Chaotic Disturbance Reduction
Using Neural Networks

by J.H.M. Bijen

M.Sc. Thesis
Carried out from October '95 to August '96
Author: J.H.M. Bijen
Eindhoven University of Technology
Department of Electrical Engineering
Measurement and Control Group
Under supervision of Prof. dr. ir. P.P.J. van den Bosch
Coached by Dr. ir. A.A.H. Damen

The Department of Electrical Engineering of the Eindhoven University of Technology accepts no responsibility for
the contents of M.Sc. Theses or reports on practical training periods.
It is always darkest just before the day dawneth
- Thomas Fuller
Abstract

This report describes the investigations done in the area of disturbance reduction. Particularly, the disturbance is considered chaotic. This type of process dynamics has become more and more topic of research in several scientific fields. The residuals formerly considered stochastic turn out to be essentially deterministic. Capturing of the chaotic dynamics enables the prediction of future values which can be used to design a proper control law to reduce the disturbance in the process output measurements.

The determination of the chaotic dynamics is done with the aid of a neural network. Single layer feedforward neural networks have proven to be powerful nonlinear identifiers but require extensive computational effort for optimization of the cost function. Several simulation are carried out using SIMULINK and MATLAB to show that disturbance reduction is possible.

The work has been carried out at the Measurement and Control Group of the Department of Electrical Engineering at the Eindhoven University of Technology in the period October '95 - August '96.
Acknowledgments

The completion of my work had not been possible without the help and support of many people. In the first place I wish to thank my parents for their support and confidence during the years at the University. I also wish to thank prof. dr. ir. P.P.J. van den Bosch for making it possible to start and successfully finish my assignment at the Measurement and Control Group. The conscientious and strict corrections of my coach dr. ir. A.A.H. Damen during the assignment and his ideas and conceptual approach have given me a lot of insight and have taught me how to deal with and resolve technical problems encountered.

Conversations with ir. Jozef Mazák and ir. Leon Ariaans have definitely made a great contribution in finishing my work as well as coffeebreaks with all fellow students at the Measurement and Control Group. I also wish to thank all my friends and everyone at the IEEE Student Branch Eindhoven for their support and friendship in and outside the University.
# Contents

1 Introduction

2 What is chaos?
   2.1 Introduction .............................................. 11
   2.2 Chaos versus randomness ................................. 11
   2.3 Sensitivity to initial conditions ...................... 12
   2.4 Conditions for chaotic behavior ...................... 13
      2.4.1 Continuous time systems .......................... 13
      2.4.2 Discrete time systems ............................. 15
   2.5 Attractors .................................................. 17
   2.6 Quantifying chaos ......................................... 18
      2.6.1 Lyapunov exponents ................................. 18
      2.6.2 Kolmogorov entropy ................................. 20
      2.6.3 Feigenbaum number .................................. 21
   2.7 Related issues ............................................. 22

3 Artificial neural networks
   3.1 Introduction ............................................... 23
   3.2 Model of a neuron .......................................... 23
   3.3 Benefits of a neural network ............................ 26
   3.4 Learning process .......................................... 27
      3.4.1 Backpropagation algorithm .......................... 28
      3.4.2 Optimization ......................................... 32
   3.5 Neurodynamics .............................................. 33
      3.5.1 Stability ............................................... 34
      3.5.2 Attractors ............................................. 34
      3.5.3 Strange attractors and chaos ...................... 34
   3.6 Application ................................................ 35
      3.6.1 Prediction of chaotic time series ................. 35
      3.6.2 System identification ............................... 36

4 Framework

5 Process identification
   5.1 Linear identification method ........................... 41
      5.1.1 Linear process ...................................... 42
      5.1.2 Nonlinear process .................................... 46
Chapter 1

Introduction

All processes deal with disturbances. These disturbances will affect the process output measurements which are often necessary to control the process. Output signals are compared to reference signals and deviations from the desired response are used to design a proper control signal.

In many cases the disturbances are considered as (white) noise additive to the system output. Figure 1.1 gives a schematic survey of this setup. Input disturbances propagate to the output and are added as such to the signal \( v(k) \) and consequently are also assumed to be additive to the output. The disturbance \( v(k) \) in figure 1.1 covers all disturbances thus, input disturbances, external influences and measurement noise due to non-ideal sensory equipment. The function \( G(q) \) denotes the process transfer where \( q \) is defined as a shift operator \( (qu(k) = u(k+1)) \).

![Figure 1.1: General overview of disturbed process. All disturbances are gathered in \( v(k) \).](image)

Most disturbances are considered as stochastic or random processes characterized by their statistical and stochastic properties. However, this assumption is in practice not always valid. Another type of disturbance is introduced that is considered chaotic. Properties, qualitative descriptions and related topics of chaos are treated in chapter 2. To show some examples from real life the following enumeration is given:

- Weather: Sewage systems and water purification plants have to deal with temperature fluctuations due to weather influence. Weather forecast can give detailed information about temperature, humidity and wind-force but only for a short period in future. Slow processes (or to be more precise their control) need more information about future weather development to behave in a desired way.
• Fluid flow: Under certain conditions fluid flow can be chaotic. Measurements of material purity, temperature or other properties can be disturbed by a persistently moving fluid flow that influences the sensory perception.

• Hartbeat: The rhythm of the heart behaves chaotic. Measurements during surgery of e.g. blood pressure or beat near the heart can be disturbed by the beating of the heart. In fact, a vast number of biological signals that lack apparent regularity are often thought to be completely stochastic. Recent investigations have revealed, however, that many biological signals, such as electroencephalograms and electrocardiograms, are, in fact, chaotic in nature.

Control of a process is usually carried out to produce an output with certain properties, qualitatively and/or quantitatively. Measurements of the output determine whether the control has to be adjusted. As seen from figure 1.1 the measurements are always disturbed. Different methods have been developed to reduce the influence of output disturbances. Output feedback and Kalman filtering are commonly used possibilities. But if the output is disturbed by a signal that is (partially) chaotic and it would be possible to quantitatively determine this chaotic disturbance, predictions could be made about the chaotic time evolution of the disturbance and take this into account by designing a control signal. A graphical representation of this idea is given in figure 1.2.

To predict the future value of a (chaotic) signal artificial neural networks will be used. These networks are able to learn different kinds of input-output relations [1]. Especially because most realistic processes turn out to be nonlinear and linear models are not always accurate enough to cope with these inherent nonlinearities. With the appropriate learning method and numerical optimization techniques neural networks turn out to be powerful prediction models. In figure 1.2 the process output is disturbed by a chaotic signal and measurement noise ($\eta(k)$). The residue (the measured output signal minus the model output),
called $\xi(k)$ in figure 1.2 is used to predict future disturbances. In an ideal situation this residue is exactly the chaotic disturbance but in general the model $(H_m)$ will deviate from the process $(H_p)$ and measurement noise can never be excluded in practical situations.

