = x_0 \cdot \frac{t-t_1}{t_0-t_1} + x_1 \cdot \frac{t-t_0}{t_1-t_0} \tag{4-102}$$

If we substitute the approximation function $\Phi_3^{MLG}$ for x(t) in the equations (4-101), and if we replace $t_1 = t_0 - \Delta t$ and $x'(t_0) = \Psi'(t;t_0,t_1;x_0,x_1)\big|_{t=t_0}$, we get the full equation system (4-103) for the parameters $a_k$.

$$\begin{aligned} a_0 &= x_0 \\ a_1 &= \frac{x_0 - x_1}{\Delta t} \\ 2 \cdot a_2 &= -a_0 \\ 6 \cdot a_3 &= -a_1 \end{aligned} \tag{4-103}$$

4. Equations (4-103) are solved for the parameters $a_k$, resulting in the values (4-104).

$$\begin{aligned} a_0 &= x_0 \\ a_1 &= \frac{x_0 - x_1}{\Delta t} \\ a_2 &= -\frac{1}{2} \cdot x_0 \\ a_3 &= -\frac{1}{6} \cdot \frac{x_0 - x_1}{\Delta t} \end{aligned} \tag{4-104}$$

5. The multi-step integration procedure (4-105) is completed by inserting the resolved parameters $a_k$ into $\Phi_3^{MLG}$.

$$x(t_0 + \Delta t) \approx \Phi_3^{MLG}(t_0 + \Delta t; a_0, \ldots, a_3), \Delta t < \rho \tag{4-105}$$

6. Figure 4.7 compares the results computed by the multi-step procedure with the exact solution x(t)=sin(t), where the parameters $t_0$=0, $x_0$=0, $t_1$=-0.1, $x_1$=sin(-0.1), $\Delta t$=0.1 are used. Obviously, the approximated solution diverges quickly from the exact solution.

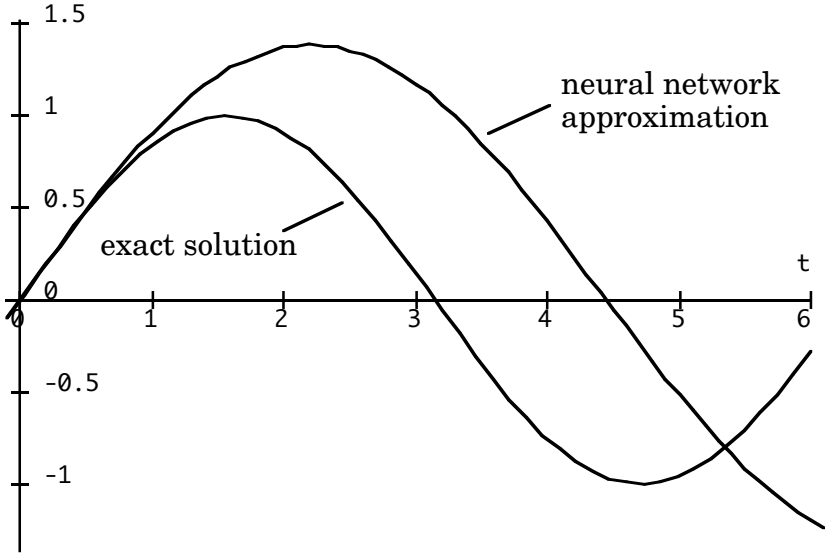



Fig. 4.7 : Approximation and exact solution (2 samples for ψ)

After increasing the number of past samples used for the Lagrange interpolation function ψ from 2 to 4, the approximation of the first derivative of x(t) is improved considerably. Now, the results in figures 4.8 and 4.9 show only small differences between the neural network output and the exact solution.

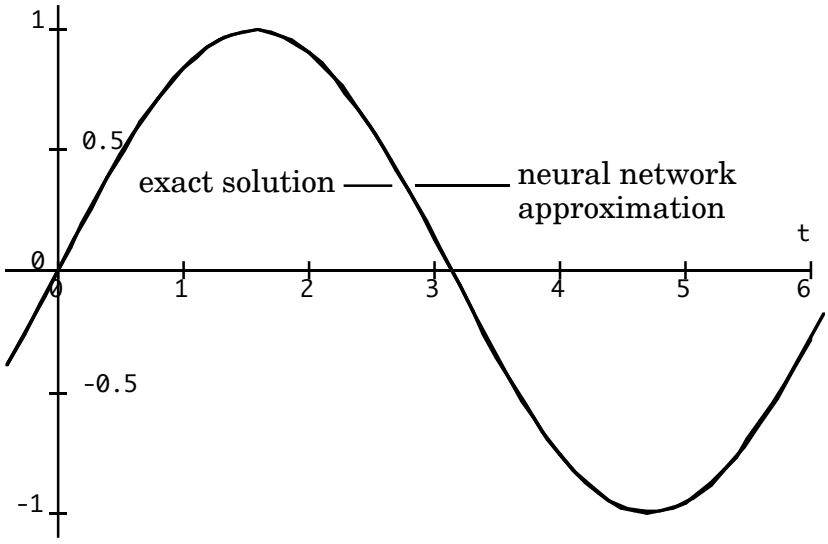



Fig. 4.8 : Approximation and exact solution (4 samples for ψ)

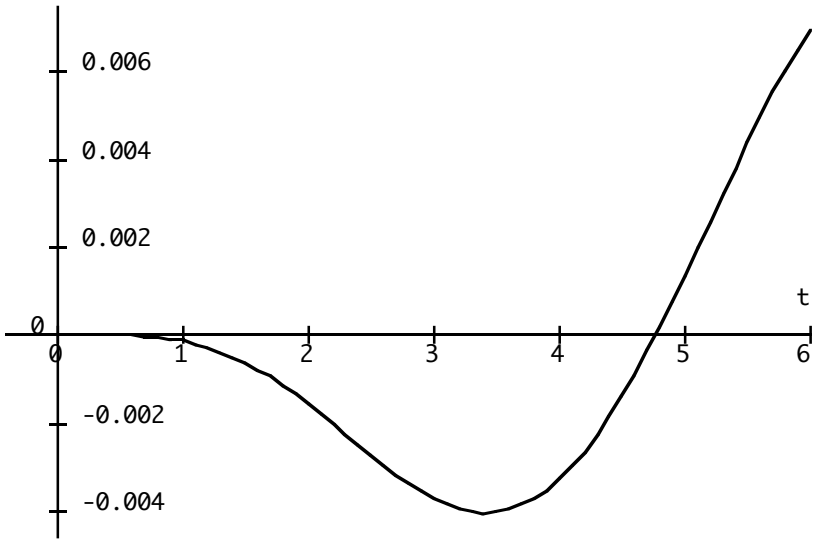



Fig. 4.9 : Approximation error (4 samples for $\psi$)

## 4.8 Training Designed Neural Networks with Observation Data

After initial design of a neural network with differential equations, there are several possibilities to exploit its learning capabilities. The next two sections show how to improve the modeling accuracy of a network and how to estimate the parameters of a differential equation.

### 4.8.1 Improving the Modeling Accuracy of a Network

The approximation quality of a neural network designed with our method depends on several factors. If the given differential equations and their physical parameters describe a dynamic system correctly, approximation errors are only introduced during the construction of the neural network. The size of the chosen network (the number of included parameters $a_k$) and the step size $\Delta t$ determine the achievable accuracy.

On the other hand, for modeling a dynamic system with differential equations, it is often necessary to rely on simplifying assumptions. Usually, not all influencing factors are known and approximation errors are introduced additionally by misspecified differential equations.

If enough observation data of the dynamic system is available, we can train the neural network to reduce its approximation errors. First, an error function E is constructed that measures the deviation of network

approximations from observations. The mean-squared error (4-106) is typically used for that purpose (see section 2.1):

$$
\begin{aligned}
E &= \frac{1}{2}\sum_{i=1}^{N}\left\|\bar{x}_i - NN_i\right\|_2^2 \\
\bar{x}_i &= \bar{x}(t_0 + i\cdot\Delta t) = \text{ time series of observations} \\
\Delta t &= \text{ time increment between observations} \\
N &= \text{ length of time series} \\
NN_i &= \text{ neural network output for observation i}
\end{aligned}
\tag{4-106}
$$

Then, neural network training tries to minimize error E by adapting the weights in the network. Recall the Modified Logistic Network (4-107) introduced in section 4.4.1 and remember the network architecture described in section 4.6.1:

$$
\Phi_K^{MLG}(t;t_0;a_0,\ldots,a_K) =
$$

$$
a_0 + \sum_{k=1}^{K}\left[\tanh(t-t_0)^{k-1}\cdot\sum_{i=0}^{\left\lfloor\frac{K}{2}\right\rfloor}\frac{\tanh(a_k\cdot(t-t_0))^{2\cdot i+1}}{2\cdot i+1}\right] \tag{4-107}
$$

Note that the weights in the network described by (4-107) are hardwired: The weights of the preprocessing layer (represented by the parameters $a_k$) are derived from the differential equation, whereas the weights of the middle and output layer are fixed. In order to train the network, we have to replace some of the fixed weights by flexible weights that can be adapted during training. There are several possibilities to do that:

1. The neural network represented by $\Phi_K^{MLG}(t;t_0;a_0..a_K)$ can be replaced by a composite network $\tilde{\Phi} = \Phi_K^{MLG}(t;t_0;a_0..a_K) + NN(t;t_0;\bar{w})$, where only the weights $\bar{w}$ of the second network NN are trained. This network learns to compensate the approximation errors of the initial network $\Phi_K^{MLG}$. There are no restrictions on the size or on the architecture of the neural network NN, but the total network will be larger than the initial network $\Phi_K^{MLG}$ of course.

2. If we do not want to increase the size of the neural network, we can simply turn some of the fixed network weights into flexible weights. A good place to do that are the middle and the output layer, because the number of weights is substantially smaller than in the preprocessing layer. If the backpropagation training algorithm is used, the computation of the gradient of the error function is also much easier for these weights than for those in the preprocessing layer. Basically, the arguments of the *tanh* functions in (4-107) are changed according to (4-108):

$$\begin{gathered}\tanh(a\cdot(t-t_0)) = \tanh(a\cdot w-\theta)\\ w^0 = t-t_0 = \Delta t, \theta^0 = 0\end{gathered} \tag{4-108}$$

After introducing the flexible network weights $v_k$, $w_{k,i}$ and the thresholds $\theta_{k,i}$, the network (4-107) turns into network (4-109). If the weights are initialized according to (4-110), the function computed by the two networks is still identical, but network training will adapt the weights of (4-109) in order to minimize the network's approximation error.

$$\begin{gathered}\Phi_K^{MLG}(\bar{a};\bar{v},(w_{k,i}),(\theta_{k,i})) =\\ a_0 + \sum_{k=1}^{K}\left[v_k\cdot\sum_{i=0}^{\left\lfloor\frac{K}{2}\right\rfloor}\frac{\tanh(a_k\cdot w_{k,i}-\theta_{k,i})^{2\cdot i+1}}{2\cdot i+1}\right]\end{gathered} \tag{4-109}$$

$$v_k^0 = \tanh(t-t_0)^{k-1}, w_{k,i}^0 = t-t_0 = \Delta t, \theta_{k,i}^0 = 0 \tag{4-110}$$

The same strategy can be applied to the Radial Basis Function Network (4-111) from section 4.4.2. Introducing the weights $v_k$, $z_{k,i}$ and $\sigma_{k,i}$ turns the network (4-111) into network (4-112). Again, initializing the weights according to (4-113) generates the same network output as (4-111).

$$\begin{gathered}\Phi_K^{RBF}(t;t_0;a_0,\ldots,a_K) =\\ a_0 + \sum_{k=1}^{K}\left[\left(\frac{1}{e}-e^{-((t-t_0)-1)^2}\right)^{k-1}\cdot\sum_{i=1}^{K}\frac{b_i\cdot e^{i-1}}{2^i\cdot i!}\cdot\left(\frac{1}{e}-e^{-(a_k\cdot(t-t_0)-1)^2}\right)^i\right]\end{gathered} \tag{4-111}$$

$$\Phi_K^{RBF}(a_0..a_K; v, (z_{k,i}), (\sigma_{k,i})) =$$

$$a_0 + \sum_{k=1}^{K}\left[ v_k \cdot \sum_{i=1}^{K} \frac{b_i \cdot e^{i-1}}{2^i \cdot i!} \cdot \left( \frac{1}{e} - e^{-\left(\frac{a_k - z_{k,i}}{\sigma_{k,i}}\right)^2} \right)^i \right] \qquad (4\text{-}112)$$

$$v_k^0 = \left( \frac{1}{e} - e^{-((t-t_0)-1)^2} \right)^{k-1}, z_{k,i}^0 = \sigma_{k,i}^0 = \frac{1}{t - t_0} = \frac{1}{\Delta t} \qquad (4\text{-}113)$$

After replacing some of the network's fixed weights by flexible weights, most neural network training algorithms can be applied to reduce approximation error E. Gradient-based algorithms like the backpropagation algorithm described in [Rume86a] are well suited for these problems.

### 4.8.2 Estimating Unknown Parameters of Differential Equations

In some industrial applications, the structure of the differential equations describing a physical process are known, but exact values of the process parameters are unknown. Assume for example, the task is to simulate the inverted pendulum mounted on a cart shown in figure 4.10. The equations (4-114) describe the motion of the pendulum (see [Guez88]).

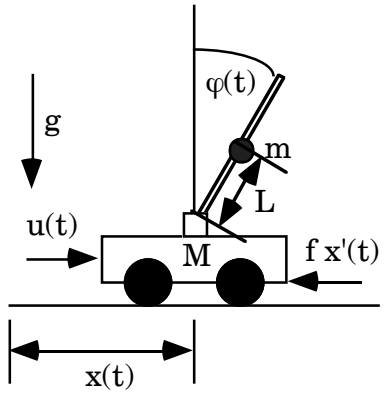



Fig. 4.10 : Cart-pole system

$$\varphi''(t) = \frac{3}{4 \cdot L} \cdot (g \cdot \sin(\varphi(t)) - x''(t) \cdot \cos(\varphi(t)))$$

$$x''(t) = \frac{m \cdot \left( L \cdot \sin(\varphi(t)) \cdot \varphi'(t)^2 - \frac{3}{8} \cdot g \cdot \sin(2 \cdot \varphi(t)) \right) - f \cdot x'(t) + u(t)}{M + m \cdot \left( 1 - \frac{3}{4} \cdot \cos(\varphi(t))^2 \right)} \qquad (4\text{-}114)$$

The exact behavior of the pendulum depends on the physical constants m, M, f, L and g (the masses of the stick and the car, etc.) Their accuracy is

crucial for a good approximation of the pendulum's trajectory. In reality, those parameters are never known precisely, but on-line data taken from real-time observations may allow to improve initial estimations considerably. This makes an adaptive approach based on neural networks worthwhile.

Problems of this type are easy to integrate into our framework: First, we generate a neural network for the approximation of the differential equations with our method described in section 4.7. The unknown parameters become network weights in the preprocessing layer of the network, whose values have to be determined yet. The resulting neural network has only the unknown physical parameters of the differential equation as remaining free variables. Their initial estimation is then improved with an algorithm for training standard multi-layer neural networks (for an overview see [Hert91] or [Hech90]).

The appropriate learning algorithm must be chosen depending on the computing resources that are available. For on-line training, which is done in real-time, the error backpropagation algorithm is often used (see [Rume86a]). For off-line training, however, algorithms such as conjugate gradient or quasi-Newton methods are usually preferred (see [Watr87]). An error function like (4-106) is needed, which measures the deviation of the network output from the real observations. The training algorithm minimizes the network error and it hopefully identifies the correct parameters of the differential equations. Nevertheless, in section 5.3 we will see that the estimation of the correct parameters is more difficult than the minimization of the remaining approximation errors.

# 5. APPLICATION TO FORECASTING OF CHAOTIC TIME SERIES

A traditional application of neural networks is *time series forecasting* (see [Kimo90], [Weig90], [Dorr91], [Wong91] or [Hopt93]). A time series is a series of observations of a dynamic system which are taken at regular time intervals. Since the future behavior of the dynamic system is usually of most interest, we speak of *forecasting* the time series. Neural networks can be used as a black-box model of the unknown process generating the time series, and the network can be trained to forecast the future behavior, given the past observations of the series. Sometimes, the time series results from observing a deterministic physical process whose dynamic behavior is described by differential equations. For such applications, forecasting the future evolution of the series is equivalent to approximating the solution of the differential equation, given the current state as initial condition.

In this context, our neural network design algorithm can be used to construct neural networks for forecasting the future behavior of time series. In case the knowledge of the generating process is complete, the network will be fully specified by the design algorithm. If the knowledge is incomplete however, the network will contain free parameters, which can be determined by using common neural network learning algorithms. In this way, any knowledge in the form of differential equations can be incorporated into the neural network and the discrepancies to the real process can be reduced or even removed by additional learning.

In this chapter, neural networks will be constructed for forecasting some time series generated by chaotic differential equations. In science, *chaos* is used as a synonym for *irregular behavior*, whose long-term development is essentially *unpredictable* (see [Schu88]). Chaotic differential equations exhibit not only irregular behavior, but they are also unstable with respect to small perturbations of their initial conditions. Consequently, it is difficult to forecast the future of a time series based on chaotic differential

equations, and they should be a good benchmark for a neural network design algorithm. We test the performance of the resulting neural networks for the Lorenz system, the Rössler system and the circular pendulum system of differential equations.

## 5.1 Lorenz System

The Lorenz system (5-1) of differential equations is an idealization of a hydrodynamic system, which serves as a simplified model for studying turbulence in fluids (see [Lore63]):

$$\begin{aligned} x'(t) &= \sigma \cdot y(t) - \sigma \cdot x(t) \\ y'(t) &= -x(t) \cdot z(t) + r \cdot x(t) - y(t) \\ z'(t) &= x(t) \cdot y(t) - b \cdot z(t) \\ x(t_0) &= x_0 \\ y(t_0) &= y_0 \\ z(t_0) &= z_0 \end{aligned} \tag{5-1}$$

We use the notation (5-2), which embeds the Lorenz system into our framework:

$$\begin{aligned} x_1'(t) &= A \cdot (x_2(t) - x_1(t)) \\ x_2'(t) &= -x_1(t) \cdot x_3(t) + B \cdot x_1(t) - x_2(t) \\ x_3'(t) &= x_1(t) \cdot x_2(t) - C \cdot x_3(t) \\ x_1(t_0) &= x_{1,0} \\ x_2(t_0) &= x_{2,0} \\ x_3(t_0) &= x_{3,0} \end{aligned} \tag{5-2}$$

Lorenz has shown that the behavior of system (5-2) becomes nonperiodic for some parameter sets. Moreover, it is unstable with respect to small modifications of the initial conditions, which is characteristic for chaotic systems.

The following figures 5.1 and 5.2 illustrate the evolution of the Lorenz system for the critical parameters A=10, B=28, C=8/3. Note that for this chapter, all reference models are generated by using the numerical

integration procedure *D02PCF* of the NAG-library[5] (see [NAG94]). It is based on an adaptive Runge-Kutta method.

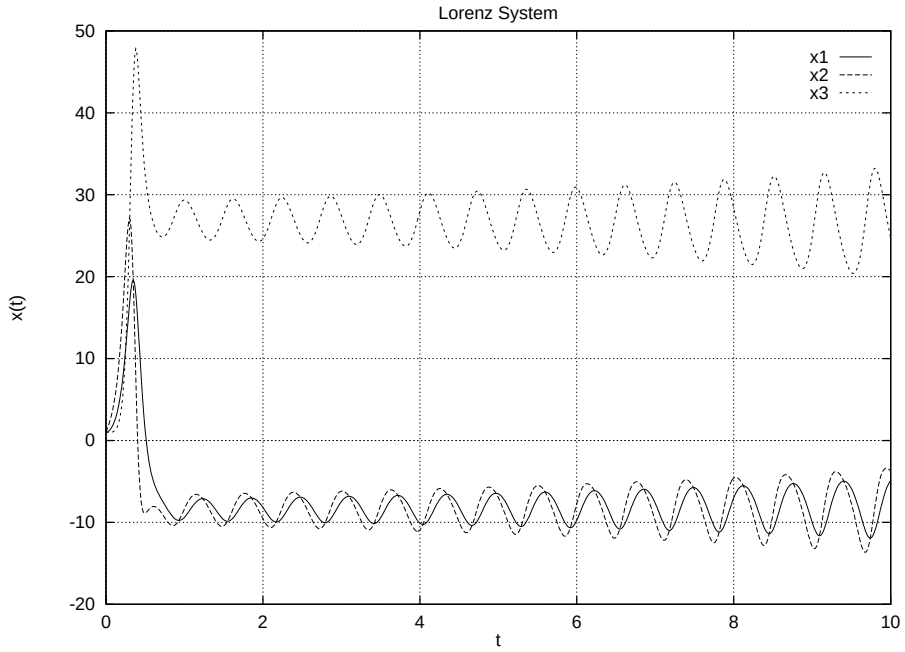



Fig. 5.1 : Evolution of the Lorenz system

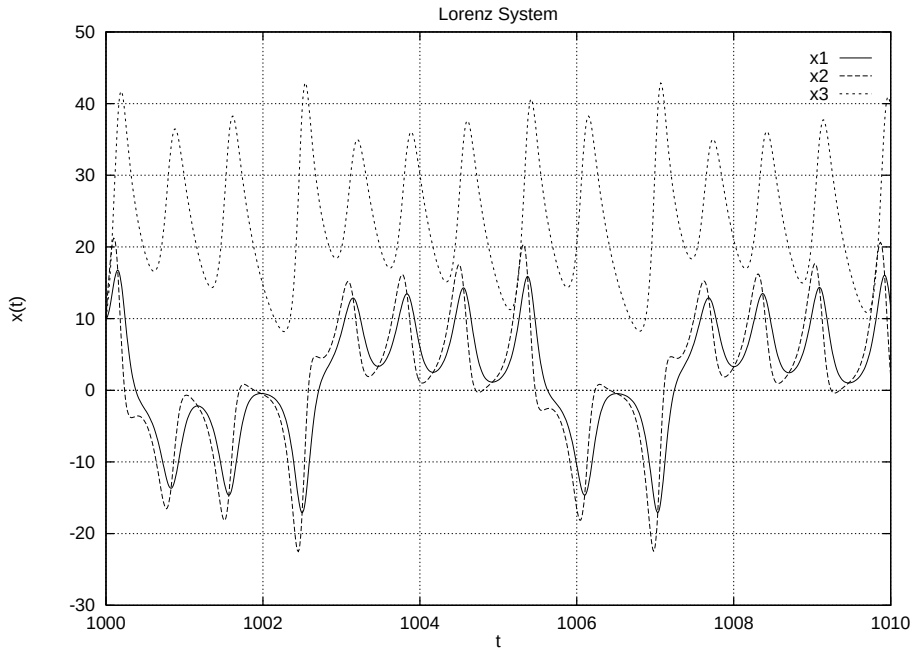



Fig. 5.2 : Evolution of the Lorenz system after $t=10^3$

5 NAG is a registered trademark of The Numerical Algorithms Group Limited, Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, United Kingdom.

Figure 5.1 shows the evolution of the Lorenz system with initial conditions $t_0=0$, $x_{1,0}=x_{2,0}=x_{3,0}=1$. First, periodic stationary behavior seems to emerge, but figure 5.2 reveals that after some time, nonperiodic chaotic behavior appears.