Another application for this scheme could be in a Model-Based Predictive Control (MPC) setup. This is a generic term for a group of related algorithms. An explicit dynamic model is used to predict the future values of the variable under examination. These future values are then calculated in such a way that they minimise a specified performance (cost function). Complex dynamics and nonlinearities can be coped with using this MPC setup. A predictive controller calculates - in contrary to the one step ahead predictors - a sequence of future controller outputs $u(k+1), u(k+2), \ldots, u(k+H_p)$ over a certain prediction horizon $H_p$. The control sequence is obtained by optimization of a cost function that could e.g. include a penalty term for the control signal. An example of such a cost function:

$$J(u) = \sum_{i=1}^{H_p} (\hat{y}(k+i) - d(k+i))^2 + \beta(u(k+i-1))^2$$  \hspace{1cm} (1.1)

with $\hat{y}$ being the predicted output, $d$ the desired process behavior and $u$ the control signal.

This report is divided into the following chapters: In chapter 2 chaotic systems will be highlighted along with some useful properties and examples. A brief overview of artificial neural networks and the training algorithms with some results of the prediction capabilities is given in chapter 3. In chapter 4 the setup of the problem is elaborated and more is said about the possibilities of how to tackle the problems expected. A short review about identification in chapter 5 followed by simulations and applications in chapters 6 and 7. Conclusions and recommendations about future investigations concerning this subject are given in chapter 8. A list of references concludes this report.
Chapter 2

What is chaos?

2.1 Introduction

Chaos is the term used to describe the apparently complex behavior of what we consider to be simple, well-behaved systems. Chaotic behavior looks erratic and almost random. This behavior however arises in very simple systems (those with only a few degrees of freedom), which are (almost) free of noise. In fact, these systems are essentially deterministic. That is, precise knowledge of the conditions of the system allow us to predict exactly the future behavior of that system. The problem of understanding chaos is to unite these apparently conflicting notions: randomness and determinism. The key element of this understanding is the notion of nonlinearity.

The reason why chaos has intrigued scientists is twofold. First, new conceptual and theoretical tools are provided enabling us to categorize and understand complex behavior and second, chaotic behavior seems universal - it shows up in mechanical, electrical, optical, chemical, and many other systems.

The importance of the system being nonlinear is concerned with the example of a parameter that describes a system. Changing the parameter in a linear system will result in a change in the system's frequency and amplitude but the qualitative nature of the behavior remains the same. As we will see, parameter changes in nonlinear systems can lead to sudden changes in both qualitative and quantitative behavior of the system. Some of these sudden changes in system behavior can give rise to complex behavior called chaos. The term chaos is used to describe the time behavior of a system when that behavior is aperiodic (it never exactly repeats) and is apparently random.

2.2 Chaos versus randomness

To divide nonlinear dynamics into categories, regular, periodic and hence determinable behavior and random or indeterminate behavior are the two categories in which the dynamics can be placed. However, a third category or kind of behavior can be distinguished which is called chaotic. We have deterministic rules of time evolution leading to behavior that, at first sight, looks indistinguishable from random behavior. The autocovariance of a signal determines the rate of coherence between successive values [2]. In case of a white noise signal, this coherence is not present, the autocovariance therefore yields a delta pulse at zero lag. If the
same procedure is carried out on a chaotic signal (the population model (2.4)) the resulting signal does not differ much from a delta pulse. Figure 2.1 shows the covariances of a random and a chaotic signal.

![Uniform distributed white noise](image1)

![Chaotic signal](image2)

![Autocovariance of white noise](image3)

![Autocovariance of chaotic signal](image4)

Figure 2.1: Covariance of random signal (left) and chaotic signal (right). The upper plots show the time evolutions of the noise and chaotic signals. The lower plots show the corresponding autocovariances.

There are methods that allow us to determine, at least approximately, the number of active degrees of freedom, and thus make a distinction between chaotic and random behavior, the latter being characterized as having an infinite (or at least very large) number of degrees of freedom. The number of degrees of freedom is defined as the number of independent variables needed to specify the dynamical state of the system.

With this information, a quantitative way of recognizing chaotic behavior would be desirable. In this way, the distinction between "real chaos" and noisy behavior due to e.g. random, external influences or erratic behavior due to complexity of the system can be made.

### 2.3 Sensitivity to initial conditions

One of the most important properties of a chaotic system is the sensitivity of the time evolution of the system to initial conditions. This makes the prediction of the future behavior of the system very difficult. The smallest perturbation in initial condition which is very likely to occur in systems with finite accuracy, can give rise to totally different long term behavior.
Usually, this is denoted as *divergence of nearby trajectories*. The notion of divergence of nearby trajectories is made formal by introducing the *Lyapunov exponent*. If two nearby trajectories start off with an initial separation of $d_0$ at time $t = 0$, then the trajectories diverge so that their separation at time $t$, denoted by $d(t)$, satisfies the expression:

$$d(t) = d_0 e^{\lambda t}$$  \hspace{1cm} (2.1)

The parameter $\lambda$ is called the Lyapunov exponent for the trajectories. If $\lambda$ is positive, the behavior is chaotic.

For systems with a $N$-dimensional state space, $N$ Lyapunov exponents can be calculated. In this case, at least one positive Lyapunov exponent must be present to assure chaotic behavior. In figure 2.2 (described in subsection 2.4.1) the divergence of two trajectories is shown. Starting off with only a small difference in initial conditions the trajectories start to diverge exponentially and the difference between the two trajectories grows with time.

![Sensitivity to initial conditions](image)

Figure 2.2: Divergence of nearby trajectories. The plot shows $z(t)$ of the Lorenz model for two trajectories with small difference in initial conditions

### 2.4 Conditions for chaotic behavior

#### 2.4.1 Continuous time systems

A condition for a dynamical system to behave in a chaotic way is that the trajectory in state space is never permitted to cross itself. In this situation only nodes or sinks can exist in a two dimensional space and chaotic behavior is therefore impossible. In three dimensions significant other behavior can be observed. The important thought is not that we live in a three
What is chaos?

dimensional world but that in three dimensional state space different types of trajectories can exist among which chaotic behavior. Intersection in a three dimensional space that represents the world coordinates does imply something completely different than an intersection in state space. In the last case, this would imply periodic behavior because the system is completely the same as a time $T$ before. In this case the following holds:

$$x(k) = x(k + nT) \quad n \in \mathbb{N} \quad (2.2)$$

An example of a continuous time system which can exhibit chaotic behavior is the Lorenz model. The Lorenz model was developed from the Navier-Stokes equation for fluid flow and the equation describing thermal energy diffusion. A good derivation from the Navier-Stokes equation to the Lorenz model can be found in Appendix C of [3]. A set of three nonlinear differential equations describe the Lorenz model. The variables in this model ($x$, $y$ and $z$) express the fluid motion and the resulting temperature differences. Variable $x$ is related to the fluid stream motion, $y$ is proportional to the temperature difference between the rising and falling parts of the fluid at a given height, while $z$ is proportional to the deviation from temperature linearity as a function of vertical position.

$$\begin{align*}
\dot{x} &= P(y - x), \\
\dot{y} &= Rx - y - xz, \\
\dot{z} &= xy - bz,
\end{align*} \quad (2.3)$$

where $P$ is analogous to the Prandtl number, $b$ is a spatial constant and $R$ is the Rayleigh number. $P$ is set to 10, $b = 8/3$ and $R$ is allowed to vary. As stated in section 2.3 the initial conditions determine the systems behavior to a large extent. In our case (2.3) these initial conditions are chosen as $x(0) = 1$, $y(0) = 0$ and $z(0) = 16$. As an example the time evolution of $x$ is plotted in figure 2.3. A state space representation is shown in figure 2.4. The Lorenz model (2.3) has been simulated using Simulink. A standard MATLAB function called lorenz.m shows a three dimensional animation of the Lorenz model.

![Figure 2.3: $x(t)$ in the Lorenz model](image)
2.4 Conditions for chaotic behavior

![X Y Plot](image)

Figure 2.4: y-z state space diagram of the Lorenz model

Note that the intersections in the state space diagram arise in a two dimensional projection of the three dimensional state space $(x, y, z)$. In three dimensions intersections do not appear.

2.4.2 Discrete time systems

The possibility of chaotic behavior can on the other hand easily be achieved in discrete time systems (also denoted as iterated maps). Successive values of the system can jump freely within its coordinate system. First order discrete time nonlinear systems can already behave in a chaotic way. As an example serves the simplified prey-predator system from here further denoted as the population model.

\[ x(k + 1) = rx(k)(1 - x(k)) \]  

(2.4)

The variable $r$ ($r > 0$) can be chosen freely and $x(0)$ is set to 0.45. In figure 2.5 the (discrete) time evolution of $x(k)$ is plotted. After a transient different kinds of behavior can be found. Different values of $r$ clearly result in different behavior of the system.
What is chaos?

For \( r = 2.75 \) a damped oscillation is observed. Increasing \( r \) to 3.1 leads to a periodic behavior which turns to a slightly more complicated periodic nature after increase of \( r \) to 3.5. The system has bifurcated (period doubling). A bifurcation is said to occur when the qualitative behavior of the system changes suddenly. In case \( r \) is increased to 4 seemingly random behavior shows up. This type of behavior is chaotic. Note that as was the case in the continuous time system the state can always be computed from past states.

A simple way to summarize the information is to generate a bifurcation diagram. In this diagram the system state (or function variable), in steady state, is plotted as a function of the control parameter. In the case of the population model (2.4) this results in a curve where the aforementioned system behaviors can be distinguished. In figure 2.6 the route to chaos is shown after several period-doublings. In section 2.6 some quantitative features of this diagram will be discussed.
2.5 Attractors

From this figure (2.6) the values of the control parameter $r$ where a bifurcation occurs can easily be distinguished and chaotic behavior starts at a value of $r$ at approximately 3.7.

2.5 Attractors

As seen from figure 2.2 different initial values can result in different trajectories. However, if the initial value results in a trajectory that leads to a value for which holds: $x^* = f(x^*)$, $x^*$ is called a fixed point. For the population model (2.4) it can be shown that for values of $r < 1$ all trajectories (with initial values $0 \leq x_{init} \leq 1$) approach the final value $x = 0$. The point $x = 0$ is called an attractor for those trajectories or orbits. The interval $0 \leq x_{init} \leq 1$ is called the basin of attraction for that attractor. To put this definition in more general terms, the attractor is called that set of points to which trajectories tend as the number of iterations goes to infinity. Again the population model serves as an example; for $r = 3.1$, the trajectory stabilizes after approximately 40 iterations at two points. The attractor points are 0.764567 and 0.558014. In the bifurcation diagram (fig. 2.6) this corresponds with the value of $r = 3.1$ where the diagram has two branches.

The feature of state space with three or more dimensions (subsection 2.4.2) that permits chaotic behavior is that the trajectories can remain within some bounded region wrapping around each other without intersecting and thus creating some odd shaped figure that is
called a *strange attractor*. Figure 2.4 gives an idea of such a strange attractor (projected onto two dimensions).

### 2.6 Quantifying chaos

Several reasons exist why chaos should be quantified. As seen before chaos and randomness look similar. A definitive, quantitative way of recognizing chaos and sorting out noisy or erratic behavior is desirable. Second, the quantifiers may help to determine how many variables are needed to model the dynamics of the system [3]. Another problem that could be dealt with is the capability of prediction of for example a chaotic system or a chaotic disturbed system output. Is the prediction capability or the prediction horizon related to the quantitative measure of chaos? Finally, if it would be possible to link the quantifier to the system behavior, predictions could be made about a system initiating chaotic behavior. Different quantifiers are commonly used to quantify chaos:

- Lyapunov exponents
- Kolmogorov entropy
- Feigenbaum number

#### 2.6.1 Lyapunov exponents

As mentioned in section 2.3 one of the signatures of chaos is the divergence of nearby trajectories and was indicated exponential. However, a chaotic trajectory returns infinitely often (for a bounded system) very close to any previous point on the trajectory. Thus, the signature of divergence could be transformed to the divergence of nearby points (corresponding to different times) on a single trajectory. Two points are considered: a point \( x_0 \) and a neighbouring point \( x_0 + \epsilon \). When the iterated map is applied \( n \) times to each value (the absolute value of) the difference between the two results is:

\[
d_n = | f^{(n)}(x_0 + \epsilon) - f^{(n)}(x_0) |
\]

If the behavior is chaotic, this difference should grow exponentially with \( n \), so:

\[
\frac{d_n}{\epsilon} = \frac{| f^{(n)}(x_0 + \epsilon) - f^{(n)}(x_0) |}{\epsilon} \equiv e^{\lambda n}
\]

or

\[
\lambda = \frac{1}{n} \ln \left( \frac{| f^{(n)}(x_0 + \epsilon) - f^{(n)}(x_0) |}{\epsilon} \right) \quad (2.7)
\]