We now develop a neural network for the approximation of the Lorenz system according to the algorithm summarized in section 4.7:

1. Since the differential equations are already in polynomial form, no transformation is necessary.

2. For each variable $x_1(t)..x_3(t)$, we have to choose an approximation function $\Phi_K^i$. We select the Radial Basis Function Networks (5-3) with five parameters $a_{i,0}..a_{i,4}$ for that purpose (see section 4.4.2).

$$x_i(t) \approx \Phi_4^{RBF_i}(t;t_0;a_{i,0},\ldots,a_{i,4}), i = 1..3 \tag{5-3}$$

3. We substitute approximation functions $\Phi_4^{RBF_i}$ for $x_i(t)$ in the equations generated by successive differentiation, and we get an equation system for the parameters $a_{i,k}$ (omitted).

4. The equation system is solved for the parameters $a_{i,k}$, resulting in the values (5-4.1) to (5-6.5).

$$a_{1,0} = x_{1,0} \tag{5-4.1}$$

$$a_{1,1} = \left(x_{1,0} - x_{2,0}\right) \cdot A \cdot e \tag{5-4.2}$$

$$a_{1,2} = \left(x_{1,0} \cdot (A + B) - x_{2,0} \cdot (A + 1) - x_{1,0} \cdot x_{3,0}\right) \cdot \frac{A \cdot e^2}{4} \tag{5-4.3}$$

$$a_{1,3} = \left( \begin{array}{l} x_{1,0} \cdot (3 \cdot A + 5 \cdot B + 4 \cdot A \cdot B + 2 \cdot A^2) - \\ x_{2,0} \cdot (5 + 5 \cdot A + 2 \cdot A \cdot B + 2 \cdot A^2) - \\ x_{1,0} \cdot x_{3,0} \cdot (5 + 4 \cdot A + 2 \cdot C) + \\ x_{2,0} \cdot x_{3,0} \cdot 2 \cdot A + {x_{1,0}}^2 \cdot x_{2,0} \cdot 2 \end{array} \right) \cdot \frac{A \cdot e^3}{48} \tag{5-4.4}$$

$$a_{1,4} = \left( \begin{array}{l} x_{1,0} \cdot (A \cdot (10 + A \cdot (4 + A + 3 \cdot B) + 10 \cdot B + B^2) + 15 \cdot B) - \\ x_{2,0} \cdot (15 + A \cdot (15 + 6 \cdot B + A \cdot (5 + 2 \cdot B + A))) - \\ x_{1,0} \cdot x_{3,0} \cdot (15 + A \cdot (10 + 2 \cdot B + 3 \cdot C + 3 \cdot A) + 5 \cdot C + C^2) + \\ x_{2,0} \cdot x_{3,0} \cdot (6 \cdot A + 2 \cdot A \cdot C + 2 \cdot A^2) + \\ {x_{1,0}}^2 \cdot x_{2,0} \cdot (6 + 4 \cdot A + C) - \\ x_{1,0} \cdot {x_{2,0}}^2 \cdot 3 \cdot A + x_{1,0} \cdot {x_{3,0}}^2 \cdot A - B \cdot {x_{1,0}}^3 + {x_{1,0}}^3 \cdot x_{3,0} \end{array} \right) \cdot \frac{A \cdot e^4}{192} \qquad (5\text{-}4.5)$$

$$a_{2,0} = x_{2,0} \qquad (5\text{-}5.1)$$

$$a_{2,1} = -\left( x_{1,0} \cdot B - x_{2,0} - x_{1,0} \cdot x_{3,0} \right) \cdot e \qquad (5\text{-}5.2)$$

$$a_{2,2} = -\left( \begin{array}{l} x_{1,0} \cdot (B + A \cdot B) - x_{2,0} \cdot (A \cdot B + 1) - \\ x_{1,0} \cdot x_{3,0} \cdot (A + C + 1) + \\ x_{2,0} \cdot x_{3,0} \cdot A + {x_{1,0}}^2 \cdot x_{2,0} \end{array} \right) \cdot \frac{e^2}{4} \qquad (5\text{-}5.3)$$

$$a_{2,3} = -\left( \begin{array}{l} x_{1,0} \cdot (A \cdot (5 \cdot B + 2 \cdot A \cdot B + 2 \cdot B^2) + 5 \cdot B) - \\ x_{2,0} \cdot (5 + 7 \cdot A \cdot B + 2 \cdot A^2 \cdot B) - \\ x_{1,0} \cdot x_{3,0} \cdot (5 + A \cdot (5 + 4 \cdot B + 4 \cdot C + 2 \cdot A) + 5 \cdot C + 2 \cdot C^2) + \\ x_{2,0} \cdot x_{3,0} \cdot (A \cdot (7 + 2 \cdot A + 4 \cdot C)) + \\ {x_{1,0}}^2 \cdot x_{2,0} \cdot (7 + 6 \cdot A + 2 \cdot C) - x_{1,0} \cdot {x_{2,0}}^2 \cdot 6 \cdot A + \\ x_{1,0} \cdot {x_{3,0}}^2 \cdot 2 \cdot A - {x_{1,0}}^3 \cdot 2 \cdot B + {x_{1,0}}^3 \cdot x_{3,0} \cdot 2 \end{array} \right) \cdot \frac{e^3}{48} \qquad (5\text{-}5.4)$$

$$a_{2,4} = -\left( \begin{array}{l} x_{1,0} \cdot (A \cdot (A \cdot (A \cdot B + 5 \cdot B + 2 \cdot B^2) + 15 \cdot B + 6 \cdot B^2) + 15 \cdot B) - \\ x_{2,0} \cdot (15 + A \cdot (21 \cdot B + A \cdot (6 \cdot B + A \cdot B + B^2))) - \\ x_{1,0} \cdot x_{3,0} \cdot (15 + A \cdot (15 + A \cdot (5 + A + 4 \cdot B + 3 \cdot C) + 12 \cdot B) + \\ C \cdot (A \cdot (10 + 4 \cdot B + 3 \cdot C) + 15 + 5 \cdot C + C^2)) + \\ x_{2,0} \cdot x_{3,0} \cdot (A \cdot (21 + A \cdot (6 + A + 2 \cdot B + 3 \cdot C) + 13 \cdot C + 3 \cdot C^2)) + \\ {x_{1,0}}^2 \cdot x_{2,0} \cdot (21 + A \cdot (20 + 7 \cdot A + 11 \cdot B + 4 \cdot C) + 6 \cdot C + C^2) - \\ x_{1,0} \cdot {x_{2,0}}^2 \cdot (A \cdot (24 + 10 \cdot A + 4 \cdot C)) + \\ x_{1,0} \cdot {x_{3,0}}^2 \cdot (A \cdot (6 + 2 \cdot A + 4 \cdot C)) + \\ {x_{2,0}}^3 \cdot 3 \cdot A^2 - {x_{1,0}}^3 \cdot (B \cdot (6 + 6 \cdot A + C)) + \\ {x_{1,0}}^3 \cdot x_{3,0} \cdot (6 + 6 \cdot A + 2 \cdot C) - \\ {x_{1,0}}^2 \cdot x_{2,0} \cdot x_{3,0} \cdot 11 \cdot A - x_{2,0} \cdot {x_{3,0}}^2 \cdot A^2 - {x_{1,0}}^4 \cdot x_{2,0} \end{array} \right) \cdot \frac{e^4}{192} \qquad (5\text{-}5.5)$$

$$a_{3,0} = x_{3,0} \tag{5-6.1}$$

$$a_{3,1} = \left(x_{3,0} \cdot C - x_{1,0} \cdot x_{2,0}\right) \cdot e \tag{5-6.2}$$

$$a_{3,2} = -\left(\begin{array}{l} \left(-x_{3,0} \cdot C^2 + x_{1,0} \cdot x_{2,0} \cdot (A + C + 1) - \right. \\ \left. x_{1,0}{}^2 \cdot B - x_{2,0}{}^2 \cdot A + x_{1,0}{}^2 \cdot x_{3,0} \right) \end{array}\right) \cdot \frac{e^2}{4} \tag{5-6.3}$$

$$a_{3,3} = -\left(\begin{array}{l} \left(-x_{3,0} \cdot (3 \cdot C^2 + 2 \cdot C^3) + \right. \\ x_{1,0} \cdot x_{2,0} \cdot (5 + A \cdot (7 + 2 \cdot A + 8 \cdot B + 2 \cdot C) + 5 \cdot C + 2 \cdot C^2) - \\ x_{1,0} \cdot x_{2,0} \cdot x_{3,0} \cdot 8 \cdot A + x_{1,0}{}^2 \cdot (B \cdot (2 \cdot C - 6 \cdot A - 5)) - \\ x_{2,0}{}^2 \cdot (A \cdot (9 + 2 \cdot C + 2 \cdot A)) + \\ \left. x_{1,0}{}^2 \cdot x_{3,0} \cdot (5 + 6 \cdot A + 4 \cdot C) - x_{1,0}{}^3 \cdot x_{2,0} \cdot 2 \right) \end{array}\right) \cdot \frac{e^3}{48} \tag{5-6.4}$$

$$a_{3,4} = -\left(\begin{array}{l} \left(-x_{3,0} \cdot (C^2 \cdot (10 + 4 \cdot C + C^2)) + \right. \\ x_{1,0} \cdot x_{2,0} \cdot (15 + A \cdot (21 + A \cdot (7 + A + 12 \cdot B) + \\ 28 \cdot B + C \cdot (6 + A + 4 \cdot B + C)) + C \cdot (15 + 5 \cdot C + C^2)) - \\ x_{1,0} \cdot x_{2,0} \cdot x_{3,0} \cdot (A \cdot (28 + 12 \cdot A + 10 \cdot C)) - \\ x_{1,0}{}^2 (A \cdot (B \cdot (16 + 7 \cdot A + 4 \cdot B + 3 \cdot C)) + B \cdot (15 + 5 \cdot C + C^2)) - \\ x_{2,0}{}^2 (A \cdot (29 + A \cdot (8 + A + 4 \cdot B) + C \cdot (7 + A + C))) + \\ x_{1,0}{}^2 \cdot x_{3,0} \cdot (15 + A \cdot (16 + 7 \cdot A + 8 \cdot B + 8 \cdot C) + 10 \cdot C + 3 \cdot C^2) + \\ x_{2,0}{}^2 \cdot x_{3,0} \cdot 4 \cdot A^2 + x_{1,0}{}^2 \cdot x_{2,0}{}^2 \cdot 7 \cdot A - x_{1,0}{}^2 \cdot x_{3,0}{}^2 \cdot 4 \cdot A - \\ \left. x_{1,0}{}^3 \cdot x_{2,0} \cdot (6 + 6 \cdot A + 2 \cdot C) + x_{1,0}{}^4 \cdot B - x_{1,0}{}^4 \cdot x_{3,0} \right) \end{array}\right) \cdot \frac{e^4}{192} \tag{5-6.5}$$

5. The single-step integration procedure (5-7) is completed by inserting the resolved parameters $a_{i,k}$ into $\Phi_4^{RBF_i}$.

$$\begin{aligned} x_i(t_0 + \Delta t) &\approx \Phi_4^{RBF_i}(t_0 + \Delta t; t_0; a_{i,0}, \ldots, a_{i,4}) \\ &= \hat{\Phi}_4^{RBF_i}(\Delta t; x_{1,0}, x_{2,0}, x_{3,0}; A, B, C) \end{aligned} \tag{5-7}$$

### 5.1.1 One-Step-Ahead Forecasts

For our first experiment, we use the single-step integration procedure (5-7) for generating one-step-ahead forecasts of the Lorenz system. During each integration step, the last state observation of the system is entered into the neural network to predict the new state after the next time step. Figure 5.3

illustrates the approximation quality that can be achieved with this method, showing only small approximation errors in figure 5.4 (A=10, B=28, C=8/3, $t_0$=0, $x_{1,0}$=$x_{2,0}$=$x_{3,0}$=1, $\Delta t$=0.001).

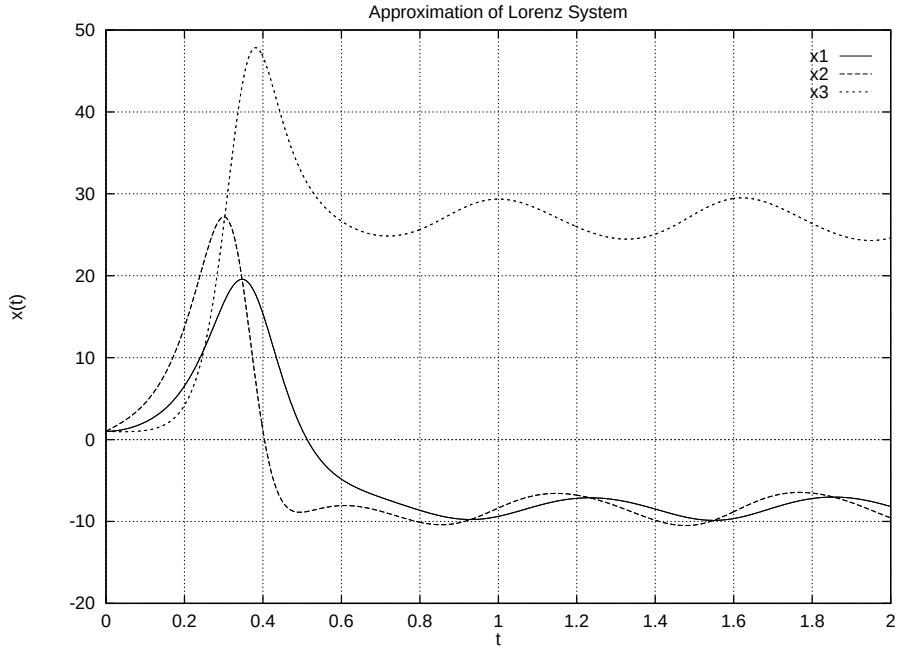



Fig. 5.3 : One-step-ahead forecasts and reference model ($t_0$=0)

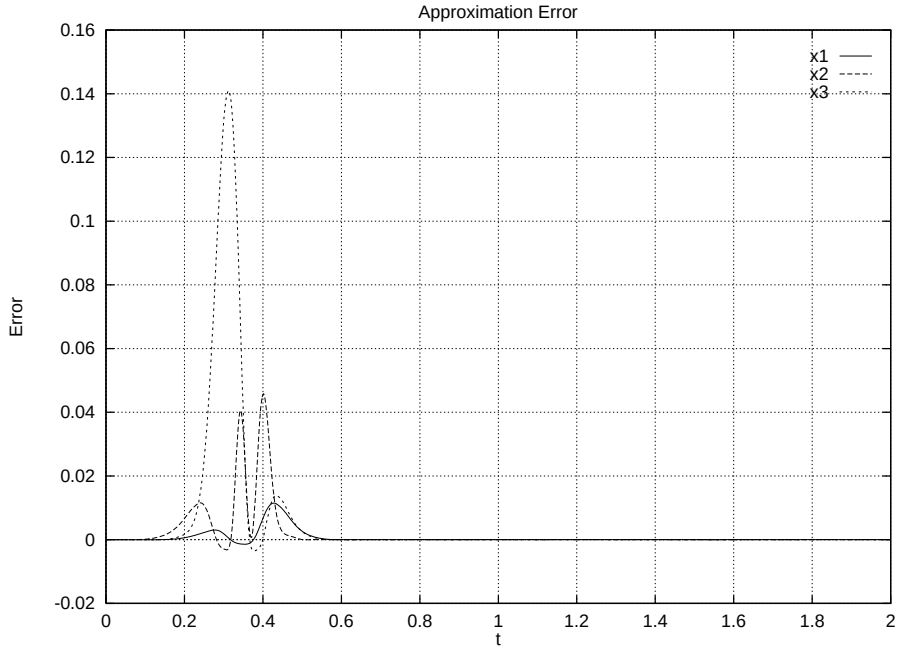



Fig. 5.4 : Approximation error of one-step-ahead forecasts ($t_0$=0)

Figure 5.5 shows the one-step-ahead forecasts for the parameters A=10, B=28, C=8/3, $t_0$=$10^3$, $x_{1,0}$=$x_{2,0}$=$x_{3,0}$=10, $\Delta t$=0.001. The approximation errors are displayed in figure 5.6.

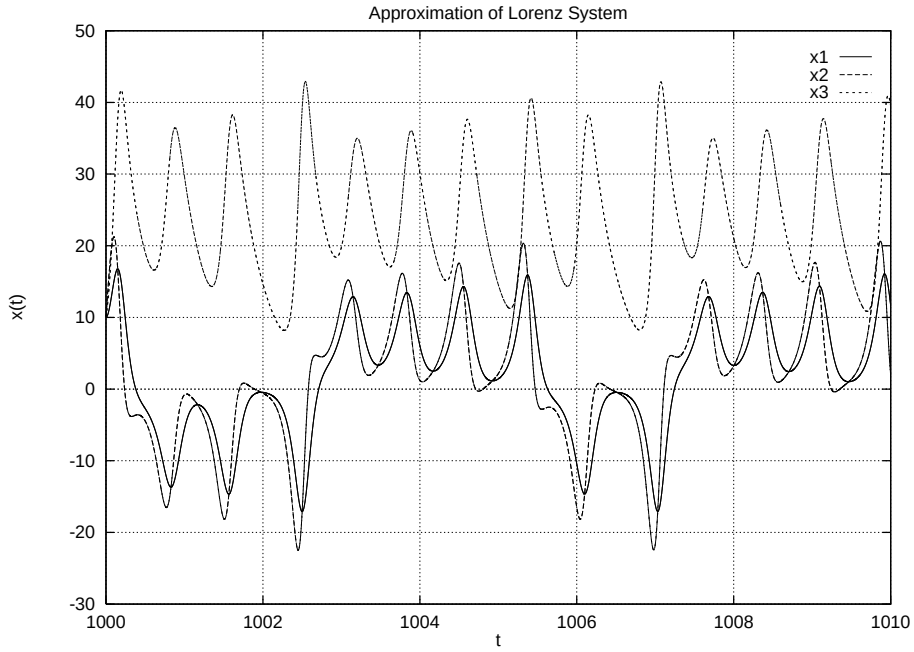



Fig. 5.5 : One-step-ahead forecasts and reference model ($t_0=10^3$)

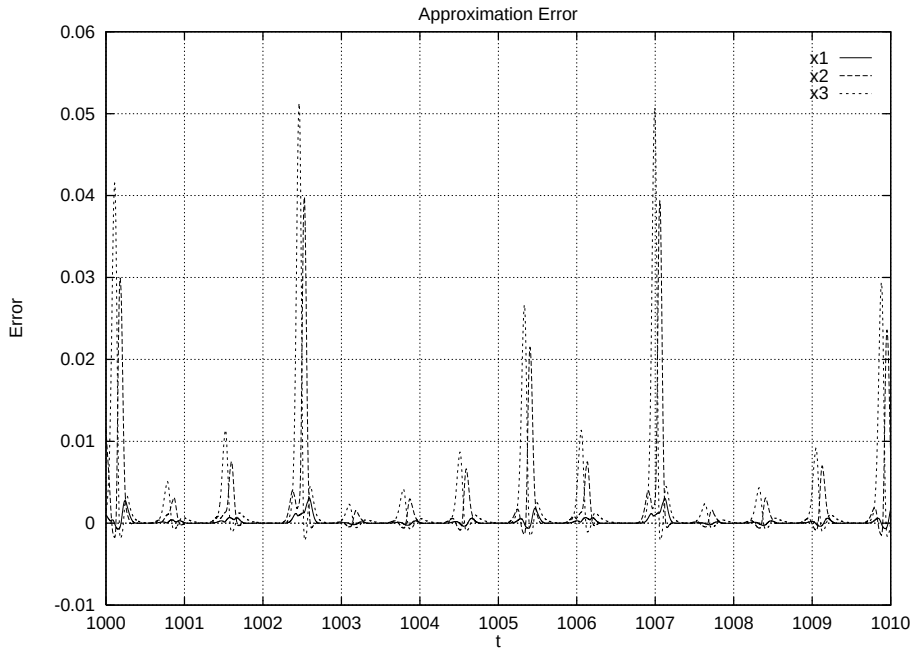



Fig. 5.6 : Approximation error of one-step-ahead forecasts ($t_0=10^3$)

In general, the approximation errors of the one-step-ahead forecasts are quite small, but they are maximal wherever the trajectory of the Lorenz system changes direction.

### 5.1.2 Repeated Forecasts

If forecasts for time horizons longer than one time step should be obtained, the standard approach of iterating a one-step-ahead forecasting procedure can be used. This is equivalent to iterating the single-step integration procedure for the approximation of differential equations, where intermediate approximation errors accumulate. Therefore, only the initial state is taken from the observations and all subsequent forecasts are generated by feeding back the previous forecasts as input into the network. Figure 5.7 compares the repeated forecasts of the neural network with the reference model. Again, we use the parameters A=10, B=28, C=8/3, $t_0$=0, $x_{1,0}$=$x_{2,0}$=$x_{3,0}$=1 with a step size of $\Delta t$=0.001. The approximation errors are shown in figure 5.8. The repeated forecasts diverge soon from the reference model, but the approximation error remains bounded.

The first neural network for the approximation of the Lorenz system is based on Radial Basis Function Networks. Of course, Modified Logistic Networks discussed in section 4.4.1 can be chosen as well. The next experiment shown in figure 5.9 demonstrates that a Modified Logistic Network provides better repeated forecasts of the Lorenz system with step size $\Delta t$=0.001 than a Radial Basis Function Network. The approximation errors for the Modified Logistic Network are shown in figure 5.10.

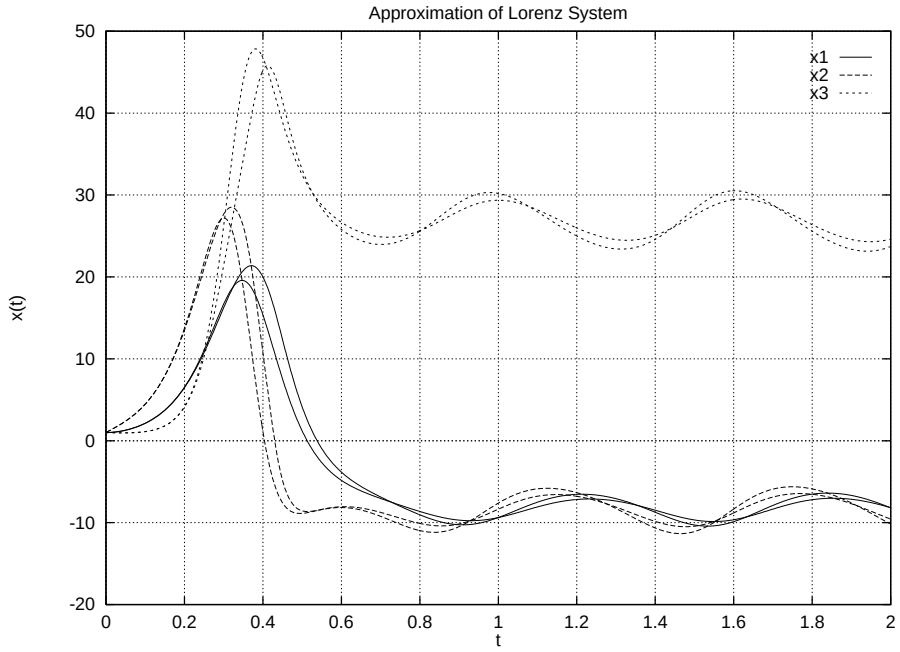



Fig. 5.7 : Repeated forecasts and reference model ($t_0$=0)

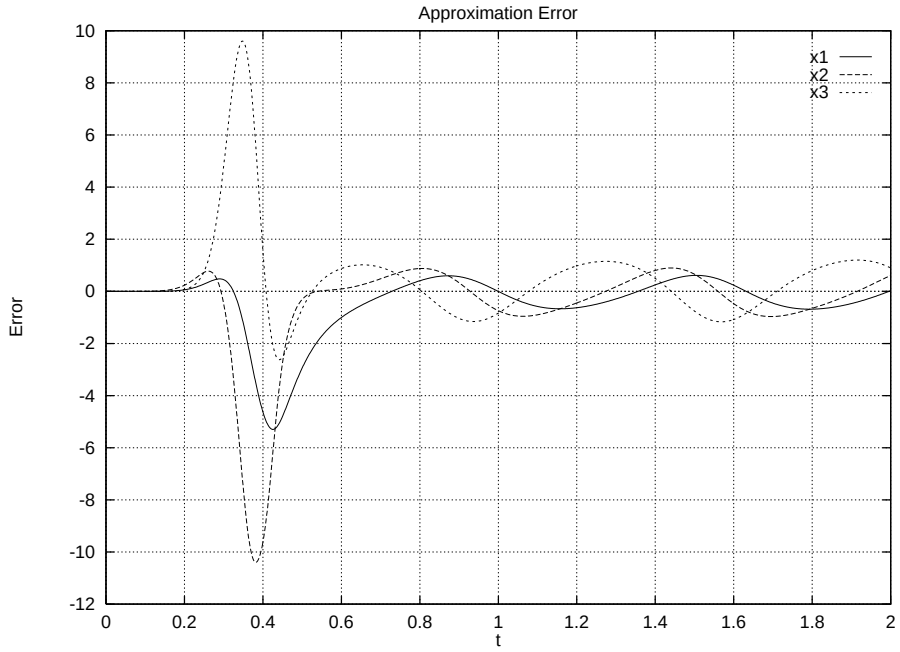



Fig. 5.8 : Approximation error of repeated forecasts ($t_0$=0)

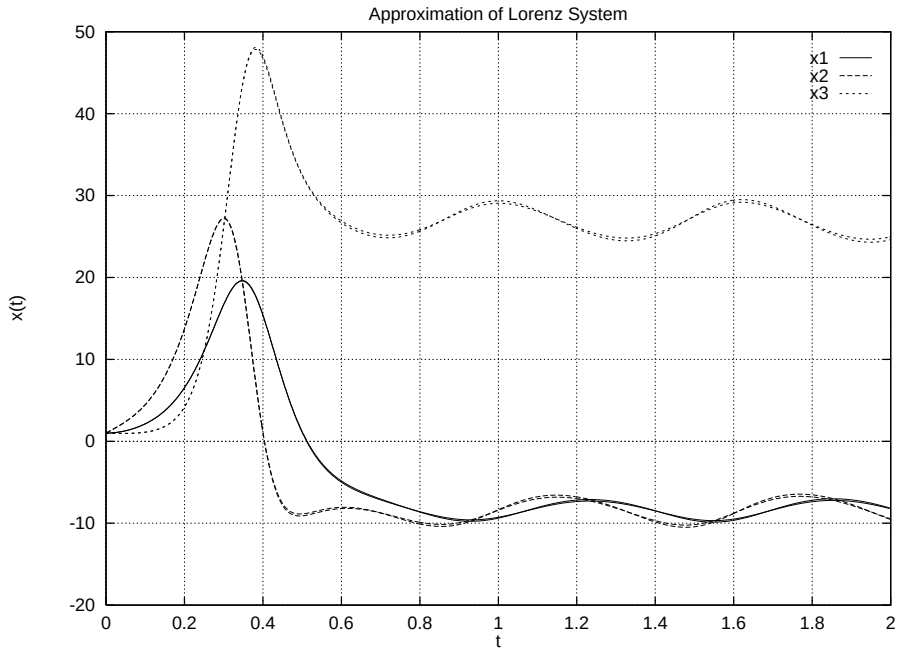



Fig. 5.9 : Repeated forecasts of Modified Logistic Network ($t_0$=0)

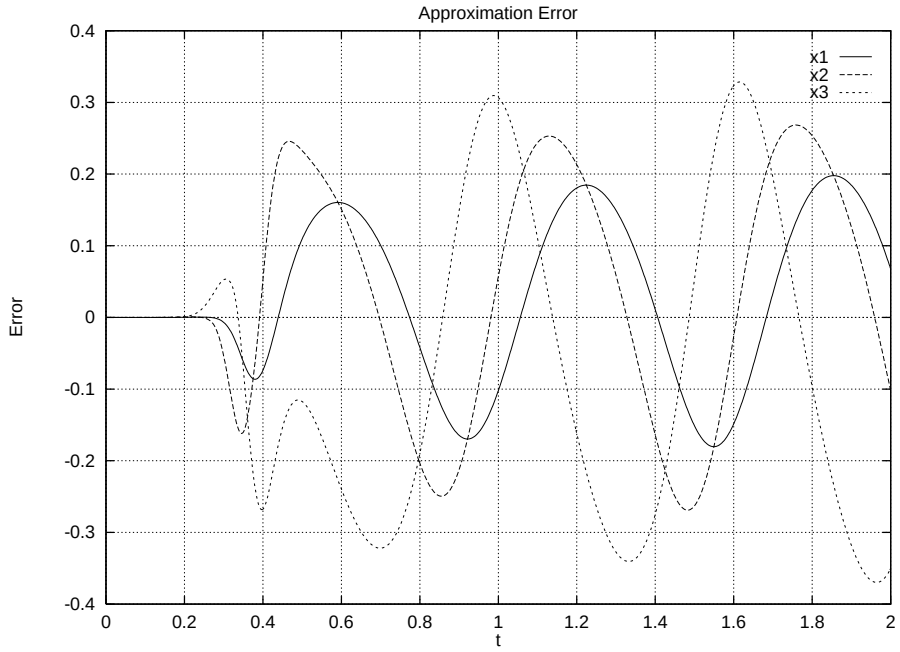



Fig. 5.10 : Approximation error of Modified Logistic Network ($t_0$=0)

We repeat both experiments for the parameters $t_0=10^3$, $x_{1,0}=x_{2,0}=x_{3,0}=10$. Figure 5.11 shows the repeated forecasts of the Radial Basis Function Network. Its approximation errors are displayed in figure 5.12.

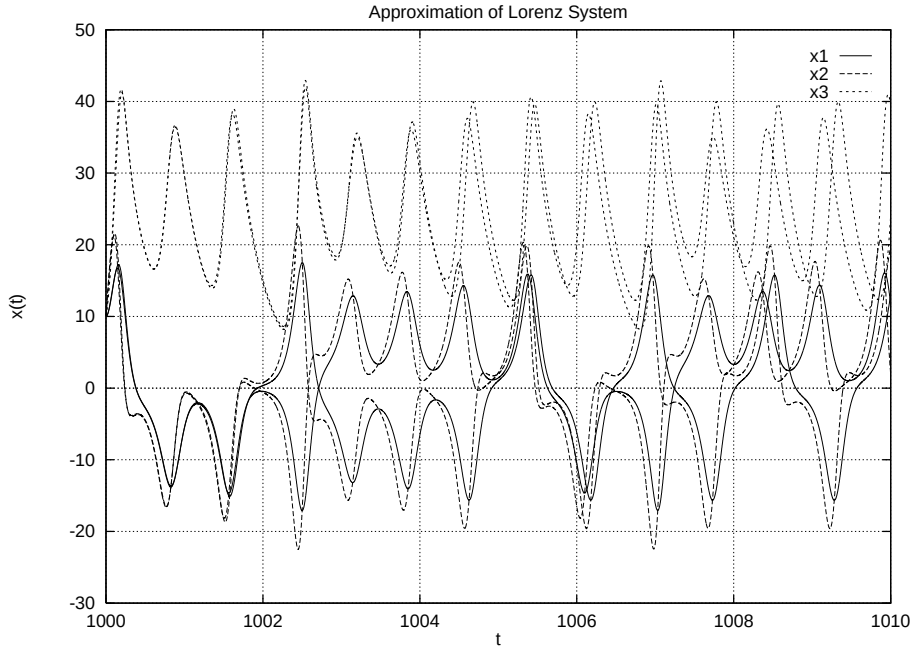



Fig. 5.11 : Repeated forecasts of Radial Basis Function Network ($t_0=10^3$)

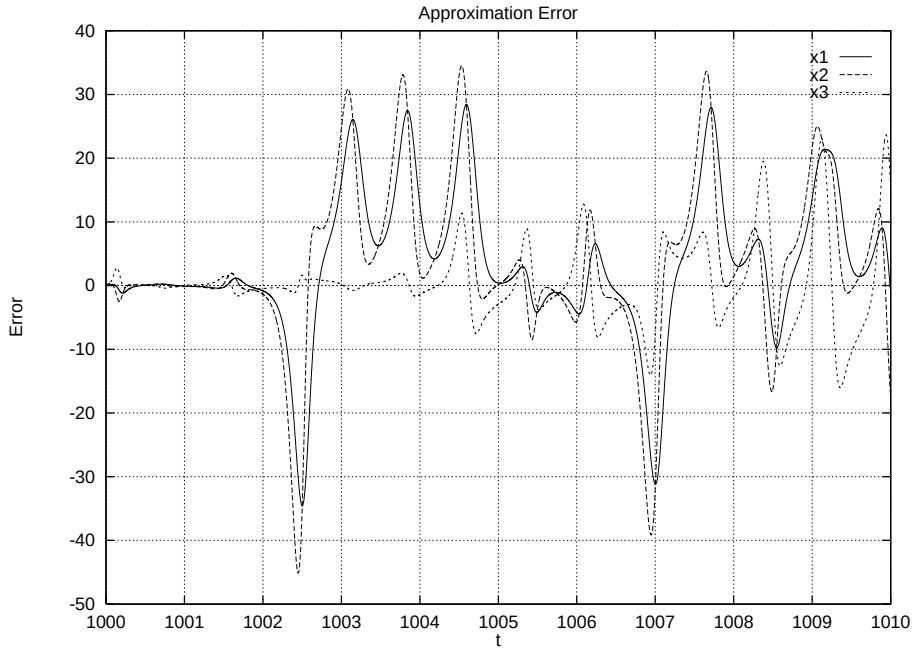



Fig. 5.12 : Approximation error of Radial Basis Function Network ($t_0=10^3$)

Figure 5.13 displays the forecasts of the Modified Logistic Network for $t_0=10^3$, where the corresponding approximation errors are illustrated in figure 5.14.

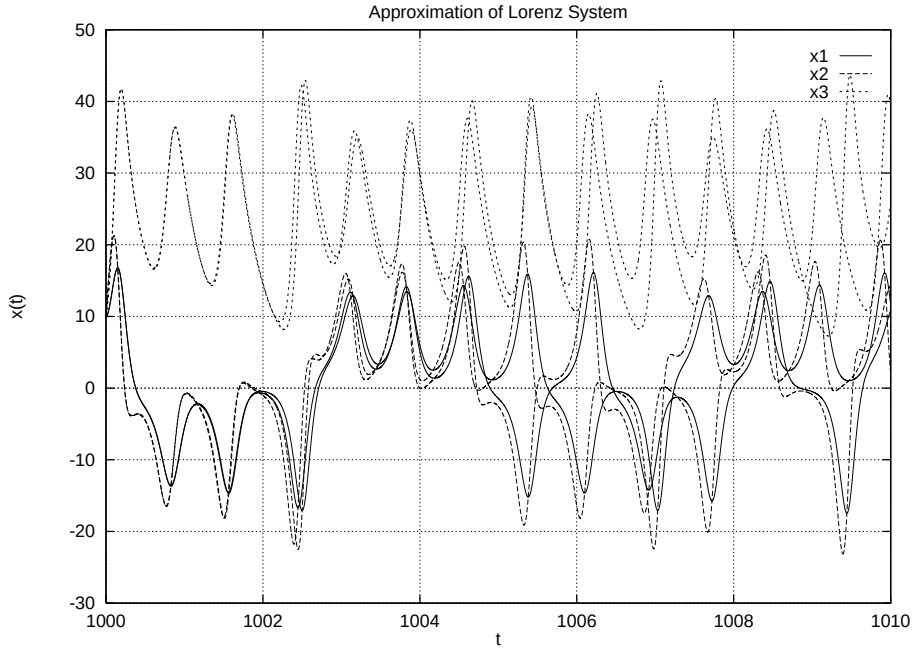



Fig. 5.13 : Repeated forecasts of Modified Logistic Network ($t_0=10^3$)

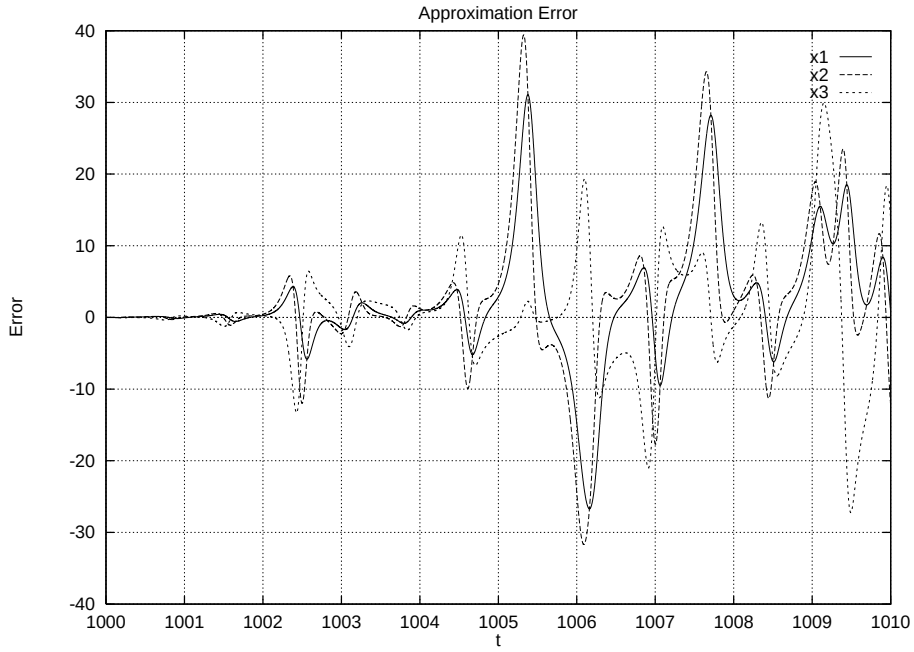



Fig. 5.14 : Approximation error of Modified Logistic Network ($t_0=10^3$)

The repeated forecasts of the Modified Logistic Network exhibit superior long-term behavior, because they reach the error level of the Radial Basis Function Network much later.

### 5.1.3 Learning of Unknown Parameters

All experiments of sections 5.1.1 and 5.1.2 have been done with neural networks designed on the basis of complete knowledge of the Lorenz differential equation generating the time series. This includes the parameters A, B and C, whose values determine the characteristic behavior of the Lorenz system. In section 4.8.2, we have suggested a procedure to learn the parameters from observation data, in case this knowledge is not available:

1. Design a neural network approximating the differential equation with the method presented in section 4.7. The unknown parameters of the differential equation become (unknown) weights of the resulting neural network.

2. Construct a forecasting error function E that measures the deviation of the neural network forecasts from a time series of observations. A standard error function E is given in (5-8) (see section 4.8.1):

$$
\begin{aligned}
E &= \frac{1}{2}\sum_{i=1}^{N} \left\| \bar{x}_i - NN_i \right\|_2^2 \\
\bar{x}_i &= \bar{x}(t_0 + i \cdot \Delta t) = \text{ time series of observations} \\
\Delta t &= \text{ time increment between observations} \\
N &= \text{ length of time series} \\
NN_i &= \text{ neural network forecast for observation i}
\end{aligned}
\tag{5-8}
$$

3. Minimize error E with a neural network training algorithm that finds the best values for the unknown network weights (the unknown parameters of the differential equation).

The parameter values estimated in step 3 optimize the mean-squared forecasting error of the neural network, but they are not necessarily identical to the real parameters of the differential equation. This problem occurs if there exists an alternative set of parameters, which generates better forecasts than the correct parameters do. In such cases the estimated parameters of the differential equation will not be accurate.

We use an adaptive simulated annealing algorithm[6] (see [Ingb89] or [Ingb93]) to find the unknown parameters A, B and C of the differential equation. The training data consists of 2000 samples generated by our reference model with parameter values A=10, B=28, C=8/3, $t_0=0$, $x_{1,0}=x_{2,0}=x_{3,0}=1$, $\Delta t=0.001$. Figure 5.15 shows the convergence of error E for one-step-ahead forecasts of the Radial Basis Function Network, and figure 5.16 illustrates the convergence of the corresponding parameters to the final approximations A=9.745, B=28.335 and C=2.589 after 4133 steps[7].

---

6 This software is available from Lester Ingber by anonymous ftp on the Internet at ftp.alumni.caltech.edu:/pub/ingber/ASA.tar.gz.

7 This number is equivalent to the number of error function evaluations performed by the simulated annealing algorithm.

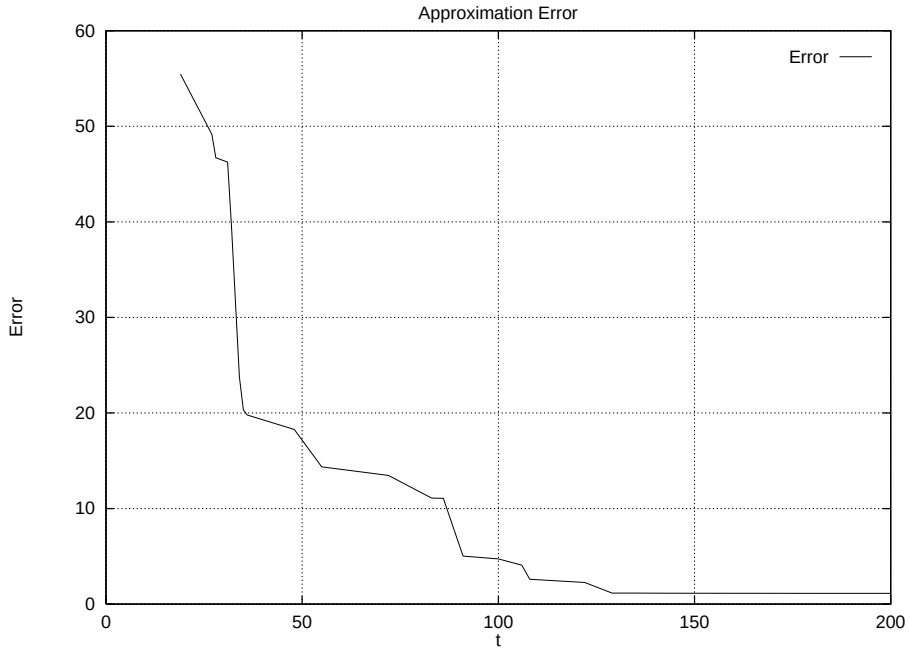



Fig. 5.15 : Forecasting error of Radial Basis Function Network

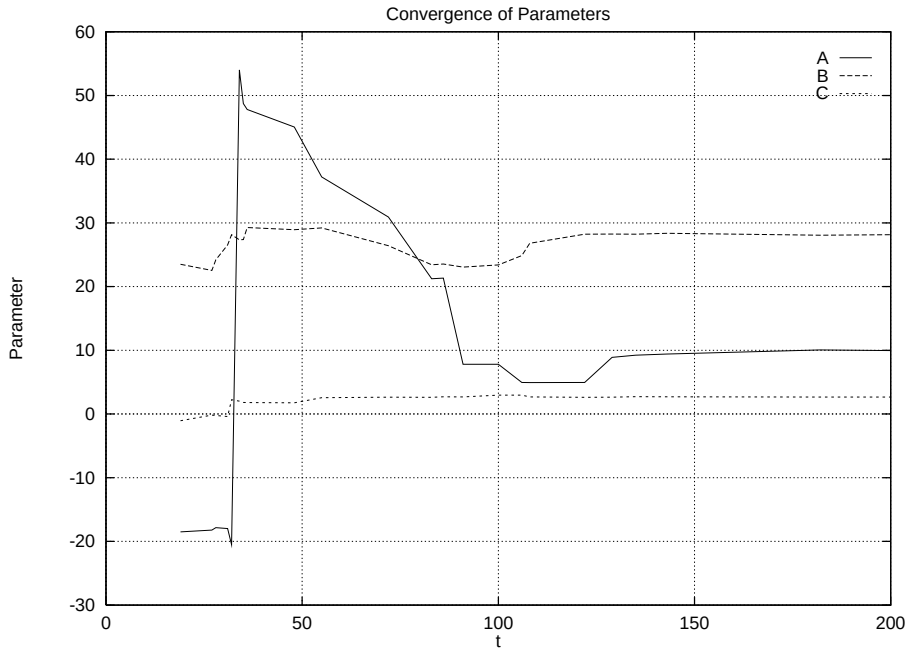



Fig. 5.16 : Convergence of learned parameters

Note that the error function E based on one-step-ahead forecasts produces more reliable estimations of the parameter values than the one using repeated forecasts. This is a consequence of the higher sensitivity of repeated forecasts to errors caused by neural networks with insufficient number of parameters $a_k$.

Although the final forecasting error of the Radial Basis Function Network is rather small, the estimated parameters are not very close to the correct values A=10, B=28, C=8/3. The simulated annealing algorithm asymptotically finds the global optimum, which does not coincide exactly with the correct parameter values. In section 5.1.2, Modified Logistic Networks performed better for the Lorenz system than Radial Basis Function Networks. Therefore, we repeat this experiment with Modified Logistic Networks and we expect to find better approximations of the parameters A, B, C.

Figures 5.17 and 5.18 show the convergence of the forecasting error E and the parameters A, B and C for a Modified Logistic Network. The final parameter values A=10.002, B=27.982, C=2.669 have been reached after 4056 steps[8].

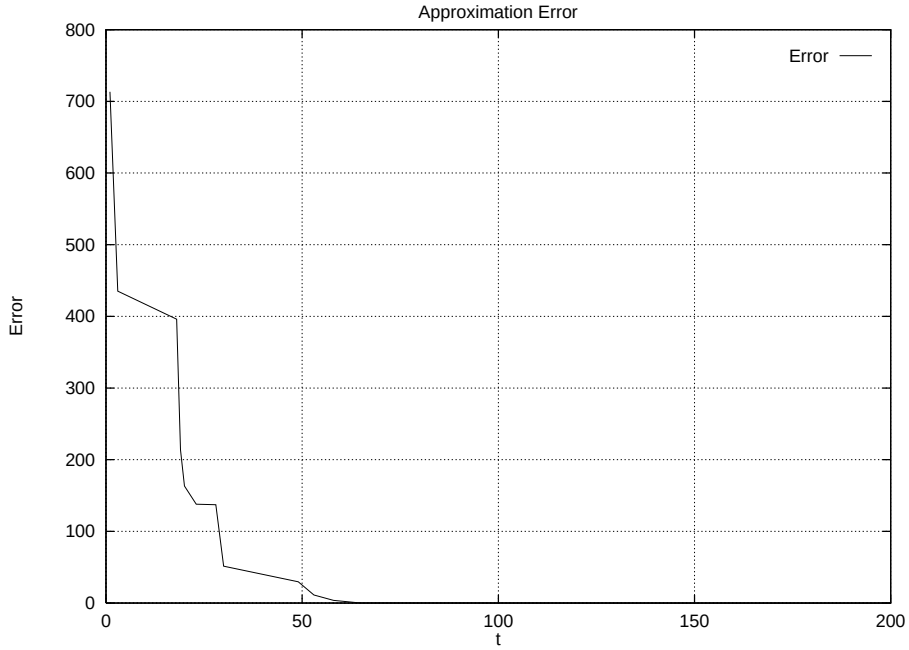



Fig. 5.17 : Forecasting error of Modified Logistic Network

The final parameter values estimated by training the Modified Logistic Network are better approximations of the correct parameters than those estimated by the Radial Basis Function Network. This confirms our assumption that minimizing the forecasting error does not necessarily lead

[8] The exact number of steps executed by the simulated annealing algorithm is determined by several termination conditions. Therefore, the number of steps may be different from experiment to experiment.

to the exact parameter values if the neural network makes weak forecasts. This problem can be solved by using more training data, by decreasing the integration step size or by increasing the accuracy of the neural network.