If \( \epsilon \to 0 \) the right-hand side of (2.7) equals the definition of the absolute value of the derivative of \( f^{(n)}(x) \) with respect to \( x \). When the chain rule of differentiation is applied, the derivative of \( f^{(n)} \) can be written as a product of \( n \) derivatives of \( f(x) \) evaluated at the successive trajectory points \( x_0, x_1, \ldots, x_n \). The definition of \( \lambda \) can be rewritten to:

\[
\lambda = \frac{1}{n} \ln (| f'(x_0) || f'(x_1) | \cdots | f'(x_n) |)
\]
or

$$\lambda = \frac{1}{n}(\ln | f'(x_0) | + \ln | f'(x_1) | + \cdots + \ln | f'(x_n) |)$$

(2.9)

In other words, if the application of the map function to two points leads to two points further apart, then the absolute value of the derivative of the map function is greater than 1 and the logarithm is positive. From these statements a quantitative definition of chaotic behavior is that an iterated map function has chaotic trajectories for a particular parameter value if the Lyapunov exponent is positive for that parameter value.

In state spaces with more than three dimensions and two positive Lyapunov exponents the behavior is denoted as hyperchaos. In figure 2.7 the Lyapunov exponent is drawn for the population model. To make an easy comparison with the population the bifurcation diagram is also drawn. Parameter values greater than approximately 3.57 correspond to chaotic behavior.

Figure 2.7: Bifurcation diagram and Lyapunov exponent of population model as function of parameter $r$
2.6.2 Kolmogorov entropy

From the point of view of statistical mechanics the concept of entropy is given by counting the number of accessible states for a system. Macroscopic properties of a system (e.g. pressure, temperature) are related to the microscopic description of atoms and molecules of for example a gas. A vast number of states can be defined that correspond to positions and velocities of the atoms or molecules. The entropy $S$ of a system in thermal equilibrium is defined as

$$S = k \ln N$$

(2.10)

where $N$ is the number of states (on microscopic scale) and $k$ is Boltzmann’s constant. The second law of thermodynamics states that the entropy in a spontaneous thermal process of an isolated system will either stay the same or increase and thus tend to evolve toward a set of conditions that has the largest number of accessible states. In case of equally-likely probabilities of the states (i.e. $p_i = 1/N$) the entropy becomes

$$S = -k \ln p$$

(2.11)

If the probabilities are not equal we can generalize (2.11) to

$$S = -k \sum_r p_r \ln p_r$$

(2.12)

where $p_r$ is the probability of the $r^{th}$ state. If a state is not accessible ($p_r = 0$) the state makes no contribution to the entropy.

In many systems the counting of the states can be difficult. In state space systems only the region occupied by the attractor has to be divided into cells. Starting with a set of initial conditions the entropy of the system after $n$ units of time ($S_n$) is defined by

$$S_n = -k \sum_r p_r \ln p_r$$

(2.13)

If the number of initial trajectory points is $M$ and after $n$ units of time the number of points in cell $r$ is $M_r$ the probability is defined as $p_r = M_r/M$ and can be seen as the fraction of trajectory points that end up in the $r^{th}$ cell at that time. An example is that all trajectories move together from cell to cell as time goes on. Then, $p_r = 1$ for the occupied cells and $p_r = 0$ for the unoccupied cells. In this case $S_n = 0$ for all $n$. A constant entropy can therefore be assigned to a regular motion. When the number of cells $N_n$ increases as time goes on (and assumed the probability is equal for all occupied cells namely $1/N_n$) the entropy is

$$S_n = -k \ln \frac{1}{N_n} = k \ln N_n$$

(2.14)

For a random system each trajectory point would jump to its own cell and in that case the entropy would be $S_n = \ln M$. For large values of $M$ this would increase without limit. This property results in viewing the entropy not as a particular value but rather the change in entropy as time evolves. This change in entropy is called the Kolmogorov-entropy and is defined by

$$K_n = \frac{1}{\tau} (S_{n+1} - S_n)$$

(2.15)
2.6 Quantifying chaos

and is the rate of change of entropy between the time values \( t = n\tau \) and \( t = (n + 1)\tau \). To characterize the whole attractor the average of the K-entropy has to be determined

\[
K = \lim_{N \to \infty} \frac{1}{N\tau} \sum_{n=0}^{N-1} (S_{n+1} - S_n)
\]

(2.16)

of which only the first and the last value of the sum remain leading to

\[
\lim_{N \to \infty} \frac{1}{N\tau} (S_N - S_0)
\]

(2.17)

Now two refinements are introduced. The first divides the state space into infinite small cells (cell length \( L \to 0 \)), the second reduces the time interval \( \tau \) to zero. In discrete time systems the value of \( \tau \) is set to 1 which cancels the limit of \( \tau \to 0 \) in the definition of the K-entropy:

\[
K = -\lim_{\tau \to 0} \lim_{L \to 0} \lim_{N \to \infty} \frac{1}{N\tau} (S_N - S_0)
\]

(2.18)

When the number of cells occupied increases exponentially with time and the cells have the same probability, the K-entropy equals the Lyapunov exponent.

2.6.3 Feigenbaum number

When looking at the bifurcation diagram for iterated maps (e.g. fig. 2.6) the parameter values at which the successive period-doublings occur seem to possess some geometric ratio. If the ratio is calculated between the splitting values \( A_n \) at which the sequence bifurcates

\[
\delta_n = \frac{A_n - A_{n-1}}{A_{n+1} - A_n}
\]

(2.19)

it can be found that the value of \( \delta \) is almost the same for all values of \( n \). More surprisingly however was that this value approached a number (for large values of \( n \)) that was the same for other map functions\(^1\) (e.g. \( x(k+1) = A\sin(\pi x(k)) \)).

This value of \( \delta \) has been named the "Feigenbaum delta" after its discoverer Mitchell Feigenbaum.

\[
\delta \equiv \lim_{n \to \infty} \delta_n = 4.66920161 \ldots
\]

(2.20)

The existence of a universal number such as \( \delta \) allows us to make quantitative predictions about the behavior of a nonlinear system. If a system is observed that undergoes a period-doubling bifurcation from period-1 to period-2 at a parameter value \( A_1 \), and from period-2 to period-4 at \( A_2 \), then \( \delta \) can be used to predict that the system will make a transition from period-4 to period-8 at a value of \( A \) given by

\[
A_3 = \frac{A_2 - A_1}{\delta} + A_2
\]

(2.21)

Note here that this equation only applies if a successive bifurcation occurs.