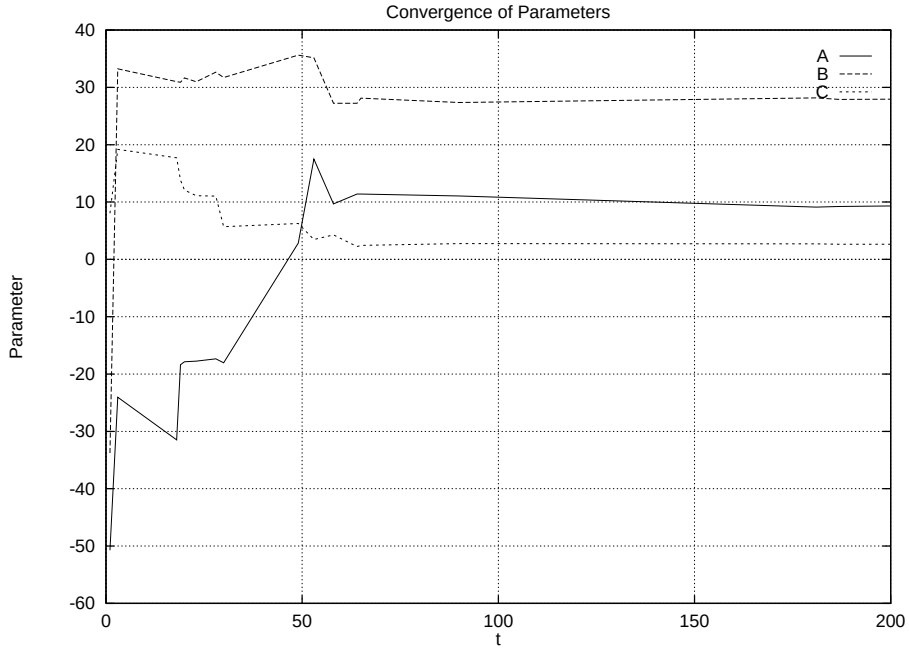



Fig. 5.18 : Convergence of learned parameters

## 5.2 Rössler System

The Rössler system (5-9) of differential equations is described in [Roes76] and [Pack80]. It is actually a simplification of the Lorenz model of turbulence in fluids. It has similar characteristic behavior, but it lacks an immediate physical interpretation.

$$\begin{aligned} x'(t) &= -y(t) - z(t) \\ y'(t) &= x(t) + A \cdot y(t) \\ z'(t) &= B + x(t) \cdot z(t) - C \cdot z(t) \\ x(t_0) &= x_0 \\ y(t_0) &= y_0 \\ z(t_0) &= z_0 \end{aligned} \tag{5-9}$$

We use the notation (5-10), which embeds the Rössler system into our framework:

$$\begin{aligned} x_1'(t) &= -x_2(t) - x_3(t) \\ x_2'(t) &= x_1(t) + A \cdot x_2(t) \\ x_3'(t) &= B + x_1(t) \cdot x_3(t) - C \cdot x_3(t) \\ x_1(t_0) &= x_{1,0} \\ x_2(t_0) &= x_{2,0} \\ x_3(t_0) &= x_{3,0} \end{aligned} \tag{5-10}$$

Figure 5.19 illustrates the evolution of the Rössler system for the critical parameters A=0.2, B=0.4 C=5.7 listed in [Pack80], where we start with initial conditions $t_0$=0, $x_{1,0}$=$x_{2,0}$=$x_{3,0}$=10.

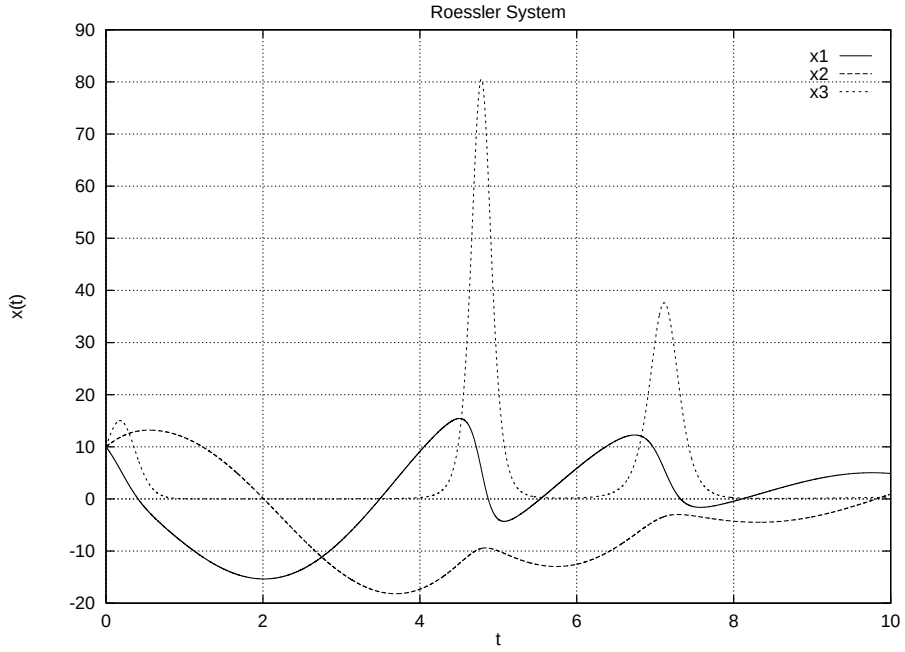



Fig. 5.19 : Evolution of the Rössler system

Again, a neural network for the approximation of the Rössler system is constructed with the algorithm described in section 4.7:

1. Since the differential equations are already in polynomial form, no transformation is necessary.

2. For each variable $x_1(t)..x_3(t)$, we have to choose an approximation function $\Phi_K^i$. We select the Radial Basis Function Networks (5-11) with five parameters $a_{i,0}..a_{i,4}$ for that purpose (see section 4.4.2).

$$x_i(t) \approx \Phi_4^{RBF_i}(t;t_0;a_{i,0},\ldots,a_{i,4}), i = 1..3 \tag{5-11}$$

3. We substitute the approximation functions $\Phi_4^{RBF_i}$ for $x_i(t)$ in the equations generated by successive differentiation, and we get an equation system for the parameters $a_{i,k}$ (omitted).

4. The equation system is solved for the parameters $a_{i,k}$, resulting in the values (5-12.1) to (5-14.5).

$$a_{1,0} = x_{1,0} \tag{5-12.1}$$

$$a_{1,1} = \left(x_{2,0} + x_{3,0}\right) \cdot e \tag{5-12.2}$$

$$a_{1,2} = -\left(B + x_{1,0} + x_{2,0} \cdot A - x_{3,0} \cdot C + x_{1,0} \cdot x_{3,0}\right) \cdot \frac{e^2}{4} \tag{5-12.3}$$

$$a_{1,3} = \left( \begin{array}{l} 2 \cdot A^2 - 3 \cdot B - 2 \cdot B \cdot C + x_{1,0} \cdot (2 \cdot A + 2 \cdot B - 3) - \\ x_{2,0} \cdot (3 \cdot A + 2) + x_{3,0} \cdot (3 \cdot C + 2 \cdot C^2 - 2) - \\ x_{1,0} \cdot x_{3,0} \cdot (4 \cdot C + 3) - x_{2,0} \cdot x_{3,0} \cdot 2 + \\ {x_{1,0}}^2 \cdot x_{3,0} \cdot 2 - {x_{3,0}}^2 \cdot 2 \end{array} \right) \cdot \frac{e^3}{48} \tag{5-12.4}$$

$$a_{1,4} = -\left( \begin{array}{l} B \cdot (9 + 4 \cdot C + C^2) + x_{1,0} \cdot (9 - 4 \cdot A - 4 \cdot B - 2 \cdot B \cdot C + A^2) + \\ x_{2,0} \cdot (4 + 8 \cdot A - 2 \cdot B - 4 \cdot A^2 + A^3) + \\ x_{3,0} \cdot (4 - A - 3 \cdot B - 9 \cdot C - 4 \cdot C^2 - C^3) + \\ x_{1,0} \cdot x_{3,0} \cdot (8 + 8 \cdot C + 3 \cdot C^2) + x_{2,0} \cdot x_{3,0} \cdot (4 - A + 3 \cdot C) - \\ x_{1,0} \cdot x_{2,0} \cdot x_{3,0} \cdot 3 + {x_{1,0}}^2 \cdot B + {x_{3,0}}^2 \cdot (4 + 4 \cdot C) - \\ {x_{1,0}}^2 \cdot x_{3,0} \cdot (4 + 3 \cdot C) - x_{1,0} \cdot {x_{3,0}}^2 \cdot 4 + {x_{1,0}}^3 \cdot x_{3,0} \end{array} \right) \cdot \frac{e^4}{192} \tag{5-12.5}$$

$$a_{2,0} = x_{2,0} \tag{5-13.1}$$

$$a_{2,1} = -\left(x_{1,0} + x_{2,0} \cdot A\right) \cdot e \tag{5-13.2}$$

$$a_{2,2} = \left(x_{1,0} \cdot A + x_{2,0} \cdot (A^2 - 1) - x_{3,0}\right) \cdot \frac{e^2}{4} \tag{5-13.3}$$

$$a_{2,3} = -\left( \begin{array}{l} -2 \cdot B + x_{1,0} \cdot (A \cdot (2 \cdot A - 3) - 2) + \\ x_{2,0} \cdot (3 + A \cdot (2 \cdot A^2 - 3 \cdot A - 4)) + \\ x_{3,0} \cdot (3 - 2 \cdot A + 2 \cdot C) - x_{1,0} \cdot x_{3,0} \cdot 2 \end{array} \right) \cdot \frac{e^3}{48} \tag{5-13.4}$$

$$a_{2,4} = \begin{pmatrix} B\cdot(4-A+C)+x_{1,0}\cdot(4+A\cdot(8-4\cdot A+A^2)-B)+ \\ x_{2,0}\cdot(A\cdot(8+A\cdot(7-4\cdot A+A^2))-9)+ \\ x_{3,0}\cdot(A\cdot(4-A+C)-9-4\cdot C-C^2)+ \\ x_{1,0}\cdot x_{3,0}\cdot(4-A+2\cdot C)+x_{2,0}\cdot x_{3,0}-{x_{1,0}}^2\cdot x_{3,0}+{x_{3,0}}^2 \end{pmatrix}\cdot\frac{e^4}{192} \qquad (5\text{-}13.5)$$

$$a_{3,0} = x_{3,0} \qquad (5\text{-}14.1)$$

$$a_{3,1} = -\left(B - x_{3,0}\cdot C + x_{1,0}\cdot x_{3,0}\right)\cdot e \qquad (5\text{-}14.2)$$

$$a_{3,2} = -\begin{pmatrix} B\cdot C - x_{1,0}\cdot B - x_{3,0}\cdot C^2 + x_{1,0}\cdot x_{3,0}\cdot 2\cdot C+ \\ x_{2,0}\cdot x_{3,0} + {x_{3,0}}^2 - {x_{1,0}}^2\cdot x_{3,0} \end{pmatrix}\cdot\frac{e^2}{4} \qquad (5\text{-}14.3)$$

$$a_{3,3} = \begin{pmatrix} -B\cdot C\cdot(3+2\cdot C)+x_{1,0}\cdot(B\cdot(3+4\cdot C))+ \\ x_{2,0}\cdot 4\cdot B+x_{3,0}\cdot(6\cdot B+C^2\cdot(3+2\cdot C))+ \\ x_{1,0}\cdot x_{3,0}\cdot(2-C\cdot(6+6\cdot C))+x_{2,0}\cdot x_{3,0}\cdot(2\cdot A-3-6\cdot C)+ \\ x_{1,0}\cdot x_{2,0}\cdot x_{3,0}\cdot 6-{x_{1,0}}^2\cdot 2\cdot B-{x_{3,0}}^2\cdot(3+8\cdot C)+ \\ {x_{1,0}}^2\cdot x_{3,0}\cdot(3+6\cdot C)+x_{1,0}\cdot{x_{3,0}}^2\cdot 8-{x_{1,0}}^3\cdot x_{3,0}\cdot 2 \end{pmatrix}\cdot\frac{e^3}{48} \qquad (5\text{-}14.4)$$

$$a_{3,4} = -\begin{pmatrix} B\cdot(3\cdot B+C\cdot(10+4\cdot C+C^2))- \\ x_{1,0}\cdot(B\cdot(7+8\cdot C+3\cdot C^2))+x_{2,0}\cdot(B\cdot(3\cdot A-8-5\cdot C))- \\ x_{3,0}\cdot(12\cdot B+C\cdot(13\cdot B+C\cdot(10+4\cdot C+C^2)))+ \\ x_{1,0}\cdot x_{3,0}\cdot(A-4+13\cdot B+C\cdot(16+12\cdot C+4\cdot C^2))+ \\ x_{2,0}\cdot x_{3,0}\cdot(9-4\cdot A+A^2+C\cdot(12-4\cdot A+6\cdot C))+ \\ x_{1,0}\cdot x_{2,0}\cdot 5\cdot B+x_{1,0}\cdot x_{2,0}\cdot x_{3,0}\cdot(4\cdot A-12-12\cdot C)+ \\ {x_{1,0}}^2(B\cdot(4+3\cdot C))+{x_{3,0}}^2(9+C\cdot(16+11\cdot C))- \\ {x_{1,0}}^2\cdot x_{3,0}\cdot(6+C\cdot(12+6\cdot C))-x_{1,0}\cdot{x_{3,0}}^2\cdot(16+22\cdot C)+ \\ -{x_{2,0}}^2\cdot x_{3,0}\cdot 3+-x_{2,0}\cdot{x_{3,0}}^2\cdot 7+{x_{2,0}}^2\cdot x_{3,0}\cdot 4\cdot A^2+ \\ {x_{1,0}}^2\cdot x_{2,0}\cdot x_{3,0}\cdot 6+{x_{1,0}}^2\cdot{x_{3,0}}^2\cdot 11- \\ {x_{1,0}}^3\cdot B-{x_{3,0}}^3\cdot 4+{x_{1,0}}^3\cdot x_{3,0}\cdot(4+4\cdot C)-{x_{1,0}}^4\cdot x_{3,0} \end{pmatrix}\cdot\frac{e^4}{192} \qquad (5\text{-}14.5)$$

5. The single-step integration procedure (5-15) is completed by inserting the resolved parameters $a_{i,k}$ into $\Phi_4^{RBF_i}$.

$$x_i(t) \approx \Phi_4^{RBF_i}(t_0+\Delta t; t_0; a_{i,0},\ldots,a_{i,4}) = \Phi_i^{RBF}(\Delta t; x_{1,0}, x_{2,0}, x_{3,0}; A, B, C) \qquad (5\text{-}15)$$

### 5.2.1 One-Step-Ahead Forecasts

Figure 5.20 compares the one-step-ahead forecasts of the Radial Basis Function Network with a reference model generated by an adaptive Runge-Kutta method. The results are based on the parameters A=0.2, B=0.4, C=5.7, the initial conditions $t_0$=0, $x_{1,0}$=$x_{2,0}$=$x_{3,0}$=10 and the step size $\Delta t$=0.001:

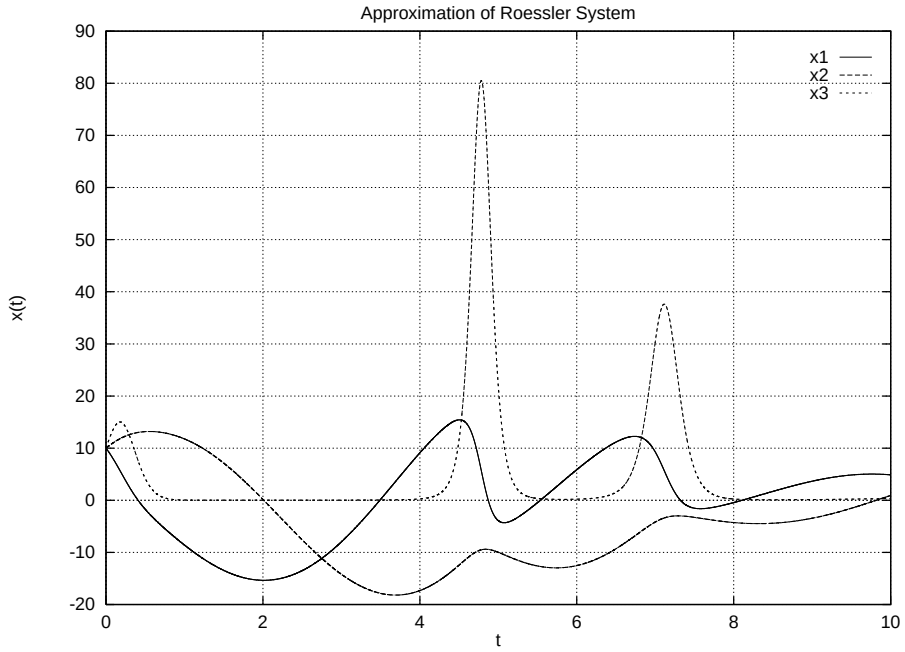



Fig. 5.20 : One-step-ahead forecasts and reference model

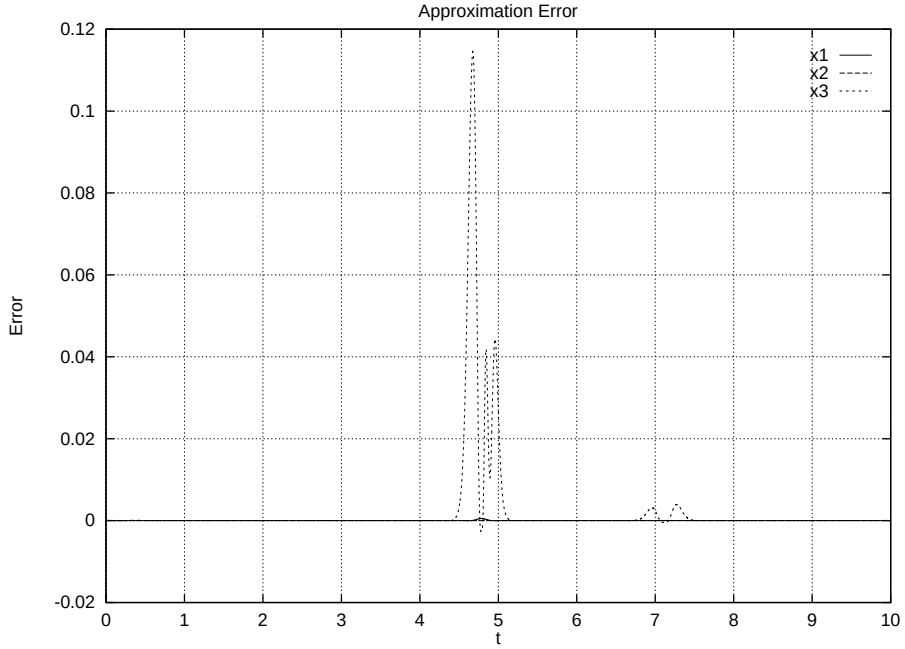



Fig. 5.21 : Approximation error of one-step-ahead forecasts

The Radial Basis Function Network generates high-quality one-step-ahead forecasts of the Rössler system, which is shown in figure 5.21.

### 5.2.2 Repeated Forecasts

The repeated forecasts and the corresponding approximation errors of the Radial Basis Function Network are displayed in figures 5.22 and 5.23:

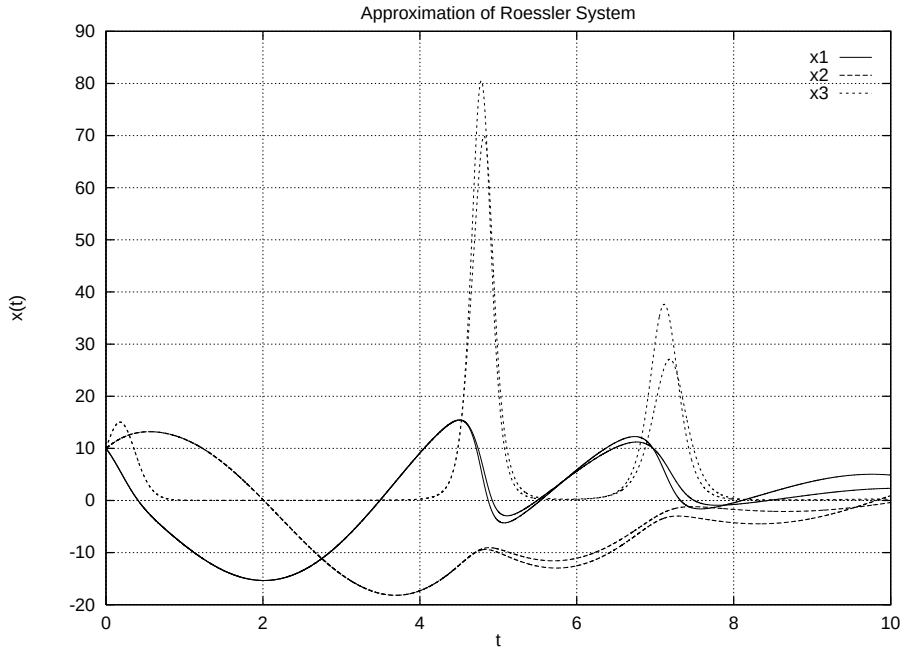



Fig. 5.22 : Repeated forecasts and reference model

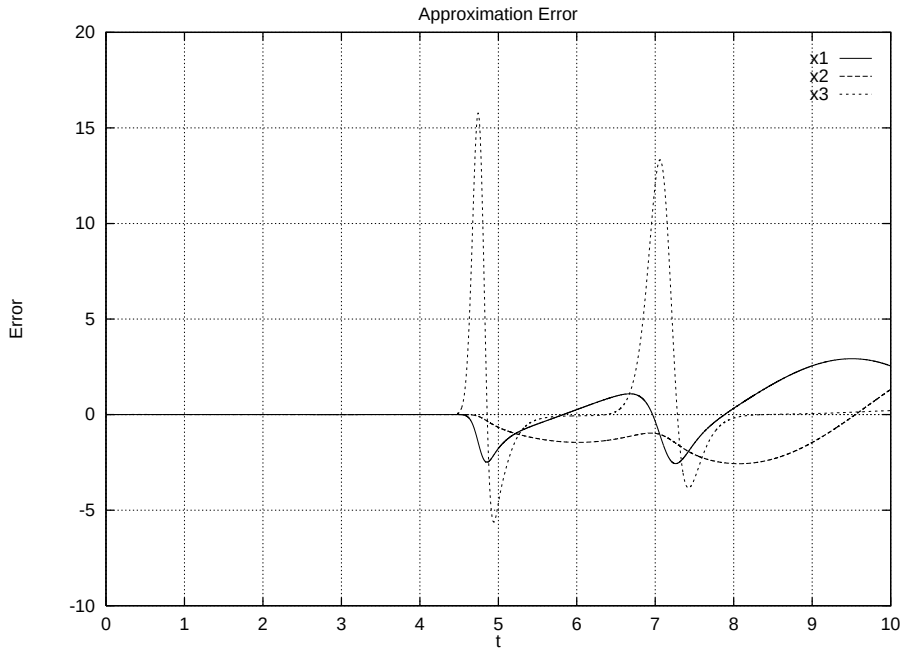



Fig. 5.23 : Approximation error of repeated forecasts

Again, the Rössler system confirms the difficulty of long-term forecasting of chaotic differential equations: After a period of about four seconds, where the forecasting quality of the Radial Basis Function Network is impressive, the approximation errors explode at the first sharp peak of $x_3(t)$ of the Rössler system. Nonetheless, the forecasts of the Rössler system are better than those of the Lorenz system.

### 5.2.3 Learning of Unknown Parameters

The parameters A, B and C of the Rössler differential equation are again estimated with the simulated annealing algorithm and the error function presented in section 5.1.3. The evolution of the forecasting error is illustrated in figure 5.24 and the convergence of the parameters is displayed in figure 5.25.

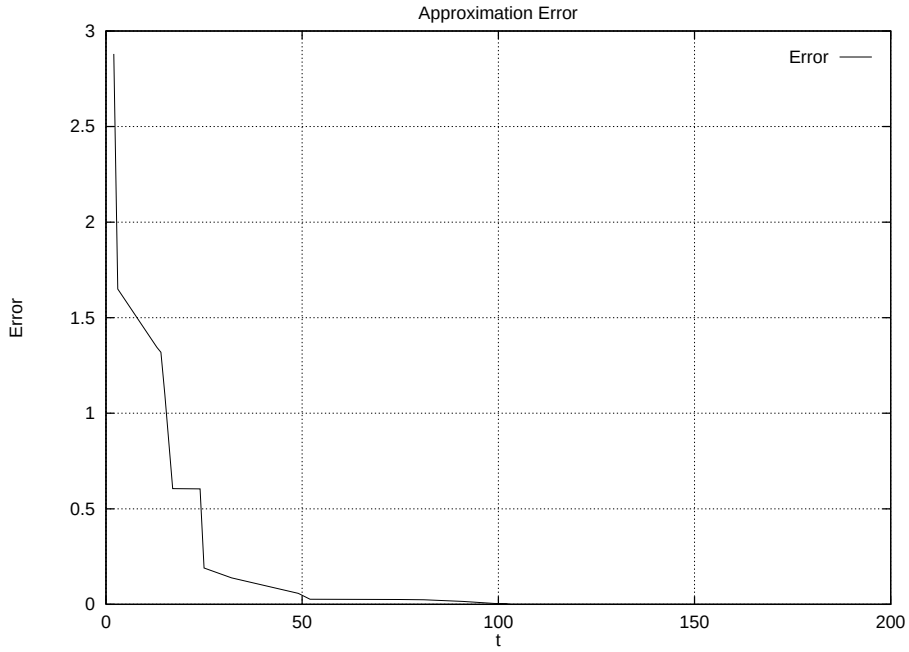



Fig. 5.24 : Forecasting error during parameter estimation

After 3587 steps, the final parameter values estimated by the simulated annealing algorithm are A=0.200, B=0.400, C=5.700 (four correct digits), which are very good approximations to the correct values. Since the Radial Basis Function Network approximates the Rössler system better than the Lorenz system, the global minimum of the forecasting error coincides with the optimal parameters. This allows the simulated annealing algorithm to converge to the correct parameter estimations.

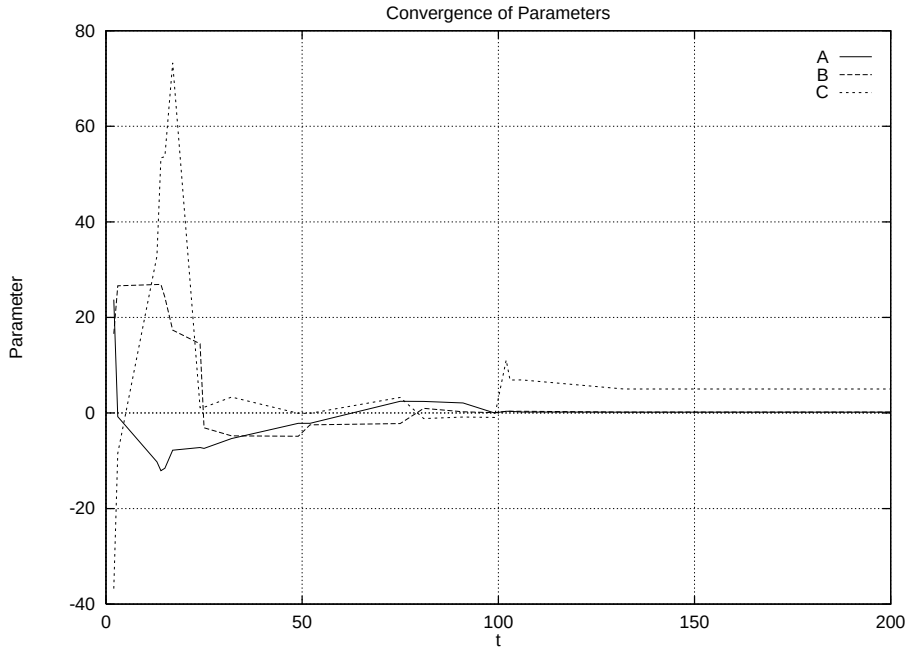



Fig. 5.25 : Convergence of learned parameters

## 5.3 Circular Pendulum

Finally, we investigate the circular pendulum system shown in figure 5.26:

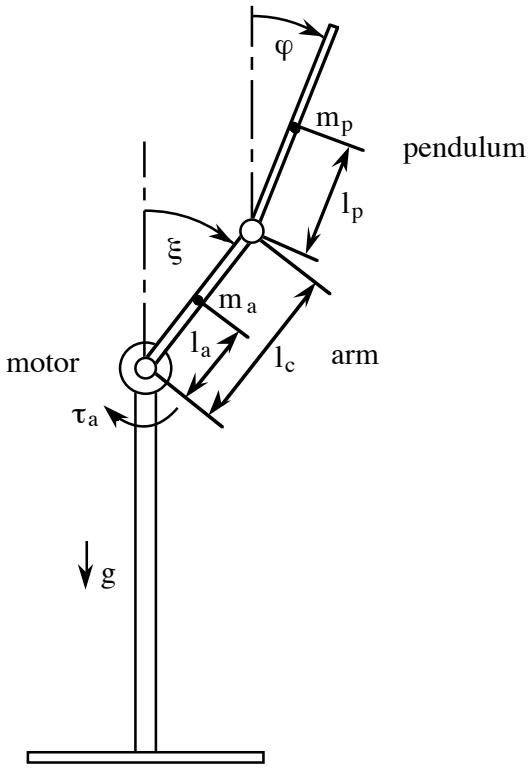



Fig. 5.26 : Circular pendulum

The circular pendulum consists of a pendulum attached to an arm. The pendulum is connected to a motor and the arm can move freely. This circular pendulum was actually built at the ABB Corporate Research Laboratory in Baden, Switzerland. A full model of the pendulum including friction and motor characteristics is described in [Sche94], which is based on [Rott70] .

We restrict the general model of the circular pendulum to a version without friction, damping and external forces. The resulting motion equations are listed in (5-16). The abbreviations and the normalized parameters contained in (5-16) are described in (5-17), (5-18) and (5-19).

$$\xi''(t) = \frac{p(\bar{x}) - A \cdot \cos(\varphi(t) - \xi(t)) \cdot q(\bar{x})}{r(\bar{x})}$$
$$\varphi''(t) = \frac{q(\bar{x}) - C \cdot \cos(\varphi(t) - \xi(t)) \cdot p(\bar{x})}{r(\bar{x})} \qquad (5\text{-}16)$$

$$\bar{x} = (\xi(t), \xi'(t), \varphi(t), \varphi'(t))$$
$$p(\bar{x}) = A \cdot \sin(\varphi(t) - \xi(t)) \cdot \varphi'(t)^2 + B \cdot \sin(\xi(t))$$
$$q(\bar{x}) = -C \cdot \sin(\varphi(t) - \xi(t)) \cdot \xi'(t)^2 + D \cdot \sin(\varphi(t)) \qquad (5\text{-}17)$$
$$r(\bar{x}) = 1 - A \cdot C \cdot \cos(\varphi(t) - \xi(t))^2$$

$$A = \frac{J_c}{J_t}, B = \frac{g_t}{J_t}, C = \frac{J_c}{J_p}, D = \frac{g_p}{J_p} \qquad (5\text{-}18)$$

$$J_t = J_a + l_c^2 \cdot m_p + n_g^2 \cdot (J_m + J_g)$$
$$J_c = l_c \cdot l_p \cdot m_p$$
$$g_t = g \cdot (l_a \cdot m_a + l_c \cdot m_p) \qquad (5\text{-}19)$$
$$g_p = g \cdot l_p \cdot m_p$$

The physical parameters of the actual circular pendulum are listed in table 5.1.

| Parameter | Value | Unit | Description |
|---|---|---|---|
| $m_a$ | 0.44 | kg | Mass of arm |
| $l_a$ | 0.4 | m | Distance of center of arm's mass to its axis |
| $l_c$ | 0.5 | m | Length of arm |
| $J_a$ | 0.08 | kg $m^2$ | Arm's moment of mass inertia |
| $m_p$ | 0.1 | kg | Mass of pendulum |
| $l_p$ | 0.37 | m | Distance of center of pendulum's mass to its axis |
| $J_p$ | 0.019 | kg $m^2$ | Pendulum's moment of mass inertia |
| $n_g$ | 16 | - | Gear ratio |
| $J_g$ | $1.9\ 10^{-5}$ | kg $m^2$ | Gear's moment of mass inertia |
| $J_m$ | $4.3\ 10^{-4}$ | kg $m^2$ | Motor's moment of mass inertia |
| g | 9.81 | $m/s^2$ | Acceleration constant |

Table 5.1 : Physical parameters of the pendulum

The values of the parameters A, B, C, D are listed in table 5.2:

| Parameter | Value |
|---|---|
| A | 0.0841 |
| B | 10.1 |
| C | 0.974 |
| D | 19.1 |

Table 5.2 : Parameters of the circular pendulum system

The circular pendulum system described by (5-16) and (5-17) is transformed into the first order differential equations (5-20).

$$
\begin{aligned}
& x_1(t) = \xi(t), x_2(t) = \xi'(t), x_3(t) = \varphi(t), x_4(t) = \varphi'(t) \\
& x'(t)_1 = x_2(t) \\
& x_2'(t) = \frac{\begin{bmatrix} A \cdot \sin(x_3(t) - x_1(t)) \cdot x_4(t)^2 + B \cdot \sin(x_1(t)) \\ -A \cdot \cos(x_3(t) - x_1(t)) \cdot \\ (-C \cdot \sin(x_3(t) - x_1(t)) \cdot x_2(t)^2 + D \cdot \sin(x_3(t))) \end{bmatrix}}{1 - A \cdot C \cdot \cos(x_3(t) - x_1(t))^2} \\
& x_3'(t) = x_4(t) \\
& x_4'(t) = \frac{\begin{bmatrix} -C \cdot \sin(x_3(t) - x_1(t)) \cdot x_2(t)^2 + D \cdot \sin(x_3(t)) \\ -C \cdot \cos(x_3(t) - x_1(t)) \cdot \\ (A \cdot \sin(x_3(t) - x_1(t)) \cdot x_4(t)^2 + B \cdot \sin(x_1(t))) \end{bmatrix}}{1 - A \cdot C \cdot \cos(x_3(t) - x_1(t))^2}
\end{aligned}
\qquad (5\text{-}20)
$$

The following figure 5.27 illustrates the complex motion of the circular pendulum starting with initial conditions $t_0$=0, $x_{1,0}$=1.0, $x_{2,0}$=0, $x_{3,0}$=2.0, $x_{4,0}$=0:

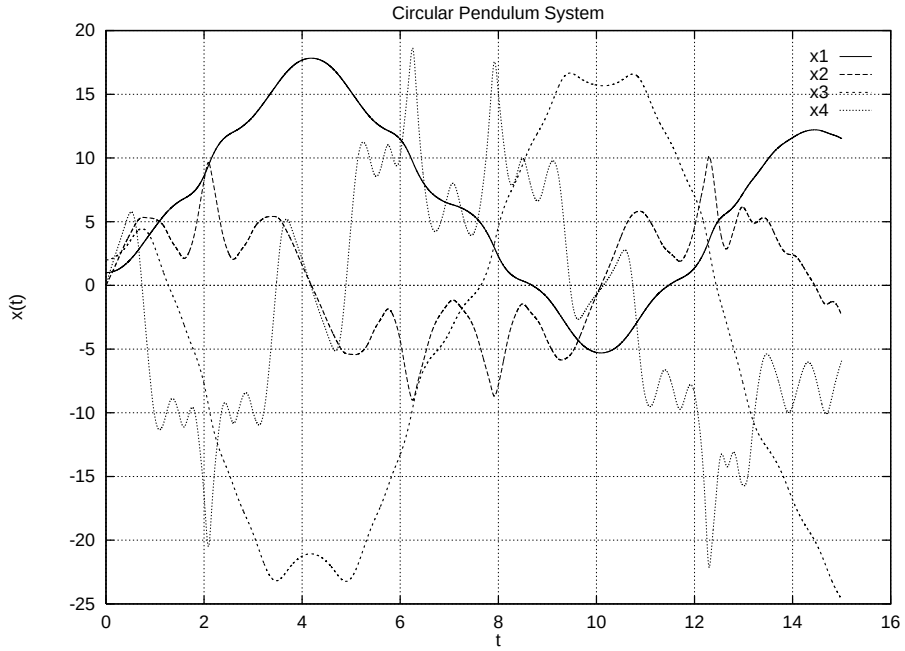



Fig. 5.27 : Evolution of the circular pendulum system

We now develop a neural network for the approximation of the circular pendulum system with the design algorithm presented in section 4.7:

1. The differential equations (5-20) have to be transformed into polynomial form. We use the substitutions (5-21), the trigonometric identities (5-22) and the transformation rules of table 4.5 in section 4.5 to derive the transformed circular pendulum system (5-23).

$$\begin{aligned}
&x_1(t) = \xi(t), x_2(t) = \xi'(t), x_3(t) = \varphi(t), x_4(t) = \varphi'(t)\\
&x_5(t) = \sin(x_1(t)), x_6(t) = \cos(x_1(t))\\
&x_7(t) = \sin(x_3(t)), x_8(t) = \cos(x_3(t))\\
&x_9(t) = \frac{1}{1 - A \cdot C \cdot \cos(x_3(t) - x_1(t))^2}\\
&\qquad = \frac{1}{1 - A \cdot C \cdot (x_5(t) \cdot x_7(t) + x_6(t) \cdot x_8(t))^2}
\end{aligned} \tag{5-21}$$

$$\begin{aligned}
&\sin(\alpha - \beta) = \sin(\alpha) \cdot \cos(\beta) - \cos(\alpha) \cdot \sin(\beta)\\
&\cos(\alpha - \beta) = \cos(\alpha) \cdot \cos(\beta) + \sin(\alpha) \cdot \sin(\beta)
\end{aligned} \tag{5-22}$$

$$\begin{aligned}
x_1'(t) &= x_2(t)\\
x_2'(t) &= \begin{bmatrix} A \cdot (x_6(t) \cdot x_7(t) - x_5(t) \cdot x_8(t)) \cdot x_4(t)^2 + B \cdot x_5(t) + \\ A \cdot (x_5(t) \cdot x_7(t) + x_6(t) \cdot x_8(t)) \cdot \\ (C \cdot (x_6(t) \cdot x_7(t) - x_5(t) \cdot x_8(t)) \cdot x_2(t)^2 + D \cdot x_7(t)) \end{bmatrix} \cdot x_9(t)\\
x_3'(t) &= x_4(t)\\
x_4'(t) &= \begin{bmatrix} -C \cdot (x_6(t) \cdot x_7(t) - x_5(t) \cdot x_8(t)) \cdot x_2(t)^2 + D \cdot x_7(t) - \\ C \cdot (x_5(t) \cdot x_7(t) + x_6(t) \cdot x_8(t)) \cdot \\ (A \cdot (x_6(t) \cdot x_7(t) - x_5(t) \cdot x_8(t)) \cdot x_4(t)^2 + B \cdot x_5(t)) \end{bmatrix} \cdot x_9(t)\\
x_5'(t) &= x_2(t) \cdot x_6(t)\\
x_6'(t) &= -x_2(t) \cdot x_5(t)\\
x_7'(t) &= x_4(t) \cdot x_8(t)\\
x_8'(t) &= -x_4(t) \cdot x_7(t)\\
x_9'(t) &= 2 \cdot A \cdot C \cdot x_9(t) \cdot (x_5(t) \cdot x_7(t) + x_6(t) \cdot x_8(t)) \cdot \\
&\qquad \begin{bmatrix} x_4(t) \cdot x_5(t) \cdot x_8(t) + x_2(t) \cdot x_6(t) \cdot x_7(t) \\ -x_4(t) \cdot x_6(t) \cdot x_7(t) - x_2(t) \cdot x_5(t) \cdot x_8(t) \end{bmatrix}
\end{aligned} \tag{5-23}$$

2. For each variable $x_1(t)..x_9(t)$, we have to choose an approximation function $\Phi_K^i$. This time, we select Modified Logistic Networks (5-24) with five parameters $a_{i,0}..a_{i,4}$ for that purpose (see section 4.4.1), because Modified Logistic Networks proved to be superior than Radial Basis Function Networks for both the Lorenz and the Rössler system.

$$x_i(t) \approx \Phi_4^{MLG_i}(t;t_0;a_{i,0},...,a_{i,4}), i = 1..9 \tag{5-24}$$

3. We substitute the approximation functions $\Phi_4^{MLG_i}$ for $x_i(t)$ in the equations generated by successive differentiation, and we get an equation system for the parameters $a_{i,k}$ (omitted).

4. The equation system is solved for the parameters $a_{i,k}$, resulting in the values (5-25.1) to (5-33.2). Because of the prohibitive complexity, we list only a subset of the parameters $a_{i,k}$ actually computed.

$$a_{1,0} = x_{1,0} \tag{5-25.1}$$

$$a_{1,1} = x_{2,0} \tag{5-25.2}$$

$$a_{1,2} = -\frac{x_{9,0}}{2} \cdot \begin{pmatrix} ((x_{5,0} \cdot x_{8,0} - x_{6,0} \cdot x_{7,0}) \cdot {x_{4,0}}^2 - \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{7,0} \cdot D + \\ (({x_{8,0}}^2 - {x_{7,0}}^2) \cdot x_{6,0} + x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} - \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{2,0}}^2 \cdot C) \cdot A - B \cdot x_{5,0} \end{pmatrix} \tag{5-25.3}$$

etc.

$$a_{2,0} = x_{2,0} \tag{5-26.1}$$

$$a_{2,1} = -x_{9,0} \cdot \begin{pmatrix} ((x_{5,0} \cdot x_{8,0} - x_{6,0} \cdot x_{7,0}) \cdot {x_{4,0}}^2 - \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{7,0} \cdot D + \\ (({x_{8,0}}^2 - {x_{7,0}}^2) \cdot x_{6,0} + x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} - \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{2,0}}^2 \cdot C) \cdot A - B \cdot x_{5,0} \end{pmatrix} \tag{5-26.2}$$

etc.

$$a_{3,0} = x_{3,0} \tag{5-27.1}$$

$$a_{3,1} = x_{4,0} \tag{5-27.2}$$

$$a_{3,2} = \frac{x_{9,0}}{2} \cdot \begin{pmatrix} (x_{5,0} \cdot x_{8,0} - x_{6,0} \cdot x_{7,0}) \cdot {x_{2,0}}^2 \cdot C - \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{5,0} \cdot B \cdot C + \\ (({x_{8,0}}^2 - {x_{7,0}}^2) \cdot x_{6,0} + x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} - \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{4,0}}^2 \cdot A \cdot C + D \cdot x_{7,0} \end{pmatrix} \tag{5-27.3}$$

etc.

$$a_{4,0} = x_{4,0} \tag{5-28.1}$$

$$a_{4,1} = x_{9,0} \cdot \begin{pmatrix} (x_{5,0} \cdot x_{8,0} - x_{6,0} \cdot x_{7,0}) \cdot {x_{2,0}}^2 \cdot C - \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{5,0} \cdot B \cdot C + \\ (({x_{8,0}}^2 - {x_{7,0}}^2) \cdot x_{6,0} + x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} - \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{4,0}}^2 \cdot A \cdot C + D \cdot x_{7,0} \end{pmatrix} \tag{5-28.2}$$

etc.

$$a_{5,0} = x_{5,0} \tag{5-29.1}$$

$$a_{5,1} = x_{2,0} \cdot x_{6,0} \tag{5-29.2}$$

$$a_{5,2} = \frac{x_{6,0} \cdot x_{9,0}}{2} \cdot \begin{pmatrix} (({x_{6,0}} \cdot x_{7,0} - x_{5,0} \cdot x_{8,0}) \cdot {x_{4,0}}^2 + \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{7,0} \cdot D + \\ (({x_{7,0}}^2 - {x_{8,0}}^2) \cdot x_{6,0} - x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} + \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{2,0}}^2 \cdot C) \cdot A + B \cdot x_{5,0} \end{pmatrix} - \frac{{x_{2,0}}^2 \cdot x_{5,0}}{2} \tag{5-29.3}$$

etc.

$$a_{6,0} = x_{6,0} \tag{5-30.1}$$

$$a_{6,1} = -x_{2,0} \cdot x_{5,0} \tag{5-30.2}$$

$$a_{6,2} = \frac{x_{5,0} \cdot x_{9,0}}{2} \cdot \begin{pmatrix} (({x_{5,0}} \cdot x_{8,0} - x_{6,0} \cdot x_{7,0}) \cdot {x_{4,0}}^2 - \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{7,0} \cdot D + \\ (({x_{8,0}}^2 - {x_{7,0}}^2) \cdot x_{6,0} + x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} - \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{2,0}}^2 \cdot C) \cdot A - B \cdot x_{5,0} \end{pmatrix} - \frac{{x_{2,0}}^2 \cdot x_{6,0}}{2} \tag{5-30.3}$$

etc.

$$a_{7,0} = x_{7,0} \tag{5-31.1}$$

$$a_{7,1} = x_{4,0} \cdot x_{8,0} \tag{5-31.2}$$

$$a_{7,2} = \frac{x_{8,0} \cdot x_{9,0}}{2} \cdot \begin{pmatrix} ((x_{5,0} \cdot x_{8,0} - x_{6,0} \cdot x_{7,0}) \cdot {x_{2,0}}^2 - \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{5,0} \cdot B + \\ (({x_{8,0}}^2 - {x_{7,0}}^2) \cdot x_{6,0} + x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} - \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{4,0}}^2 \cdot A) \cdot C - D \cdot x_{7,0} \end{pmatrix} - \frac{{x_{4,0}}^2 \cdot x_{7,0}}{2} \tag{5-31.3}$$

etc.

$$a_{8,0} = x_{8,0} \tag{5-32.1}$$

$$a_{8,1} = -x_{4,0} \cdot x_{7,0} \tag{5-32.2}$$

$$a_{8,2} = \frac{x_{7,0} \cdot x_{9,0}}{2} \cdot \begin{pmatrix} ((x_{6,0} \cdot x_{7,0} - x_{5,0} \cdot x_{8,0}) \cdot {x_{2,0}}^2 + \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot x_{5,0} \cdot B + \\ (({x_{7,0}}^2 - {x_{8,0}}^2) \cdot x_{6,0} - x_{5,0} \cdot x_{7,0} \cdot x_{8,0}) \cdot x_{5,0} + \\ {x_{6,0}}^2 \cdot x_{7,0} \cdot x_{8,0}) \cdot {x_{4,0}}^2 \cdot A) \cdot C - D \cdot x_{7,0} \end{pmatrix} - \frac{{x_{4,0}}^2 \cdot x_{8,0}}{2} \tag{5-32.3}$$

etc.

$$a_{9,0} = x_{9,0} \tag{5-33.1}$$

$$\begin{aligned} a_{9,1} = 2 \cdot A \cdot C \cdot {x_{9,0}}^2 \cdot (x_{6,0} \cdot x_{7,0} - x_{5,0} \cdot x_{8,0}) \cdot \\ (x_{5,0} \cdot x_{7,0} + x_{6,0} \cdot x_{8,0}) \cdot (x_{2,0} - x_{4,0}) \end{aligned} \tag{5-33.2}$$

etc.