In this way an iterative method can be applied to calculate the successive values at which

\(^1\)This value of the Feigenbaum delta apply only to those one-dimensional iterated maps that have quadratic behavior near their maximum values
the system shows period-doublings and when the system initiates chaotic behavior. The expression for \( A_4 \) is given by

\[
A_4 = \frac{A_3 - A_2}{\delta} + A_3
= (A_2 - A_1) \left( \frac{1}{\delta} + \frac{1}{\delta^2} \right) + A_2
\]

Continuing this procedure for higher values of \( A \) yields a geometric series of powers of \( \delta \) and results in

\[
A_\infty = (A_2 - A_1) \frac{1}{\delta - 1} + A_2
\]

Because the values of \( A_1 \) and \( A_2 \) are obtained experimentally and the value of \( \delta_n \) is considered constant for all values of \( n \) the prediction will not be exact. In practice however, the prediction of \( A_\infty \) will be reasonably close to the value at which chaotic behavior starts. If the prediction is applied to the population model (2.4) and the matching bifurcation diagram (Fig. 2.6) we obtain \( A_1 \approx 2.95 \) and \( A_2 \approx 3.49 \) which leads to \( A_\infty \approx 3.64 \). Comparing this to the point at which chaotic behavior starts in figure 2.6 the computation gives a reasonable indication for a system initiating chaotic behavior.

2.7 Related issues

Chaos is only a part of the behavior of complex systems. It turns out that a counterpart of chaos also exists which might be called antichaos. Some very disordered systems spontaneously turn towards a high degree of order. Experiments and computer modeling done with complex genetic structures suggest that these systems in some way exhibit self-regularization of gene activity. It is believed, that this self-organisation is an important part in the biological development and evolution [4].

Chaotic behavior can be used in communication systems. If the transmitted signal is modulated on a chaotic carrier and therefore almost impossible to decrypt an easy way of encrypting information is obtained. The receiver however, must know the parameters of the system which encrypts the signal. A way to reach this goal is to introduce a master/slave configuration in which the system is decomposed into two separate parts.
Chapter 3

Artificial neural networks

3.1 Introduction

The brain is a highly complex, nonlinear and parallel computer [5]. It has the capability of organising neurons (seen as structural constituents of the brain) to perform certain computations faster than a digital computer. To act as a brain, networks are made up of artificial neurons. These neural networks are usually implemented using electronic components or simulated in software to form a machine designed to model the way in which a brain performs a particular task. Neural networks are also referred to as neurocomputers or connectionist networks. They resemble the brain in two ways:

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge. Synapsis mediate the interactions between neurons.

3.2 Model of a neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network. Figure 3.1 shows a model for a neuron. Three basic elements may be distinguished:

- A set of synapses or connecting links, each characterized by its own weight or strength. A signal \( x_j \) at the input of synapse \( j \) connected to neuron \( k \) is multiplied by the synaptic weight \( w_{kj} \).
- An adder for summing the weighted input signals.
- An activation function for limiting the output amplitude of a neuron. It limits the permissible amplitude range of the output signal to some finite value.

The model of the neuron shown in figure 3.1 also includes an externally applied threshold \( \theta_k \) that has the effect of lowering the net input of the activation function. Thus:

\[
\theta_k = w_{k0}x_0
\]
A neuron may than be described by the following pair of equations:

\[ v_k = \sum_{j=0}^{p} w_{kj} x_j \]  \hspace{1cm} (3.2)

and

\[ y_k = \varphi(v_k) \] \hspace{1cm} (3.3)

Three basic activation function are often used:

- Threshold function

\[ \varphi(v) = \begin{cases} 
1, & v \geq 0 \\
0, & v < 0
\end{cases} \]  \hspace{1cm} (3.4)

- Piecewise-linear function

\[ \varphi(v) = \begin{cases} 
1, & v \geq \frac{1}{2} \\
v, & -\frac{1}{2} < v < \frac{1}{2} \\
0, & v \leq \frac{1}{2}
\end{cases} \] \hspace{1cm} (3.5)

- Sigmoid function

\[ \varphi(v) = \frac{1}{1 + \exp(-av)} \] \hspace{1cm} (3.6)
3.2 Model of a neuron

The sigmoidal activation function is most commonly used. It is defined as a strictly increasing function that exhibits smoothness and asymptotic properties. The parameter $a$ can be varied to obtain sigmoid functions of different slopes. It is often desirable to define the activation functions anti-symmetric with respect to the origin, i.e. ranging from $-1$ to $1$ instead of $0$ to $1$. In this case the activation functions (3.4, 3.5 and 3.6) must be slightly redefined. In figure 3.2 these functions are plotted anti-symmetric.

When several neurons are connected with each other a neural network arises. Usually the network is formed by an input layer, an output layer and between these two one or more hidden layers. In case the inputs of a layer only consist of outputs of the previous layer this neural network is denoted as a feedforward network. If output signals are fed back to the input the network becomes recurrent. More about this type of dynamical behavior is given in section 3.5. In figure 3.3 a network consisting of an input layer, an output layer and two hidden layers with several hidden nodes is shown. The input layer serves nothing more than distribution of the input signals. Between the layers, the weights are positioned making the connection between consecutive layers. The output layer is usually chosen to be linear. This type of network is called feedforward, networks that possess nodes fed by signals of "outputs" in the next layer are called recurrent.
3.3 Benefits of a neural network

The power of a neural net is derived from the parallel distributed structure and the ability to learn and therefore generalize; to produce reasonable outputs for inputs not encountered during training. These capabilities make it possible for a neural net to solve complex problems. Drawback is the fact that neural nets cannot provide the solution alone. They need to be integrated into a consistent system and neural nets are assigned a specific (relatively simple) task that matches their capabilities. Useful properties of neural nets are the following [5]:

- **Nonlinearity.** A neuron is a nonlinear device. Consequently, a neural network is itself nonlinear.

- **Input-output Mapping.** In supervised learning the synaptic weights of a neural network are modified by applying a set of task examples. The network is presented an example and the parameters of the network are modified so as to minimize the difference between the input signal and the corresponding desired response. The applied training examples may be reapplied in a different order. The network learns by constructing an input-output mapping for the problem at hand.

- **Adaptivity.** Neural networks have a capability to adapt their synaptic weights to changes in the surrounding environment. When it is operating in a nonstationary environment (i.e. whose statistics change with time), a neural network can be designed to change its synaptic weights in real time.

- **Fault Tolerance.** A neural network has the potential to act fault tolerant. If a neuron or connecting link is damaged, the distributed nature of information in the network results in a degradation of network performance rather than a catastrophic failure.
3.4 Learning process

The property of a neural network that is of primary significance is the ability of the network to learn from its environment, and to improve its performance through learning. A neural network learns about its environment through an iterative process of adjustments applied to its synaptic weights and thresholds. Learning in the context of neural networks is defined as follows:

Definition 3.1 Learning is a process by which the free parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place [5].