5. The single-step integration procedure (5-34) is completed by inserting the resolved parameters $a_{i,k}$ into $\Phi_i$.

$$\begin{aligned} x_i(t_0 + \Delta t) &\approx \Phi_4^{MLG_i}(t_0 + \Delta t; t_0; a_{i,0}, \ldots, a_{i,4}) \\ &= \Phi_i^{MLG}(\Delta t; x_{1,0}, \ldots, x_{9,0}; A, B, C, D) \end{aligned} \tag{5-34}$$

Note that for the circular pendulum system, the resulting expressions for the parameters $a_{i,k}$ become very large, requiring symbolic computation tools

to handle the complexity. The symbolic algebra system MAPLE[9] is used to compute the parameters $a_{i,k.}$ MAPLE transforms the results into a C program, which is then imported into our neural network simulator (an example is listed in the Appendix in chapter 8).

### 5.3.1 One-Step-Ahead Forecasts

Again, a series of experiments is performed to demonstrate the forecasting quality of the constructed neural network. First, we test the one-step-ahead forecasts for initial conditions $t_0$=0, $x_{1,0}$=1, $x_{2,0}$=0, $x_{3,0}$=2, $x_{4,0}$=0 (the other variables are derived from equations (5-21)) and step size $\Delta t$=0.001. Figure 5.28 shows the approximation quality, where the corresponding approximation errors are presented in figure 5.29. Note that only the main variables $x_1$...$x_4$ are displayed.

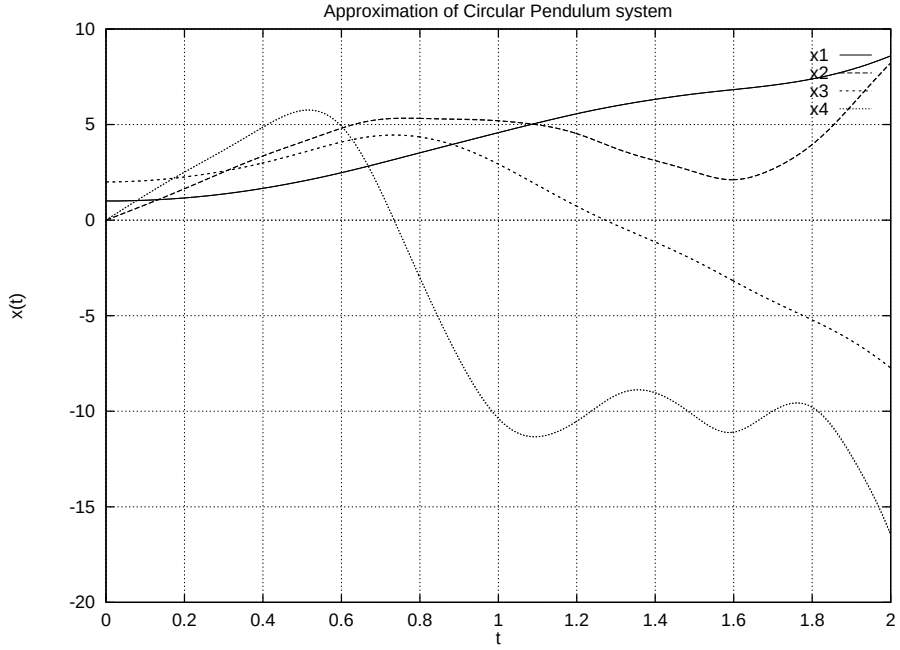



Fig. 5.28 : One-step-ahead forecasts and reference model

Figure 5.29 reveals the difficulty of approximating the circular pendulum system: The approximation errors are distributed over the whole time axis, which indicates difficult long-term forecasting where errors accumulate quickly.

9 MAPLE is a registered trademark of Waterloo Maple Software Company.

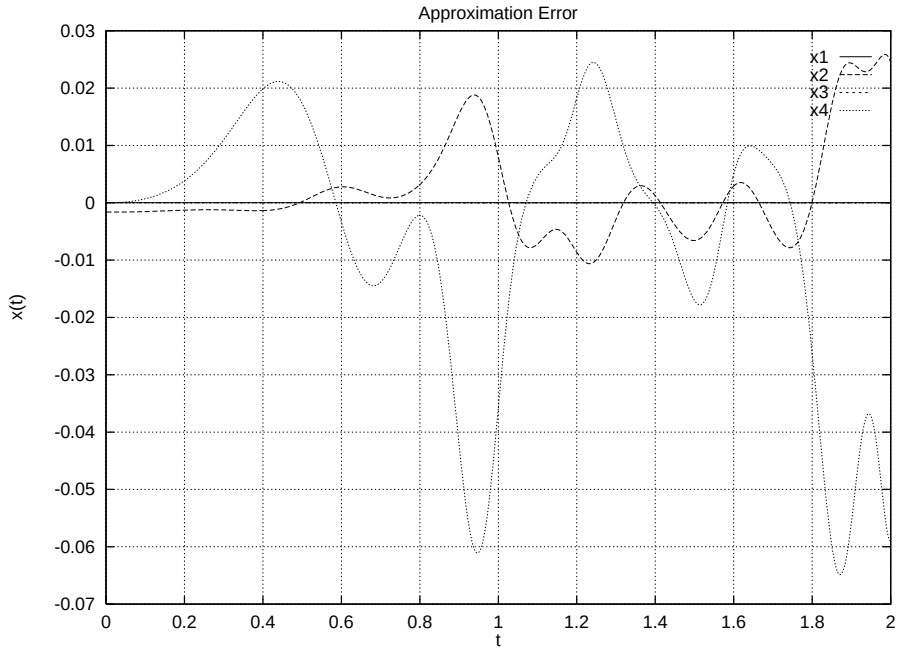



Fig. 5.29 : Approximation error of one-step-ahead forecasts

### 5.3.2 Repeated Forecasts

The second experiment tests the forecasting quality of the neural network for larger time horizons. Figure 5.30 compares the repeated forecasts with the reference model and figure 5.31 shows the related approximation errors.

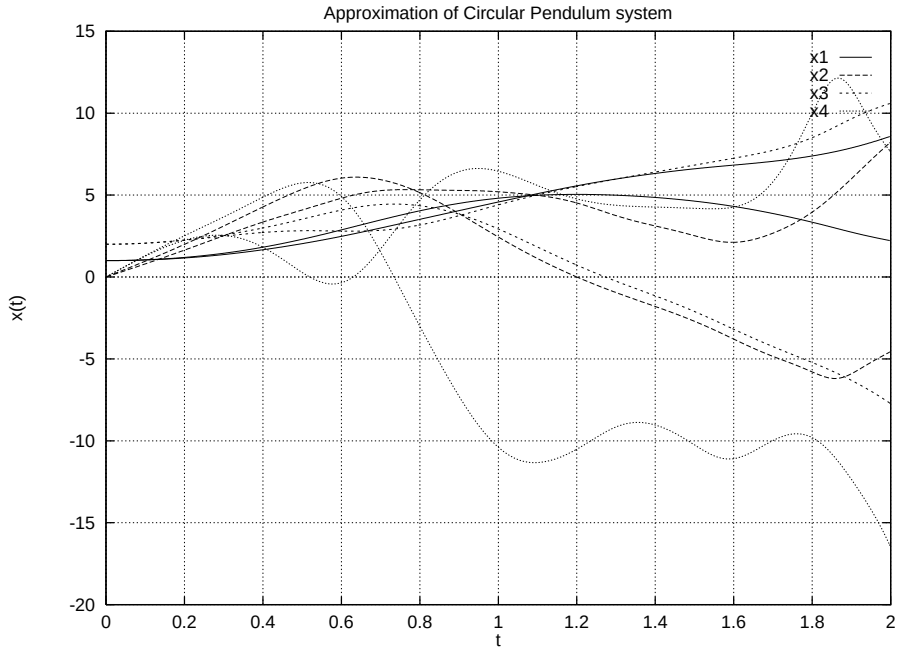



Fig. 5.30 : Repeated forecasts and reference model ($\Delta t$=0.001)

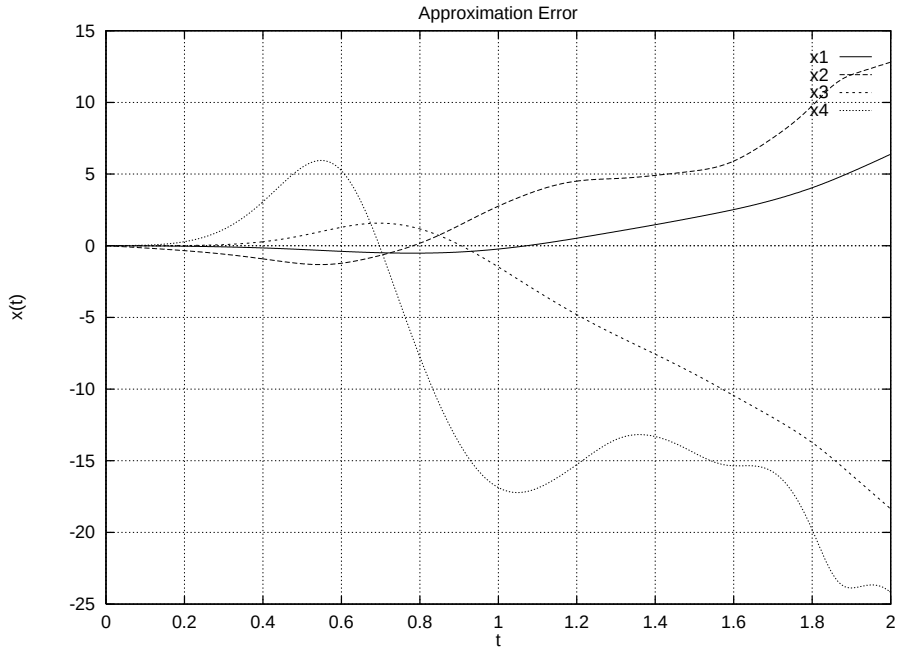



Fig. 5.31 : Approximation error of repeated forecasts ($\Delta t=0.001$)

We see that with step size $\Delta t=0.001$, the forecasts diverge rapidly from the reference model. The results for the smaller step size $\Delta t=10^{-5}$ are shown in figures 5.32 and 5.33 .

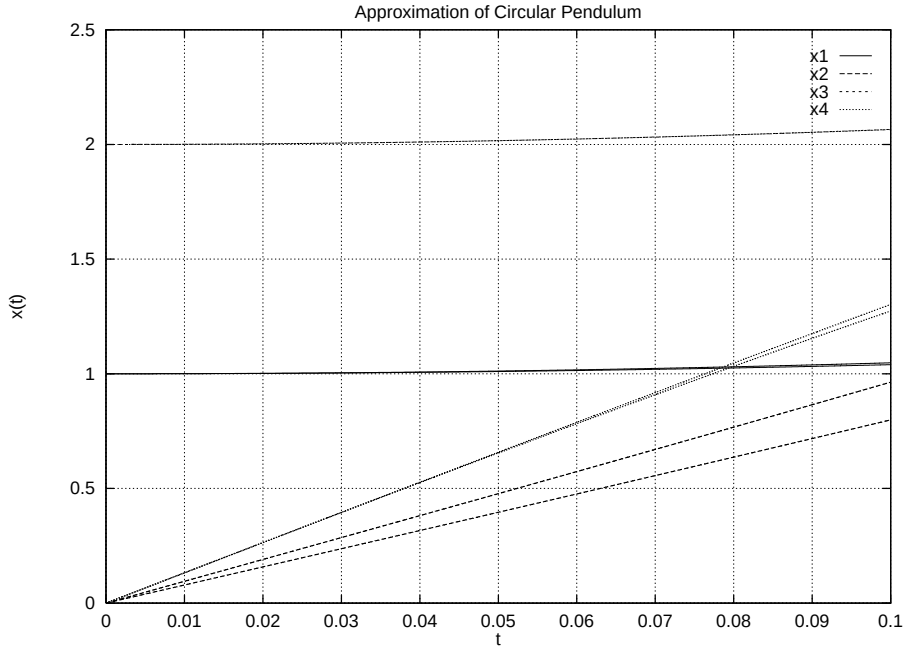



Fig. 5.32 : Repeated forecasts and reference model ($\Delta t=10^{-5}$)

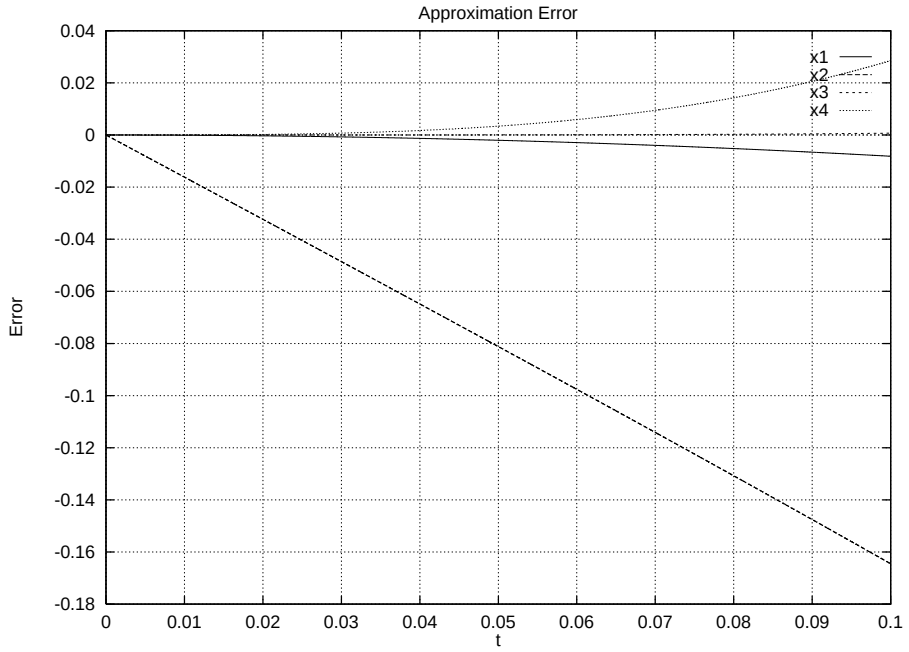



Fig. 5.33 : Approximation error of repeated forecasts ($\Delta t=10^{-5}$)

Although we have reduced the step size by a factor of 100, the approximation errors shown in figure 5.31 do not decrease significantly. The Modified Logistic Network fails to achieve good long-term forecasting quality for the circular pendulum system, because the one-step-ahead forecasting errors accumulate very quickly.

### 5.3.3 Learning of Unknown Parameters

We try to estimate the parameters A, B, C, D of the circular pendulum system with the error function from section 5.1.3 based on one-step-ahead forecasts of 1000 observations. Again, we apply the simulated annealing algorithm to learn the parameters. The evolution of the forecasting error is illustrated in figure 5.34 and the convergence of the parameters is shown in figure 5.35.

After 4927 steps, the final values for the parameters are A=-0.1605, B=12.1596, C=0.1474, D=32.5860. Obviously, the parameters found by minimizing the forecasting error do not coincide with the correct parameters. When we check the remaining forecasting errors, we observe an error of E=0.4805 for the correct parameters and E=0.20901 for the parameters selected by the optimization algorithm. Therefore, the forecasting quality of the Modified Logistic Network with 5 parameters $a_{i,k}$

and $\Delta t=0.001$ does not allow the correct set of parameters A, B, C, D to be found by globally minimizing E. This confirms our experiences with the Lorenz and the Rössler system, where the correct parameters can only be reliably estimated if we use neural networks with sufficient forecasting quality.

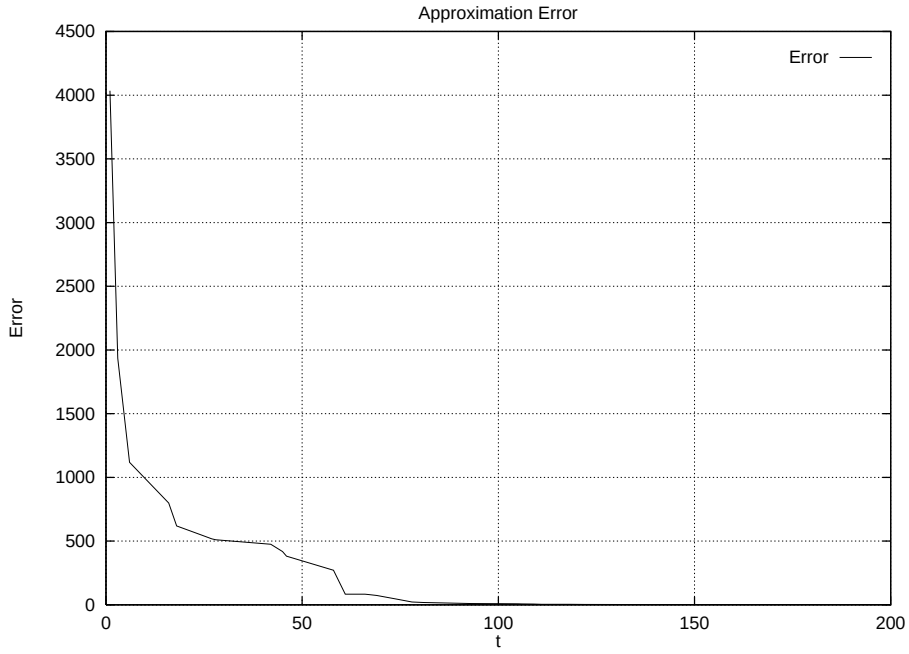



Fig. 5.34 : Forecasting error during parameter estimation

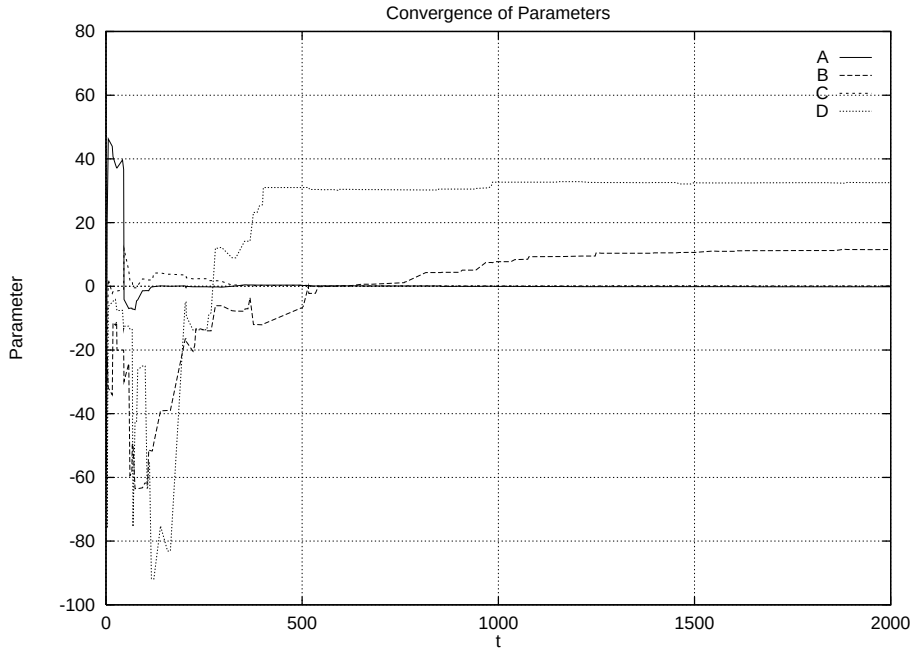



Fig. 5.35 : Convergence of learned parameters

# 6. CONCLUSIONS

In this thesis, we have explored methods for the design of neural networks that use application specific knowledge. Although it is common in the field of neural networks to replace application specific knowledge and algorithms by data and learning, neglecting available knowledge is an inefficient usage of information. For complex problems, where neural networks reveal their own specific difficulties like inefficient learning or bad generalization, the usage of a□priori knowledge can decide on success or failure. We therefore support the approach that all available information should be used, regardless of whether it is available as data or in the form of structured knowledge.

The main advantages of using a□priori knowledge for the design of neural networks are smaller needs for learning data, improved training efficiency and better generalization. The potential for such improvements varies from application to application, but our literature study has shown that using a□priori knowledge for the design of networks generates consistent benefits for a wide range of different applications.

In this thesis, we have concentrated on the usage of differential equations for the design of neural networks, because they are an important type of a□priori knowledge which has not been covered before. We have developed a method for constructing neural networks for the approximation of differential equations. The algorithm determines the structure as well as the weights of a network. It generates a set of equations for the network weights by the method of successive differentiation of the differential equations. This is a variant of the Taylor series method generalized to nonpolynomial approximation functions.

We have identified a set of conditions to guarantee the solvability of these equations, which depends on the properties of the derivatives of a neural network. Consequently, our algorithm is not restricted to one specific neural

network architecture. All classes of neural networks that satisfy the conditions can be used. Both the Radial Basis Function Networks and the Modified Logistic Networks have been shown to meet these requirements. They belong to special classes of neural networks that are able to implement polynoms of their members exactly. This property has made it much easier to derive networks that are suitable for our algorithm.

We have shown that most ordinary differential equations that are relevant for modeling physical processes can be transformed into polynomial first order systems. Consequently, the solutions of the network equations are polynomial expressions of the network inputs, which requires the introduction of a preprocessing layer in the network. If we restrict ourselves to linear differential equations, we do not need this preprocessing layer. In this case, we can keep the standard network architecture.

Therefore, the general network architecture created by the design algorithm consists of four layers: An input layer, a preprocessing layer, a middle layer and an output layer. All weights of the network are specified: The weights from the middle layer to the output layer are fixed, and the weights from the polynomial preprocessing layer to the middle layer are derived from the differential equation. Also the direct connections from the input layer to the output layer are fixed. The final architecture is called *Higher-Order Radial Basis Function Network* or *Higher-Order Modified Logistic Network* respectively.

In principle, we can approximate virtually all differential equations of practical interest with neural networks designed by our algorithm. Nevertheless, the complexity of the resulting networks and the induced numerical problems pose practical limits. For example, the transformation of a system of differential equations into polynomial form can expand its size substantially. Moreover, the size of the generated network is of cubic order relative to the number of derivatives that should match the differential equation. This means that the complexity of the preprocessing layer increases quickly. Obviously, standard numerical integration procedures achieve the same results more efficiently. However, many applications are characterized by incomplete differential equations, where observation data is available. For such applications the learning capabilities of the neural networks outweigh the disadvantages. These are the applications we are aiming for.

The neural networks obtained by our design algorithm have been used to construct single-step and multi-step numerical integration procedures. Multi-step procedures for higher order differential equations can be constructed by approximating the lower order derivatives with finite differences. This introduces additional approximation errors, but it is useful for applications, where only the main variables can be measured, but not their derivatives.

We have tested the approximation quality of the networks by forecasting some time series generated by sampling chaotic differential equations. We have examined one-step-ahead forecasts and repeated forecasts of the Lorenz system, the Rössler system and a circular pendulum system of differential equations. In all tests, the Modified Logistic Networks proved to be superior to the Radial Basis Function Networks. Both the Lorenz and the Rössler system could be approximated well, whereas the circular pendulum posed greater difficulties for long-term forecasting. These problems can be partially attributed to the high order of the system after its transformation (it was expanded into 9 differential equations) and to the complex preprocessing layer obtained (polynoms with several hundred terms). This has made it difficult to numerically evaluate the output of the networks, where decreasing the step size did not help.

Again, our design algorithm does not try to beat standard numerical integration algorithms at the approximation of differential equations. They often can perform this task more accurately and more efficiently. The main advantage consists in the integration of differential equations into a neural network model of a system that is not fully specified by the differential equations, but where additional observations of the system are available. For such cases, it is possible to exploit the learning capabilities of the networks after their initial design. There are several possibilities how to do that: For example, if a network is constructed on the basis of incomplete differential equations, it can profit from observation data by additional learning, which reduces the remaining modeling defects. Alternatively, if the differential equations contain unknown parameters, they can be estimated from observation data by neural network training algorithms.

The feasibility of the latter approach has been tested on the estimation of the parameters of the Lorenz system, the Rössler system and the circular pendulum system. We have trained the networks with an adaptive

simulated annealing algorithm, which searches for the optimal parameters that minimize the mean-squared forecasting error over a training set. We have seen that minimizing the forecasting error is quite successful. This confirms the advantages of the neural network to reduce modeling errors.

We have observed that one-step-ahead forecasts produce more stable parameter estimations than repeated forecasts. This is due to the fact that repeated forecasts amplify the network errors relative to the errors caused by imprecise parameter estimations. Consequently, while the optimization algorithm tries to improve the network forecasts, it is distracted from the correct parameters.

However, good parameter estimations are difficult to obtain with one-step-ahead forecasts as well: If the correct physical parameters of the differential equations should be identified, the minimum of the forecasting error must coincide with the correct parameters. Moreover, if the forecasting quality of the network is weak, alternative parameters will generate better forecasts. As a result, the simulated annealing algorithm will converge to the wrong set of parameters, as we have observed in the case of the circular pendulum system. This problem can be solved by using more training data, by decreasing the integration step size or by increasing the accuracy of the neural network.

In this thesis, we have emphasized the importance of using all available application knowledge for solving problems with neural networks. Although structured knowledge is usually neglected in favor of observation data and learning, we cannot afford this inefficient usage of information for complex problems. We have presented a method to exploit application specific knowledge in the form of differential equations for the design of neural networks. After initial design, these networks can still learn from observation data. This allows an optimal integration of both a□priori knowledge and of learning data into neural networks.

# 7. BIBLIOGRAPHY


[Abu-90] Abu-Mostafa, Y.S., "Learning from Hints in Neural Networks," *Journal of Complexity*, no. 6, pp. 192-198, 1990.

[Abu-93] Abu-Mostafa, Y.S., "Hints and the VC Dimension," *Neural Computation*, vol. 5, pp. 278-288, 1993.

[Ajja89] Ajjanagadde, V. and Shastri, L., "Efficient Inference with multi-place predicates and variables in a Connectionist System," in *Proceedings of the 11th annual Conference of the Cognitive Science Society,* 1989, pp. 396-403.

[Ajja90] Ajjanagadde, V. and Shastri, L., "An Optimally Efficient Limited Inference System," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-90),* 1990, pp. 563-570.

[Al-M91] Al-Mashouq, K.A. and Reed, I.S., "Including Hints in Training Neural Nets," *Neural Computation*, no. 3, pp. 418-427, 1991.

[Anan89] Anandan, P., Letovsky, S. and Mjolsness, E., "Connectionist Variable-Binding by Optimization," in *Proceedings of the 11th annual Conference of the Cognitive Science Society,* 1989, pp. 388-395.

[Ball86] Ballard, D.H., "Parallel Logical Inference and Energy Minimization," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-86), Philadelphia,* 1986, pp. 203-208.

[Barl89] Barlow, H.B., "Unsupervised Learning," *Neural Computation*, vol. 1, pp. 295-311, 1989.

[Barn91] Barnard, E. and Casasent, D., "Invariance and neural nets," *IEEE Transactions on Neural Networks*, vol. 2, pp. 498-508, 1991.

[Bart71] Barton, D., Willers, I.M. and Zahar, R.V.M., "The automatic solution of systems of ordinary differential equations by the method of Taylor series," *The Computer Journal*, vol. 14, no. 3, pp. 243-248, 1971.

[Bart80] Barton, D., "On Taylor Series and Stiff Equations," *ACM Transactions on Mathematical Software*, vol. 6, no. 3, pp. 280-294, September 1980.

[Bart83] Barto, A., Sutton, R. and Anderson, C., "Neuron-like adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834-846, 1983.

[Bart85] Barto, A.G. and Anadan, P., "Pattern Recognizing Stochastic Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, pp. 360-375, 1985.

[Bart91] Barto, A.G., Sutton, R.S. and Watkins, C.J.C.H., "Learning and Sequential Decision Making," in *Learning and Computational Neuroscience*, Gabriel, M. and Moore, J.W., Eds. Cambridge: MIT Press, 1991.

[Blum89] Blum, E.K., "Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions," *Neural Computation*, vol. 1, pp. 532-540, 1989.

[Brad89] Brady, M., Raghavan, R. and Slawny, J., "Back propagation fails to separate where perceptrons succeed," *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 665-674, 1989.

[Burg92] Burg, K., Haf, H. and Wille, F., *Höhere Mathematik für Ingenieure*, vol. I, 3rd ed. Teubner Verlag, Stuttgart, 1992.

[Camp61] Campbel, E.S., Buehler, R., Hirschfelder, J.O. and Hughes, D., "Numerical Construction of Taylor Series Approximations for a Set of Simultaneous First Order Differential Equations," *Journal of the ACM*, vol. 8, pp. 374-383, 1961.

[Chan74] Chang, Y.F., "Automatic Solution of Differential Equations," in *Constructive & computational methods for differential & integral equations, Lecture Notes in Math. 430*. Springer-Verlag, 1974.

[Cich93] Cichocki, A. and Unbehauen, R., *Neural Networks for Optimization and Signal Processing*. John Wiley & Sons, 1993.

[Cole90] Cole, R.A., Fanty, M., Muthusamy, Y. and Gopalakrishnan, M., "Speaker-Independent Recognition of Spoken English Letters," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 2, 1990, pp. 45-52.

[Corl77] Corliss, G. and Lowery, D., "Choosing a stepsize for Taylor series methods for solving ODE's," *Journal of Computational and Applied Mathematics*, vol. 3, no. 4, pp. 251-256, 1977.

[Corl82] Corliss, G. and Chang, Y.F., "Solving Ordinary Differential Equations Using Taylor Series," *ACM Transactions on Mathematical Software*, vol. 8, no. 2, pp. 114-144, June 1982.

[Cott87] Cottrell, G.W., Munro, P. and Zipser, D., "Image compression by backpropagation: An example of extensional programming," ICS Report 8702, University of California, San Diego, February 1987.

[Cott90] Cotter, N.E. and Conwell, P.R., "The Stone-Weierstrass Theorem and Its Application to Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 4, 1990.

[Cott93] Cotter, N.E. and Conwell, P.R., "Universal Approximation by Phase Series and Fixed-Weight Networks," *Neural Computation*, vol. 5, pp. 359-362, 1993.

[Cox92] Cox, E., "Integrating Fuzzy Logic into Neural Nets," *AI Expert*, pp. 43-47, June 1992.

[Cybe89] Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function," *Math. Control Signals Systems*, vol. 2, pp. 303-314, 1989.

[Dagl94] Dagli, C.H., *Artificial Neural Networks for Intelligent Manufacturing*. Chapman & Hall, 1994.

[DARP88] *DARPA Neural Network Study*. AFCEA International Press, 1988.

[DeMe92] DeMers, D. and Kreutz-Delgado, K., "Learning Global Direct Inverse Kinematics," in *Advances in Neural Information Processing Systems*, Moody, J.E., Hanson, S.J. and Lippmann, R.P., Eds. Morgan Kaufmann Publishers, Inc., 1992, pp. 589-594.

[Dorr91] Dorronsoro, J.R. and López, V., "Neural Network Learning of Polynomial Formats for Coupled Time Series," in *Artificial Neural Networks*, Kohonen, T., Mäkisara, K., Simula, O. and Kangas, J., Eds. North-Holland, 1991, pp. 201-206.

[Dyer89a] Dyer, M.G. and Lange, T.E., "High-Level Inferencing in a Connectionist Network," *Connection Science*, vol. 1, no. 2, pp. 181-217, 1989.

[Dyer89b] Dyer, M.G. and Lange, T.E., "Frame selection in a connectionist model of high-level inferencing," in *Proceedings of the 11th annual Conference of the Cognitive Science Society,* 1989, pp. 706-713.

[Fair88] Fairén, V., López, V. and Conde, L., "Power series approximation to solutions of nonlinear systems of differential equations," *Am. J. Phys.*, vol. 56, no. 1, January 1988.

[Fran91] Franklin, S. and Garzon, M., "Neural Computability," in *Progress in Neural Networks*, Omidvar, O.M., Ed. Ablex Publishing Corp., 1991, pp. 127-145.

[Fu94] Fu, L.M., "Rule Generation from Neural Networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 8, pp. 1114-1124, August 1994.

[Funa89] Funahashi, K.I., "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

[Gall93] Gallant, S.I., *Neural Network Learning and Expert Systems*. MIT Press, 1993.

[Gile92] Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z. and Lee, Y.C., "Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks," *Neural Computation*, no. 4, pp. 393-405, 1992.

[Giro90] Girosi, F. and Poggio, T., "Networks and the Best Approximation Property," *Biological Cybernetics*, vol. 63, pp. 169-176, 1990.

[Good92] Goodman, R.M., Higgins, C.M. and Miller, J.W., "Rule-Based Neural Networks for Classification and Probability Estimation," *Neural Computation*, vol. 4, pp. 781-804, 1992.

[Gori92] Gori, M.T.A. and Tesi, A., "On the local minima in backpropagation.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76-86, 1992.

[Guez88] Guez, A. and Selinsky, J., "A Trainable Neuromorphic Controller," *Journal of Robotic Systems*, vol. 5, no. 4, pp. 363-388, 1988.

[Guyo89] Guyon, L., Poujaud, I., Personnaz, L., Dreyfus, G., Denker, J. and Le Cun, Y., "Comparing different neural network architectures for classifying handwritten digits," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 2, IEEE Press, 1989, pp. 127-132.

[H90a] Hölldobler, S., "A structured connectionist unification algorithm," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-90),* 1990.

[H90b] Hölldobler, S., "A connectionist unification algorithm," Technical Report TR-90-012, International Computer Science Institute, Berkeley, 1990.

[H90c] Hölldobler, S., "CHCL: a connectionist inference system for horn logic based on the connection method and using limited resources," Technical Report TR-90-042, International Computer Science Institute, Berkeley, 1990.

[Hair91] Hairer, E., Nørsett, S.P. and Wanner, G., *Solving Ordinary Differential Equations I: nonstiff problems*, vol. 8, Springer Series in Computational Mathematics, 2nd ed. Springer-Verlag, 1991.

[Hart90] Hartman, E.J., Keeler, J.D. and Kowalski, J.M., "Layered Neural Networks with Gaussian Hidden Units as Universal Approximations," *Neural Computation*, vol. 2, pp. 210-215, 1990.

[Hayk94] Haykin, S., *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company Inc., 1994.

[Hech90] Hecht-Nielsen, R., *Neurocomputing*. Addison-Wesley Publish., 1990.

[Henr56] Henrici, P., "Automatic Computations with Power Series," *Journal of the ACM*, vol. 3, no. 1, pp. 15, 1956.

[Hert91] Hertz, J., Krogh, A. and Palmer, R.G., *Introduction to the Theory of Neural Computation*. Addison-Wesley Publish., 1991.

[Hint89] Hinton, G.E., "Connectionist Learning Procedures," *Artificial Intelligence*, vol. 40, pp. 185-234, 1989.

[Hopt93] Hoptroff, R.G., "The Principles and Practice of Time Series Forecasting and Business Modelling Using Neural Nets," *Neural Computing & Applications*, no. 1, pp. 59-66, 1993.

[Hori92] Horikawa, S., Furuhashi, T. and Uchikawa, Y., "On Fuzzy Modeling Using Fuzzy Neural Networks with the Back-propagation Algorithm," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 801-806, September 1992.

[Horn90] Hornik, K., Stinchcombe, M. and White, H., "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Networks*, vol. 3, pp. 551-560, 1990.

[Ingb89] Ingber, L., "Very Fast Simulated Re-Annealing," *Mathl. Comput. Modelling*, vol. 12, pp. 967-973, 1989.

[Ingb93] Ingber, L., "Simulated Annealing: Practice versus Theory," *Mathl. Comput. Modelling*, vol. 18, no. 11, pp. 29-57, 1993.

[Intr92] Intrator, N., Gold, J.I., Bülthoff, H.H. and Edelman, S., "3D Object Recognition Using Unsupervised Feature Extraction," in *Advances in Neural Information Processing Systems*, Moody, J.E., Hanson, S.J. and Lippmann, R.P., Eds. Morgan Kaufmann Publishers, Inc., 1992, pp. 460-470.

[Ito92] Ito, Y., "Approximations of Continuous Functions on $R^d$ by Linear Combinations of Shifted Rotations of a Sigmoid Function With and Without Scaling," *Neural Networks*, vol. 5, pp. 105-115, 1992.

[Jabr92] Jabri, M., Pickard, S., Leong, P., Chi, Z., Flower, B. and Xie, Y., "ANN Based Classification of Heart Defibrillators," in *Advances in Neural Information Processing Systems*, Moody, J.E., Hanson, S.J. and Lippmann, R.P., Eds. Morgan Kaufmann Publishers, Inc., 1992, pp. 637-644.

[Jack88] Jackel, L.D., Graf, H.P., Hubbard, W., Denker, J.S., Henderson, D. and Guyon, I., "An application of neural net chips: Hadwritten digit recognition," in *Proceedings of the International Conference on Neural Networks,* vol. 2, IEEE Press, 1988, pp. 107-115.

[Jang93] Jang, J.S.R. and Sun, C.T., "Functional Equivalence between Radial Basis Function Networks and Fuzzy Inference Systems," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 156-158, 1993.

[Keel92] Keeler, J. and Rumelhart, D.E., "A Self-Organizing Integrated Segmentation and Recognition Neural Net," in *Advances in Neural Information Processing Systems*, Moody, J.E., Hanson, S.J. and Lippmann, R.P., Eds. Morgan Kaufmann Publishers, Inc., 1992, pp. 589-594.

[Kern81] Kerner, E.H., "Universal formats for nonlinear ordinary differential systems," *J. Math. Phys.*, vol. 22, no. 7, pp. 1366-1371, July 1981.

[Kimo90] Kimoto, T., Asakawa, K., Yoda, M. and Takeoka, M., "Stock Market Prediction System with Modular Neural Networks," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 2, 1990, pp. 1-6.

[Koho89] Kohonen, T., *Selforganization and Associative Memory*. Springer, 1989.

[Kosk91] Kosko, B., *Neural Networks and Fuzzy Systems*. Prentice-Hall, 1991.

[Kurf89] Kurfess, F. and Reich, M., "Logic and Reasoning with Neural Models," in *Connectionism in Perspective*, Pfeifer, R., Schreter, Z., Fogelman-Soulié, F. and Steels, L., Eds. Elsevier Publishers, North-Holland, 1989.

[Leav66] Leavitt, J.A., "Methods and Applications of Power Series," *Mathematics of Computation*, vol. 20, pp. 46-52, 1966.

[Lee86] Lee, Y.C., Doolen, G., Chen, H.H., Sun, G.Z., Maxwell, T., Lee, H.Y. and Giles, C.L., "Machine Learning using a higher order correlation network," *Physica D*, vol. 22, pp. 276-306, 1986.

[Lipp87] Lippmann, R.P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

[Lipp89] Lippmann, R.P., "Review of Neural Networks for Speech Recognition," *Neural Computation*, vol. 1, pp. 1-38, 1989.

[Lore63] Lorenz, E.N., "Deterministic Nonperiodic Flow," *Journal of the Atmospheric Sciences*, vol. 20, pp. 130-141, March 1963.

[Mann90] Mannaert, H. and Oosterlinck, A., "A Self-Organizing System for Object Analysis and Identification," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 2, 1990, pp. 819-824.

[Mats90] Matsuyama, Y., "Competitive Self-Organization and Combinatorial Optimization: Applications to Traveling Salesman Problem," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 3, 1990, pp. 819-824.

[Mill90] Miller, W.T., Sutton, R.S. and Werbos, P.J., *Neural Networks for Control*. MIT Press, 1990.

[Mins67] Minsky, M.L., *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[Mjol90] Mjolsness, E. and Garrett, C., "Multiscale Optimization in Neural Nets: Preliminary Report," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 3, 1990, pp. 781-786.

[Mood92] Moody, J.E., Hanson, S.J. and Lippmann, R.P., *Advances in Neural Information Processing Systems*, vol. 4. Morgan Kaufmann Publishers, Inc., 1992.

[NAG94] *The NAG Fortran Library Manual, Mark 16*, vol. 2. The Numerical Algorithms Group Limited, 1994.

[Nara92] Narazaki, H. and Ralescu, A.L., "A Connectionist Approach for Rule-Based Inference Using an Improved Relaxation Method," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 741-751, September 1992.

[Nare90] Narendra, K.S. and Parthasarathy, K., "Identification and Control of Dynamical Systems using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4-27, 1990.

[Nasr88] Nasrabadi, N.M. and King, R.A., "Image Coding Using Vector Quantization: A Review," *IEEE Transactions on Communications*, vol. 36, pp. 957-971, 1988.

[Norm76] Norman, A.C., "Expanding the solutions of implicit sets of ordinary differential equations in power series," *The Computer Journal*, vol. 19, no. 1, pp. 63-68, 1976.

[Pack80] Packard, N.H., Crutchfield, J.P., Farmer, J.D. and Shaw, R.S., "Geometry from a Time Series," *Physical Review Letters*, vol. 45, no. 9, pp. 712-716, 1980.

[Park91] Park, J. and Sandberg, I.W., "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, vol. 3, pp. 246-257, 1991.

[Park93] Park, J. and Sandberg, I.W., "Approximation and Radial-Basis-Function Networks," *Neural Computation*, vol. 5, pp. 305-316, 1993.

[Pina91] Pinas, G., "Propositional Non-Monotonic Reasoning and Inconsistency in Symmetric Neural Networks," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),* 1991.

[Pine87] Pineda, F.J., "Generalization of Backpropagation to Recurrent and Higher Order Neural Networks," in *Proceedings of the IEEE Conference on Neural Information Processing Systems (NIPS),* IEEE, 1987, pp. 602-611.

[Pink91] Pinkas, G., "Symmetric Neural Networks and Propositional Logic Satisfiability," *Neural Computation*, no. 3, pp. 282-291, 1991.

[Pogg90] Poggio, T., "Networks for Approximation and Learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481-1497, September 1990.

[Poll87] Pollack, J.B., "On Connectionist Models of Natural Language Processing, MCCS-87-100," Ph.D. thesis, New Mexico State University, 1987.

[Powe87] Powell, M.J.D., “Radial basis functions for multivariable interpolation: A review,” in *Algorithms for Approximation*, Mason, J.C. and Cox, M.G., Eds. Clarendon Press, Oxford, 1987.

[Ritt91] Ritter, H., Martinetz, T. and Schulten, K., *Neuronale Netze*. Addison-Wesley, 1991.

[Roes76] Rössler, O.E., “An Equation for Continuous Chaos,” *Physics Letters*, vol. 57 A, no. 5, pp. 397-398, July 1976.

[Roge93] Roger Jang, J.S. and Sun, C.T., “Functional Equivalence between Radial Basis Function Networks and Fuzzy Inference Systems,” *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 156-158, 1993.

[Rott70] Rott, N., “A multiple pendulum for the demonstration of non-linear coupling,” Swiss Federal Institute of Technology, to Prof. Hans Ziegler, on his sixtieth birthday, ~1970.

[Rume86a] Rumelhart, D.E., Hinton, , G. E. and Williams, R.J., “Learning internal representations by error backpropagation,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Rumelhart, D.E. and McClelland, J.L., Eds. MIT Press, 1986, pp. 318-362.

[Rume86b] Rumelhart, D.E. and McClelland, J.L., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. I & II. MIT Press, 1986.

[Sche94] Scheiben, E., “Modellierung eines Doppelpendels,” Technical Report, ABB Corporate Research, Baden, 1994.

[Schu88] Schuster, H.G., *Deterministic Chaos*, 2nd ed. VCH Verlagsgesellschaft mbH, 1988.

[Sejn86] Sejnowski, T.J., “Higher-Order Boltzmann Machines,” *Proceedings of American Institute of Physics*, vol. 151, pp. 398-403, 1986.

[Shas87] Shastri, L., "A Connectionist Encoding of Semantic Networks," in *Proceedings of the 9th annual Conference of the Cognitive Science Society,* 1987, pp. 143-154.

[Sont89] Sontag, E.D. and Sussmann, H.J., "Backpropagation can give rise to spurious local minima even for networks without hidden layers," *Complex Systems*, vol. 3, pp. 91-106, 1989.

[Stin89] Stinchcombe, M. and White, H., "Universal Approximation Using Feedforward Networks With Non-Sigmod Hidden Layer Activation Functions," in *Proceedings of the International Joint Conference on Neural Networks,* vol. I, IEEE, 1989, pp. 613-617.

[Stol89] Stolcke, A., "Processing Unification-based Grammars in a Connectionist Framework," in *Proceedings of the 11th annual Conference of the Cognitive Science Society,* 1989.

[Sudd91] Suddarth, S.C., "Symbolic neural systems and the use of hints for developping complex systems," *International Journal for Man-Machine Studies*, no. 35, pp. 291-311, 1991.

[Sulz93] Sulzberger, S.M., Tschichold-Gürman, N.N. and Vestli, S.J., "FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks," in *Proceedings of ICNN'93, San Francisco,* 1993.

[Taka92] Takagi, H., Suzuki, N., Koda, T. and Kojima, Y., "Neural Networks Designed on Approximate Reasoning Architecture and Their Applications," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 752-760, September 1992.

[Tour85] Touretzky, D.S. and Hinton, G.E., "Symbols Among the Neurons: Details of a Connectionist Inference Architecture," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),* 1985.

[Towe90] Towell, G.G., Shavlik, J.W. and Noordewier, M.O., "Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks," in *Proceedings of 8th National Conference on AI, Boston,* 1990, pp. 861-865.

[Towe91] Towell, G.G., Craven, M.W. and Shavlik, J.W., "Constructive Induction in Knowledge-Based Neural Networks," in *Proceeding of 8th Machine Learning Workshop, Chicago,* 1991, pp. 213-217.

[Tsch93] Tschichold-Gürmann, N., "RuleNet: An Artificial Neural Network Model for Cooperation with Knowledge Based Systems," *submitted to the Special Issue of The Machine Learning Journal on Symbolic Knowledge and Neural learning*, 1993.

[Twom93] Twomey, J.M. and Smith, A.E., "Nonparametric Error Estimation Methods for Evaluating and Validating Artificial Neural Network Prediction Models," in *Intelligent Systems through Artificial Neural Networks, Proceedings of ANNIE'93,* vol. 3, 1993, pp. 233-238.

[Vcrk90] Vcrkovnik, C., Carter, C.R. and Haykin, S., "Radial Basis Function Classification of Impulse Radar Waveforms," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN),* vol. 2, 1990, pp. 45-50.

[Wang90] Wang, S.D. and Yeh, H.M.S., "Self-Adaptive Neural Architectures for Control Applications," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN),* vol. 3, 1990, pp. 309-314.

[Wang92] Wang, L.X. and Mendel, J.M., "Fuzzy Basis Functions, Universal Approximation and Orthogonal Least-Squares Learning," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, 1992.