This definition implies the following sequence of events:

- The neural network is stimulated by an environment.
- The neural network undergoes changes as a result of this stimulation.
- The neural network responds in a new way to the environment, because of the changes that have occurred in its internal structure.

The oldest and most famous learning rules are named after Hebb's postulate of learning and are called Hebbian learning. His proposed learning rules would result in an enduring modification in the activity pattern of an assembly of cells.

The main goal of the learning process is that after this training or learning the network can be used as a model for estimating the future values of the system outputs or states. Different ways of learning are available. The method described here uses batch training, i.e. the patterns or time series to be learned are presented consecutively to the network after which the update of the weights is carried out. The objective of the training consists of minimizing a cost function represented by a sum of squared errors. This is elucidated by the following example. Let $f$ in the nonlinear equation $x(k+1) = f(x(k)) = rx(k)[1-x(k)]$. The network for example can be chosen to have one input (i.e. $x(k)$) and one output (i.e. $x(k+1)$). The network is presented a sequence of $N$ samples of this time series (e.g. $N = 1000$). Thus at each iteration the network is presented $N$ samples of $x(k)$ at the input and $N$ samples of the corresponding $x(k+1)$ at the output. At each epoch\(^1\) the output of the network is compared with the desired output (being $x(k+1)$). In this example the network is trained to predict the future value of the system.

The network is made up of one input, one output and one hidden layer with a different number of nodes. Each configuration is optimized, the best is chosen as network which will be

\(^1\)An epoch is considered a single computation and update of the network (internal) signals

- VLSI Implementability. The parallel nature of the neural network makes fast computations possible. The virtue of VLSI is that it provides hierarchical capturing of highly complex behavior which makes it possible to use a neural network for real time applications.
used in further computations. The optimization is in a first approach done with a stochastic method (Simulated Annealing), followed by a deterministic method (Quasi-Newton). These methods will not be explained in this report, but a vast number of books can be found that describe these methods.

In practice the computed output will not be the same as the desired output. Every epoch the error between the real and the desired output is added (squared) to the sum. The setup of this idea is given in figure 3.4.

Comparison of the real output with the approximated output results in the approximation error $e(k + 1)$ that is used to adapt the weights in the network so as to minimize the error and let the network function $F_{NN}$ converge to the real system function $F$. Note here that the estimation is a prediction error method. The real output ($x(k + 1)$) is fed back to both the real process as well as the neural network. Another identification method has been used throughout this report, the Output Error method. Adaption of the setup may change the identification of $F$.

3.4.1 Backpropagation algorithm

The backpropagation algorithm is the most widely used and successful algorithm to train a neural net. It consists of two phases: The forward and backward phase. During the forward phase input signals propagate through the network layer by layer, eventually producing some output of the network. The parameters of the network are fixed during this phase. The response of the network is then compared with a desired response, generating error signals that are then propagated in a backward direction through the network. The free parameters are adjusted so as to minimize the sum of squared errors. Drawback of this algorithm is the poor scaling behavior: an increase of the size of the network results in an exponential growth of the computation time.
3.4 Learning process

MLPs

MultiLayer Perceptrons (MLPs) are successful in solving difficult problems by training them in a supervised manner with the error-backpropagation algorithm based on the error correction learning rule. Properties of the MLP are:

- The model of a neuron includes a smooth nonlinearity at the output end (i.e. it is differentiable everywhere).

- Hidden neurons enable the network to learn complex tasks.

- MLPs possess a high degree of connectivity.

The nonlinearity and high connectivity make a theoretical analysis very difficult and it cannot be claimed that the backpropagation algorithm can provide a solution for all solvable problems. Figure 3.5 shows different signals and nodes associated with a neural network.

Circles in figure 3.5 represent the nodes. Lines that connect the nodes are associated with the weights. At the right side of the figure the computed output value of the network is subtracted from the desired value \(d_k(n)\) yielding the estimation error \(e_k(n)\).

**Derivation of the algorithm**

The net internal activity at the input of neuron \(j\) is

\[
v_j(n) = \sum_{i=0}^{p} w_{ji}(n)y_i(n)
\]  

(3.7)

where \(p\) is the number of inputs of neuron \(j\). The function signal at the output of neuron \(j\) at iteration \(n\) is

\[
y_j(n) = \phi_j(v_j(n))
\]  

(3.8)
The correction of the weight ($\Delta w_{ji}(n)$) is proportional to the instantaneous gradient

$$\frac{\partial \varepsilon}{\partial w_{ji}(n)}$$

It represents a sensitivity factor determining the direction of search in weight space. According to the chain rule this can be written as

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

(3.10)

After differentiating the following function this results in:

$$\varepsilon(n) = \frac{1}{2} \sum_j e_j^2(n) \Rightarrow \frac{\partial \varepsilon(n)}{\partial e_j(n)} = e_j(n)$$

$$e_j(n) = d_j(y_j(n)) \Rightarrow \frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

$$y_j(n) = \phi'_j(v_j(n)) \Rightarrow \frac{\partial y_j(n)}{\partial v_j(n)} = \phi'_j(v_j(n))$$

$$v_j(n) = \sum_{i=0}^\rho w_{ji}(n)y_i(n) \Rightarrow \frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

(3.11)

Resulting in

$$\frac{\partial \varepsilon}{\partial w_{ji}(n)} = -e_j(n)\phi'_j(v_j(n))y_i(n)$$

(3.12)

With the correction of the weight as

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$$

(3.13)

this leads to ($\eta \equiv$ learning-rate parameter)

$$\Delta w_{ji}(n) = \eta \phi'_j(v_j(n))y_i(n)$$

(3.14)

with local gradient

$$\delta_j(n) = e_j(n)\phi'_j(v_j(n))$$

(3.15)

Two cases can now be distinguished:

- **Neuron $j$ is an output node**
  Each output node is supplied with a desired response of its own, making it straightforward to calculate the associated error signal.

- **Neuron $j$ is a hidden node**
  When neuron $j$ is hidden, there is no specified desired response for that neuron. It should be determined recursively in terms of the error signals of all the neurons to which the hidden neuron is directly connected.

- **Neuron $j$ is an input node**
  If the output of the network is fed back to the input the network becomes recurrent and the backpropagation algorithm dynamic.
The local gradient for hidden neuron $j$ is computed as
\[ \delta_j(n) = -\frac{\partial \epsilon(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\epsilon(n)}{\partial y_j(n)} \cdot \phi_j'(v_j(n)) \] (3.16)

To calculate the partial derivative $\partial \epsilon(n)/\partial y_j(n)$ use made of the expression $\epsilon(n) = \frac{1}{2} \sum_k e_k^2(n)$ where neuron $k$ is an output node and neuron $j$ a hidden node. By differentiating $\epsilon(n)$ with respect to $y_j(n)$ we obtain
\[ \frac{\partial \epsilon(n)}{\partial y_j(n)} = \sum_k e_k \cdot \frac{\partial e_k(n)}{\partial y_j(n)} \] (3.17)

and then applying the chain rule yields
\[ \frac{\partial \epsilon(n)}{\partial y_j(n)} = \sum_k e_k \cdot \frac{\partial e_k(n)}{\partial v_k(n)} \cdot \frac{\partial v_k(n)}{\partial y_j(n)} \] (3.18)

and
\[ e_k(n) = d_k(n) - y_k(n) \]
\[ = d_k(n) - \phi_k(v_k(n)) \] (3.19)

where $k \equiv$ output node. Hence
\[ \frac{\partial \epsilon_k(n)}{\partial v_k(n)} = -\phi_k'(v_k(n)) \cdot v_k(n) = \sum_{j=0}^q w_{kj}(n)y_j(n) \] (3.20)

where $q$ represents the number of inputs of neuron $k$ (excluding the threshold). Differentiating $v_k(n)$ yields
\[ \frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \] (3.21)

which leads to the expression
\[ \frac{\partial \epsilon(n)}{\partial y_j(n)} = -\sum_k e_k(n)\phi_k'(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n) \] (3.22)

Thus, the local gradient $\delta_j(n)$ for hidden neuron $j$ is
\[ \delta_j(n) = \phi_j'(v_j(n)) \sum_k \delta_k(n)w_{kj}(n) \] (3.23)

in which the computation of $\delta_k(n)$ requires knowledge of the error $e_k(n)$.

The correction $\Delta w_{ji}(n)$ applied to synaptic weight connecting neuron $i$ to neuron $j$ is defined by
\[
\begin{pmatrix}
\text{weight correction} \\
\Delta w_{ji}(n)
\end{pmatrix} = \begin{pmatrix}
\text{learning rate} \\
\eta
\end{pmatrix} \cdot \begin{pmatrix}
\text{local gradient} \\
\delta_j(n)
\end{pmatrix} \cdot \begin{pmatrix}
\text{input signal of neuron} \ j \\
y_i(n)
\end{pmatrix} \] (3.24)

Note that the value of the local gradient depends on whether neuron $j$ is an output or a hidden node.

- $j \equiv$ output node:
  \[ \delta_j(n) = \phi_j'(v_j(n)) \cdot e_j(n) \] (Both associated with neuron $j$)

- $j \equiv$ hidden node:
  \[ \delta_j(n) = \phi_j'(v_j(n)) \cdot \sum_k \delta_k(n)w_{kj}(n) \] where the sum stands for a weighted sum of $\delta$'s for neurons in the next hidden layer or output layer that are connected to neuron $j$. 

Recurrent backpropagation

If the network is permitted to have feedback between the neurons the network becomes recurrent.

3.4.2 Optimization

Although the backpropagation algorithm is easy to use and capable of finding a minimum value of the cost function, the algorithm will generally get stuck in a local minimum. Optimization is required and formalized in the following way. Let the optimization problem be the pair \((\mathcal{D}, \mathcal{J})\) where \(\mathcal{D}\) is the set of configurations and \(\mathcal{J}\) the cost function, which assigns a real number to each configuration. Assumed is that the lower value of \(\mathcal{J}\), the better the corresponding configuration. The problem is to find a configuration for which \(\mathcal{J}\) takes its minimum value:

\[
\mathcal{J}_{\text{opt}} = \mathcal{J}(\theta^*)
\]

where

\[
\theta^* = \arg\min_{\theta \in \mathcal{D}} \mathcal{J}(\theta)
\]

\(\mathcal{J}_{\text{opt}}\) denotes the optimum (minimum) cost. Applying this to an artificial neural network, the cost could be represented by the difference between the real process output and the neural network output that is to model the real process. The configuration set \((\mathcal{D}\) is the set of values of the weights in the network (and possible the network architecture, i.e. the number of layers, inputs and outputs and the number of nodes per layer).

Two possible approaches exist while trying to solve an optimization problem:

- An optimization algorithm, yielding a globally optimal solution in possibly an unacceptable amount of computation time.
- An approximation algorithm, yielding an approximate solution in a reasonable amount of time.

The simulated annealing algorithm can be seen as a general approximation algorithm. The algorithm is based on randomization techniques and also incorporates a number of aspects related to iterative improvement algorithms. The application of iterative improvement algorithms presupposes the definition of a configuration, a cost function and a simple description to generate a transition from one configuration to another configuration by a small perturbation. The set of configurations that can be reached in one transition is called a neighbourhood. Starting off with a given configuration (e.g. a randomly chosen set of weights), a sequence of iterations is generated each consisting of a configuration selected from the neighbourhood. If the neighbourhood configuration has a lower cost it replaces the current configuration. The algorithm terminates when a configuration is obtained whose cost is no worse than any of its neighbours. Disadvantages of this technique are:

- The algorithm terminates in a local minimum. Generally there is no information by which this local minimum deviates from the global minimum.
The obtained local minimum depends on the initial configuration, for the choice of which generally no guidelines are available.

Generally, no upper bound for the computation time can be given. Despite these disadvantages, the properties of the iterative improvement algorithm (configuration set, cost function and generation mechanism) are easy to define and a single run (for one initial configuration) can usually be executed in a small amount of time and a few additions can improve the algorithm. These additions involve execution of the algorithm for a large number of initial conditions (which increases both the computation time and the probability of finding a global minimum), more complex generation mechanisms that enable the algorithm to "jump out" of the local minima (see figure 3.6 and acceptance of (a limited) increase of the cost function.

Figure 3.6: Convex and non-unimodal cost function. Optimization routines can get stuck in a local minimum in the non-unimodal cost function.

The last option (increasing cost acceptance) is generally known as simulated annealing. Solutions are close to the minimum cost but the computational effort can be quite extensive. When optimizing a complex system such as a neural network with a large number of weights the optimization routine is instead of going downhill always, going downhill most of the time. The main advantage of the simulated annealing algorithm is that it need not get stuck in a local minimum.