[Watr87] Watrous, R.L., "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," in *Proceedings of the IEEE Conference on Neural Networks,* vol. II, IEEE, 1987, pp. 619-627.

[Watr92] Watrous, R.L. and Kuhn, G.M., "Induction of Finite-State Languages using Second-Order Recurrent Networks," *Neural Computation*, no. 4, pp. 406-414, 1992.

[Weig90] Weigend, A.S., Hubermann, B. and Rumelhart, D.E., "Predicting The Future: A Connectionnist Approach," *International Journal of Neural Systems*, vol. 1, no. 3, pp. 193-209, 1990.

[Weis91] Weiss, S.M. and Kulikowski, C.A., *Computer Systems that Learn*. Morgan Kaufmann, 1991.

[Whit92] White, D.A. and Sofge, D.A., *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. Multiscience Press, Inc., 1992.

[Wong91] Wong, F.S., "Time series forecasting using backpropagation neural networks," in *Neurocomputing*. vol. 2, 1991, pp. 147-159.

[Wu84] Wu, W., "Basic Principles of Mechanical Theorem Proving in Geometries," *Journal of Sys. Sci. and Math. Sci.*, vol. 3, no. 4, pp. 207-235, 1984.

[Yama90] Yamamoto, Y. and Venkatasubramanian, V., "Integrated Approach Using Neural Networks for Fault Detection and Diagnosis," in *Proceedings of the International Joint Conference on Neural Networks,* vol. 2, 1990, pp. 317-326.

# 8. APPENDIX

## 8.1 Example MAPLE Program for the Lorenz System

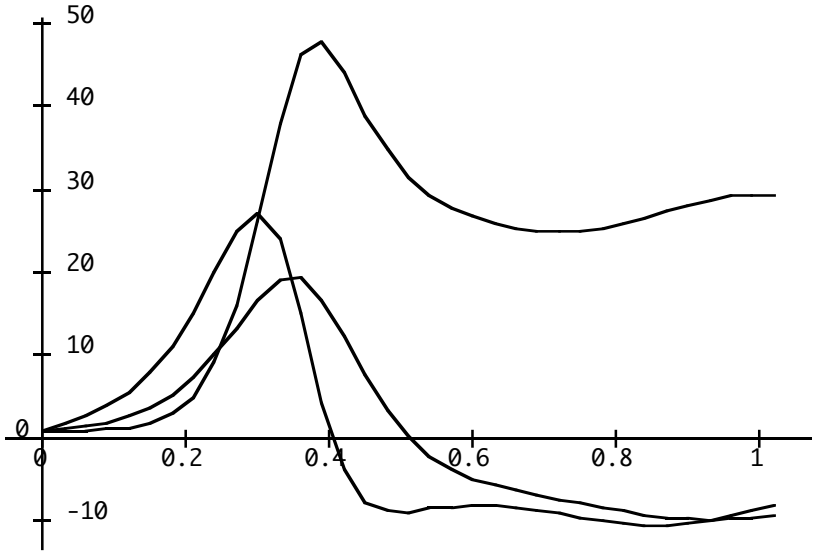
The following program source code has been used for the calculation of the parameters $a_{i,k}$ (here $a_k$, $b_k$, $c_k$) of the Radial Basis Function Networks that approximate the Lorenz system. The results are converted into C code, which is imported into our neural network simulator. The Rössler system and the circular pendulum system are treated similarly.

### *Lorenz chaotic differential equation*

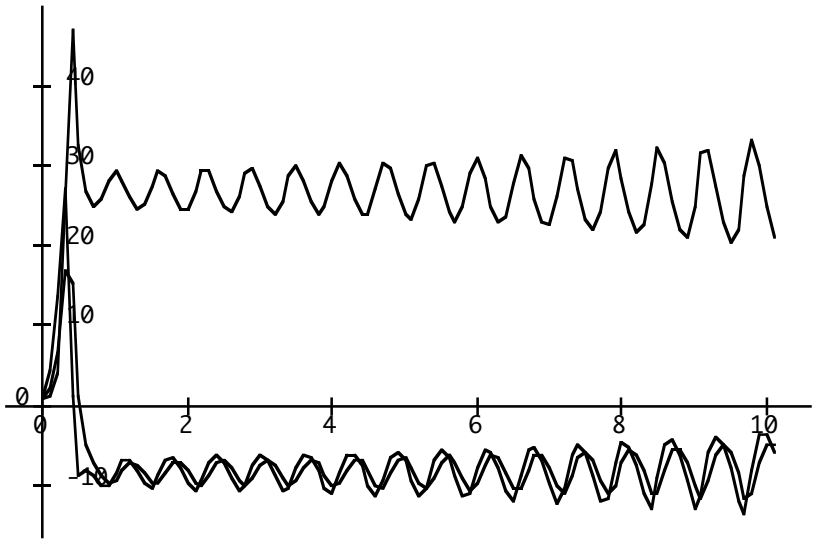
```
• eqns := {
diff(x1(t),t) = A*(x2(t)-x1(t)),
diff(x2(t),t) = -x1(t)*x3(t)+B*x1(t)-x2(t),
diff(x3(t),t) = x1(t)*x2(t)-C*x3(t)}:

• params := {
A = 10,
B = 28,
C = 8/3}:

• init := {
x1(0) = 1,
x2(0) = 1,
x3(0) = 1}:
sys := subs(params,eqns union init);
            sys := {x1(0) = 1, x2(0) = 1, x3(0) = 1,

                  d
                 ---- x1(t) = 10 x2(t) - 10 x1(t),
                  dt

                  d
                 ---- x2(t) = - x1(t) x3(t) + 28 x1(t) - x2(t),
                  dt

                  d
                 ---- x3(t) = x1(t) x2(t) - 8/3 x3(t)}
                  dt

• r := dsolve(sys,{x1(t),x2(t),x3(t)},numeric);
r := proc(t) `dsolve/numeric/result2`(t,8577904,[1,1,1]) end
```

```
• with(linalg):
plotres :=
proc(f,first,last,dt)
local i,j,set,res;
res := matrix([[f(first+i*dt)]$i=0..(last-first)/dt+1]);
set := {[[res[i,1],res[i,j]]$i=1..rowdim(res)]$j=2..coldim(res)};
set;
end:
```

```
• plot(plotres(r,0,1.0,0.03));
```




```
• plot(plotres(r,0,10.0,0.1));
```




*Radial-Basis-Function Network*

```
• with(linalg,vectdim):
rbf := proc(x,t,a,b,n)
local i,j;
a[0]+sum('(exp(-1)-exp(-((x-t)-1)^2))^(j-1)*
sum('b[i]/(2^i*i!*exp(-(i-1)))*(exp(-1)-exp(-(a[j]*(x-t)-1)^2))^i',
```

```
'i'=1..min(n,vectdim(b)))',
'j'=1..min(n,vectdim(b)));
end:
```

*Calculation of parameters b*

```
• with(linalg,vectdim):
rbfsimple := proc(x,t,a,n)
local i,b;
b := [1,1,5,25,209,1961,23589];
a+sum('b[i]/(2^i*i!*exp(-(i-1)))*(exp(-1)-exp(-(a*(x-t)-1)^2))^i',
'i'=1..min(n,vectdim(b)));
end:
paramb := proc(x,t,a,n)
(exp(1)*(-1)^(n-1))/(a^n)*subs(x=t,diff(rbfsimple(x,t,a,n-1),x$n));
end:

• [1,['simplify(paramb(x,t,a,i))'$'i'=2..7]];
                    [1, [1, 5, 25, 209, 1961, 23589]]
```

*Calculation of derivatives*

```
• drbf :=
proc(x,t,a,b,n)
local i,j;
[subs([x=t],rbf(x,t,a,b,vectdim(b))),
simplify(subs([x=t],['diff(rbf(x,t,a,b,vectdim(b)),x$j)'$'j'=1..n]))];
end:

• drbf(x,t,a,[1,1,5,25,209,1961,23589],3);
[a[0],

    [- a[1] exp(-1), 4 exp(-2) a[2], - 24 exp(-3) a[3] + 6 exp(-2) a[2]]]
```

*Modified-Logistic-Network*

```
• mlg := proc(x,t,a,n)
local i,j;
a[0]+sum('tanh(x-t)^(i-1)*sum('tanh(a[i]*(x-t))^(2*j+1)/(2*j+1)',
'j'=0..n/2)','i'=1..n);
end:
```

*Calculation of derivatives*

```
• dmlg :=
proc(x,t,a,m,n)
local i,j;
[simplify(subs([x=t],mlg(x,t,a,m))),
simplify(subs([x=t],['diff(mlg(x,t,a,m),x$j)'$'j'=1..n]))];
end:


• dmlg(x,t,a,5,5);
 [a[0], [a[1], 2 a[2], 6 a[3], - 8 a[2] + 24 a[4], - 80 a[3] + 120 a[5]]]
```

*Approximation of Lorenz equations with Radial Basis Function Network*

```
x1' = a(*x2-*x1)
x2' = -x1*x3+b*x1-x2
x3' = x1*x2-c*x3

• dx1 := A*(x2(t)-x1(t)):
dx2 := -x1(t)*x3(t)+B*x1(t)-x2(t):
dx3 := x1(t)*x2(t)-C*x3(t):
lorenzeqns := {diff(x1(t),t)=dx1,diff(x2(t),t)=dx2,diff(x3(t),t)=dx3}:
lorenzinit := {x1(t0)=x[1],x2(t0)=x[2],x3(t0)=x[3]}:
```

*Setup network and equations*

```
• with(linalg):
rbf := proc(x,t,a,b,n)
local i,j;
a[0]+sum('(exp(-1)-exp(-((x-t)-1)^2))^(j-1)*
sum('b[i]/(2^i*i!*exp(-(i-1)))*(exp(-1)-exp(-(a[j]*(x-t)-1)^2))^i',
'i'=1..min(n,vectdim(b)))',
'j'=1..min(n,vectdim(b)));
end:
fac2 := proc(x,t,h,i,k)
local p,j;
p := 1;
for j from 0 to k-1
do
     if (j <> i)
     then
          p := p*(x-(t-j*h))/((t-i*h)-(t-j*h));
     fi;
od;
p
end:


• f := proc(x,t,a,n)
rbf(x,t,a,[1,1,5,25,209,1961,23589],n);
end:

• itereqns := proc(expr,t,t0,subst,init,net,n)
local eqn,ex,i;
ex := expr;
for i from 1 to n do
     eqn[i] := (simplify(subs(t=t0,diff(net,t$i))) =
                    simplify(subs(init,subs(t=t0,ex))));
     ex := subs(subst,diff(ex,t));
od;
{'eqn[i]'$'i'=1..n};
end:

• x1eqns := itereqns(dx1,t,t0,lorenzeqns,lorenzinit,f(t,t0,a,3),3):
x2eqns := itereqns(dx2,t,t0,lorenzeqns,lorenzinit,f(t,t0,b,3),3):
x3eqns := itereqns(dx3,t,t0,lorenzeqns,lorenzinit,f(t,t0,c,3),3):
```

*Solve for parameters*

```
• x1res := convert(factor(solve(x1eqns,{'a[i]'$'i'=1..3})),horner);
x2res := convert(factor(solve(x2eqns,{'b[i]'$'i'=1..3})),horner);
x3res := convert(factor(solve(x3eqns,{'c[i]'$'i'=1..3})),horner);

                     A (- x[1] x[3] + B x[1] - x[2] + A (- x[2] + x[1]))
x1res := {a[2] = 1/4 ---------------------------------------------------,
                                        exp(-2)

          A (- x[2] + x[1])
   a[1] = -----------------,
               exp(-1)

   a[3] = 1/48 A (- 5 x[2] + (- 5 x[3] + 2 x[1] x[2]) x[1] - 2 x[1] C x[3]

        + 5 B x[1] + ((2 x[3] - 5) x[2] + (- 4 x[3] + 3) x[1]

        + (- 2 x[2] + 4 x[1]) B + (- 2 x[2] + 2 x[1]) A) A)/exp(-3)

   }

x2res := {

 b[3] = - 1/48 (- 5 x[2] + (- 5 x[3] + (7 x[2] + 2 %1) x[1]) x[1]

      + ((- 5 x[3] + 2 x[1] x[2]) x[1] - 2 x[1] C x[3]) C

                    2
      + (5 - 2 x[1] ) x[1] B + (7 %2

                                      2
      + ((2 x[3] - 5) x[3] - 6 x[2]  + 6 x[1] x[2]) x[1] + (4 %2 - 4 %1) C

      + (- 7 x[2] + (- 4 x[3] + 5) x[1] + 2 B x[1]) B

      + (2 %2 - 2 %1 + (- 2 x[2] + 2 x[1]) B) A) A)/exp(-3),

 b[2] = - 1/4 (- x[2] + (- x[3] + x[1] x[2]) x[1] - x[1] C x[3] + B x[1]

      + (%2 - %1 + (- x[2] + x[1]) B) A)/exp(-2),

          - %1 + B x[1] - x[2]
 b[1] = - --------------------}
                 exp(-1)

%1 :=                          x[1] x[3]

%2 :=                          x[3] x[2]

x3res := {

c[2] = - 1/4

                                                          2          2
    (x[2] + x[1] x[3]) x[1] + (%1 - C x[3]) C - B x[1]  + (- x[2]  + %1) A
    ----------------------------------------------------------------------
                                  exp(-2)

    ,
```

```
       - %1 + C x[3]
c[1] = -------------,
          exp(-1)

c[3] = - 1/48 ((5 x[2] + (5 x[3] - 2 %1) x[1]) x[1]

     + ((5 x[2] + 4 x[1] x[3]) x[1] + (- 3 x[3] + 2 %1 - 2 C x[3]) C) C

                 2           2               2
     + (- 5 x[1]  - 2 C x[1] ) B + (- 9 x[2]

                                                          2
     + ((- 8 x[3] + 7) x[2] + 6 x[1] x[3]) x[1] + (2 %1 - 2 x[2] ) C

                                                 2
     + (8 x[2] - 6 x[1]) x[1] B + (2 %1 - 2 x[2] ) A) A)/exp(-3)       }

%1 :=                           x[1] x[2]
```

*Conversion to C source code*

```
• readlib(C):
writeto(`maplecode.c`):
C([x1res[i]$i=1..nops(x1res),x2res[i]$i=1..nops(x2res),
x3res[i]$i=1..nops(x3res)]):
writeto(terminal):
```

## 8.2 Description of Implemented Software

The following sections describe the software that was implemented for creating reference models and training data, for training neural networks and for generating neural network forecasts.

### 8.2.1 Odesim

The program *odesim* approximates the solution of a differential equation by using the procedure *D02PCF* from the NAG-library (see [NAG94]), which is based on an adaptive Runge-Kutta method. A time series of observations is obtained by sampling the approximation in regular time intervals. The resulting time series is used as a reference model for the approximation with neural networks (see chapter 5). The following parameters have to be specified for the program *odesim*:

```
usage: odesim {parameterfile} {parameter=value} >output

parameters:

start        starting time t
```

```
end          end time t

dt           time increment

ode          name of differential equation (lorenz,roessler,pendel)

method       integration method (1-3)

init         comma-separated list of initial values

params       comma-separated list of ode parameters

output       time series obtained by sampling the approximation of the
             differential equation 'ode'
```

### 8.2.2 Predictnet

The program *predictnet* computes the neural network forecasts for a given differential equation. The result is a time series of forecasts whose quality can be compared to the reference model. The following parameters have to be specified for the program *predictnet*:

```
usage: predictnet {parameterfile} {parameter=value} >output

parameters:

net          network type (rbf,mlg)

units        number of network parameters

ode          name of differential equation (lorenz,roessler,pendel)

params       comma-separated list of ode parameters

input        input data file generated by 'odesim'

first        index of first test example, negative numbers count from
             end of data

last         index of last test example, negative numbers count from
             end of data

mode         forecasting mode (single,repeated)

output       time series obtained by forecasting the differential
             equation with the neural network 'net'
```

### 8.2.3 Trainnet

The program *trainnet* learns the best parameters of a differential equation that minimize the forecasting error of a neural network. Learning is performed with an adaptive simulated annealing algorithm (see [Ingb89]). The following parameters have to be specified for the program *trainnet*:

```
usage: trainnet {parameterfile} {parameter=value} >output

parameters:

net           network type (rbf,mlg)

units         number of network parameters

ode           name of differential equation (lorenz,roessler,pendel)

input         input data file generated by 'odesim'

errlog        log file for errors and parameters

steps         maximal number of function evaluations allowed

first         index of first test example, negative numbers count from
              end of data

last          index of last test example, negative numbers count from
              end of data

mode          forecasting mode (single,repeated)

output        progress of learning the best parameters that minimize the
              forecasting error
```

# LIST OF SYMBOLS

| | |
|---|---|
| $R^+$ | Set of all positive real numbers |
| $R^n$ | Set of all n-dimensional real vectors |
| $C(D)$ | Set of continuous real-valued functions on D |
| $\|\|...\|\|_2$ | Euclidean norm |
| $\|...\|$ | Absolute value |
| $\lfloor x \rfloor$ | The largest integer y, where $x \geq y$ |
| $[a,b]$ | Interval of real numbers x, where $a \leq x \leq b$ |
| $[a,b)$ | Interval of real numbers x, where $a \leq x < b$ |
| $(a,b)$ | Interval of real numbers x, where $a < x < b$ |
| $\overline{x} \cdot \overline{y}$ | Scalar product of two vectors |
| det A | Determinant of matrix A |
| $f(t)\|_{t=t_0}$ | Function f(t) evaluated for argument $t=t_0$ |
| $f(t)^{(k)}\|_{t=t_0}$ | K-th derivative of f(t) evaluated for argument $t=t_0$ |
| $f^{(k)}(t_0)$ | K-th derivative of f(t) evaluated for argument $t=t_0$ |
| $f^{-1}$ | Functional inverse of f |
| $P_i(x...y)$ | A polynom $P_i$ of the arguments x...y |
| $\sum_{i=m}^{n} x_i$ | Sum of numbers $x_i$, where i is an integer $m \leq i \leq n$ |
| $\prod_{i=m}^{n} x_i$ | Product of numbers $x_i$, where i is an integer $m \leq i \leq n$ |

# LIST OF FIGURES


Fig. 2.1 : Biological neuron ..... 4

Fig. 2.2 : Artificial neuron..... 5

Fig. 2.3 : Multilayer feedforward network ..... 7

Fig. 2.4 : Recurrent network ..... 7

Fig. 2.5 : Typical error surface with local minima (one-dimensional case) .....12

Fig. 2.6 : Generalization properties of a neural network .....13

Fig. 4.1 : Network architecture.....34

Fig. 4.2 : Time series of a process variable .....41

Fig. 4.3 : Architecture of Modified Logistic Network .....43

Fig. 4.4 : Architecture of Radial Basis Function Network .....44

Fig. 4.5 : Approximation and exact solution .....50

Fig. 4.6 : Approximation error .....51

Fig. 4.7 : Approximation and exact solution (2 samples for $\psi$) .....53

Fig. 4.8 : Approximation and exact solution (4 samples for $\psi$) .....53

Fig. 4.9 : Approximation error (4 samples for $\psi$).....54

Fig. 4.10 : Cart-pole system .....57

Fig. 5.1 : Evolution of the Lorenz system .....61

Fig. 5.2 : Evolution of the Lorenz system after $t=10^3$.....61

Fig. 5.3 : One-step-ahead forecasts and reference model ($t_0=0$).....65

Fig. 5.4 : Approximation error of one-step-ahead forecasts ($t_0=0$).....65

Fig. 5.5 : One-step-ahead forecasts and reference model ($t_0=10^3$).....66

Fig. 5.6 : Approximation error of one-step-ahead forecasts ($t_0=10^3$) ....66

Fig. 5.7 : Repeated forecasts and reference model ($t_0=0$) .....67

Fig. 5.8 : Approximation error of repeated forecasts ($t_0=0$).....68

Fig. 5.9 : Repeated forecasts of Modified Logistic Network ($t_0$=0) ..... 68
Fig. 5.10 : Approximation error of Modified Logistic Network ($t_0$=0) ..... 69
Fig. 5.11 : Repeated forecasts of Radial Basis Function Network ($t_0$=$10^3$) ..... 69
Fig. 5.12 : Approximation error of Radial Basis Function Network ($t_0$=$10^3$) ..... 70
Fig. 5.13 : Repeated forecasts of Modified Logistic Network ($t_0$=$10^3$) .... 70
Fig. 5.14 : Approximation error of Modified Logistic Network ($t_0$=$10^3$) . 71
Fig. 5.15 : Forecasting error of Radial Basis Function Network ..... 73
Fig. 5.16 : Convergence of learned parameters ..... 73
Fig. 5.17 : Forecasting error of Modified Logistic Network ..... 74
Fig. 5.18 : Convergence of learned parameters ..... 75
Fig. 5.19 : Evolution of the Rössler system ..... 76
Fig. 5.20 : One-step-ahead forecasts and reference model ..... 79
Fig. 5.21 : Approximation error of one-step-ahead forecasts ..... 79
Fig. 5.22 : Repeated forecasts and reference model ..... 80
Fig. 5.23 : Approximation error of repeated forecasts ..... 80
Fig. 5.24 : Forecasting error during parameter estimation ..... 81
Fig. 5.25 : Convergence of learned parameters ..... 82
Fig. 5.26 : Circular pendulum ..... 82
Fig. 5.27 : Evolution of the circular pendulum system ..... 85
Fig. 5.28 : One-step-ahead forecasts and reference model ..... 90
Fig. 5.29 : Approximation error of one-step-ahead forecasts ..... 91
Fig. 5.30 : Repeated forecasts and reference model ($\Delta t$=0.001) ..... 91
Fig. 5.31 : Approximation error of repeated forecasts ($\Delta t$=0.001) ..... 92
Fig. 5.32 : Repeated forecasts and reference model ($\Delta t$=$10^{-5}$) ..... 92
Fig. 5.33 : Approximation error of repeated forecasts ($\Delta t$=$10^{-5}$) ..... 93
Fig. 5.34 : Forecasting error during parameter estimation ..... 94
Fig. 5.35 : Convergence of learned parameters ..... 94

# LIST OF TABLES

Table 2.1 : Typical activation functions ..... 6
Table 2.2 : Typical transfer functions ..... 6
Table 4.1 : Coefficients $c_{k,i}$ .....27
Table 4.2 : Different radial basis functions.....29
Table 4.3 : Coefficients $b_i$ .....32
Table 4.4 : Coefficients $c_{k,i}$ .....33
Table 4.5 : Example of transformation rules .....36
Table 5.1 : Physical parameters of the pendulum .....85
Table 5.2 : Parameters of the circular pendulum system .....85

# CURRICULUM VITAE

| | |
|---|---|
| Last Name: | Cozzio-Büeler |
| First Names: | Enrico Albert |
| Nationality: | Swiss |
| Date of Birth: | July 6, 1965 |
| Marital Status: | Married |
| First Language: | German |
| Foreign Languages: | English, French |

1972-1978

Primary School: Primarschule Lachen

1978-1980

Secondary School: Sekundarschule Lachen

1980-1985

High School: Gymnasium Pfäffikon, Matura Type C

1985-1990

University: Studies in Computer Engineering at ETH, Zurich.
Diploma in Computer Engineering.
Title of diploma thesis: "Vergleich und Implementation von Unifikationsalgorithmen" (Comparison and Implementation of Unification Algorithms).
Supervisors: Prof. Dr. W. Guggenbühl, Prof. Dr. A. Kündig

1990-1995

Employed as Research Assistant in the Knowledge Systems Group at the ABB Corporate Research Laboratory, Baden-Dättwil, Switzerland.
Since 1991, Ph.D. work at ETH, Zurich.