### 3.5 Neurodynamics

The nonlinear behavior of artificial neural nets is also denoted as neurodynamics. An artificial neural net needs therefore mathematical modeling to describe the dynamics of this nonlinear system. A suitable model is the state-space model. When the variables \(x_1(t), x_2(t), \ldots, x_N(t)\) denote the state variables of the system which contain information to predict the future evolution of the system, \(F\) is a nonlinear function that operates on each element of the state vector:

\[
x(k) = [x_1(k), x_2(k), \ldots, x_N(k)]^T
\]  \(3.27\)
and as control or input vector
\[ u(k) = \begin{bmatrix} u_1(k), u_2(k), \ldots, u_M(k) \end{bmatrix}^T \]
we get
\[ x(k + 1) = F[x(k), u(k)] \tag{3.29} \]

One way of training a neural net is to model its dynamics in state-space. The state value of
the system is subtracted from the state value of the real process and the resulting error is to
be minimized. This setup is shown in figure 3.4. As seen in this figure the output value of the
network is used to calculate future output values and makes the network recurrent. Recurrent
artificial neural networks use supervised learning algorithms such as recurrent backpropaga-
tion to simulate a dynamic system without knowledge of the equations governing the system.
A drawback of this supervised learning is the computational effort required and the time con-
suming optimization while no guarantee of success can be given. It is mainly for this reason,
that these recurrent neural networks are not yet integrated as an engineering tool.

### 3.5.1 Stability

Some neural network models are time dependent. Either the time dependence is added to the
model or it is built into the model’s operation. In both cases a nonlinear dynamical system
is under consideration. This subject, with emphasis on the stability problem is referred to
as *neurodynamics*. When speaking of stability, *stability in the sense of Lyapunov* \(^2\) is meant.
The BIBO stability criterion is well suited for linear dynamical systems but useless to be
applied to neural networks because all neural networks are BIBO stable due to the saturating
nonlinearity built into a neuron.

However, one of the properties of chaotic systems is the divergence of nearby trajectories,
mentioned in subsection 2.6.1. A chaotic system is considered unstable, in this sense. Poincaré
therefore introduced the concept of *orbital stability* [6]. Let \( \Gamma \) be the orbit defined by \( u(t) \)
and \( \Gamma' \) the orbit defined by \( v(t) \). Orbital stability (for \( \Gamma \)) is obtained if, for any \( \epsilon > 0 \), there
exists a \( \delta(\epsilon) > 0 \) such that, if \( |u(0) - v(\tau)| < \delta(\epsilon) \) for some \( \tau \), then there exists a \( t'(t) \)
such that \( |u(t) - v(t')| < \epsilon \) for all \( t > 0 \). This implies that the two solutions will experience the
same history but possibly on different time scales, related by \( t'(t) \).

### 3.5.2 Attractors

Experiments with dynamical systems \( (x(k + 1) = F[x(k)] \) with \( x \in \mathbb{R}^m \) usually exhibit
transient behavior followed by a motion on an attractor in state space, namely a subset of \( \mathbb{R}^m \)
on which the orbits converge. The complexity of motion can be classified at different levels
by looking at the geometric structure of the attractor. Examples of these levels may be fixed
points, limit cycles (periodic attractors) and strange attractors.

### 3.5.3 Strange attractors and chaos

A system is said to possess a strange attractor when two orbits with neighbouring initial
conditions tend to move away from each other with increasing time; the sensitivity to initial

\(^2\)This method is also known as the *direct method of Lyapunov* or *second method of Lyapunov*.
conditions (See sect. 2.3). Chaos plays an important role in neurobiology. It is therefore not surprising that the relation between chaos and neural nets gets more and more attention. Experiments already have shown that chaotic behavior arises in different levels of the brain.

3.6 Application

3.6.1 Prediction of chaotic time series

In chapter 1 the neural network was introduced to predict future values of the (chaotic) disturbance. Several authors ([7], [8] and [9]) have shown that prediction of chaotic time series is possible using neural networks. In this section some results of the predicting capabilities will be shown. First the one-step ahead prediction is treated followed by the multi-step prediction both applied to the population model (2.4).

The network must be able to learn the chaotic dynamics of the population model. In the situation of learning the one-step ahead prediction the network is shown \( x(k) \) at the input and \( x(k+1) \) at the output. If the transfer from input to output is drawn the parabolic curve of the population model will appear. The neural network therefore has to approximate this curve as closely as possible. The network (one hidden layer with four nodes) is trained with a dataset of 1000 samples, the validation set also contained 1000 samples.

Networks could also be designed to predict more than one future output. In fact, a vast number of configurations could be thought of. An example is a layout of one input and several future outputs, i.e. \( x(k) \) as input and \( x(k+1) \ldots x(k+n) \) as successive outputs. Also more inputs can be used. Complex nonlinear functions of higher order will usually require neural networks with a large number of inputs, outputs, hidden layers and nodes. Prediction of two future outputs with only one input value is done with a network consisting of one hidden layer with 7 nodes. Two different transfer functions have to be approximated in this case:

\[
\begin{align*}
x(k + 1) &= rx(k)[1 - x(k)] \\
x(k + 2) &= rx(k + 1)[1 - x(k + 1)] \\
&= r^2 x(k)[1 - x(k)](1 - rx(k)[1 - x(k)])
\end{align*}
\] (3.30)

(3.31)

Figure 3.7: Transfer functions from \( x(k) \) to \( x(k+1) \) (left plot) and \( x(k) \) to \( x(k+2) \) (right plot). Thin dots refer to the real transfer function, thick dots to the approximated transfer function.
Figure 3.7 shows these functions and the approximations done by the neural network. Gaps in the curves are caused by the limited size of the validation set. This validation set was chosen to be of length 1000 so that it covers quite a vast number of possible states. Gaps in the curve correspond to states that have not (yet) been accessed.

However, the setup used is not necessary. The population model is of first order and the prediction of more than one future output can be done by cascading several networks as shown in figure 4.4. The function to be approximated \( F \) can be considered as:

\[
x(k+1) = F[x(k)]
\]

and is therefore a nonlinear function of first order. In case the neural network is capable of predicting \( x(k+1) \) based on \( x(k) \) then cascading networks (and functions \( F \)) lead to prediction of \( x(k+2), x(k+3), \ldots, x(k+p) \) depending on the number of cascaded networks.

\[
x(k+1) = F[x(k)]
\]
\[
x(k+2) = F^2[x(k)]
\]
\[
\vdots
\]
\[
x(k+p) = F^p[x(k)]
\]

Inaccurate approximation of \( F \) will limit the number of cascaded networks that can be applied to obtain useful results.

### 3.6.2 System identification

The previous subsection discussed the problem of prediction of a system. The neural network could also be used to identify a system. In the same way as in linear system identification the neural network can be used to model a process. The difference between the linear method and the neural network approach that in the first parameters are approximated while in the latter weights are calculated that have no particular meaning. Only the full network has meaning, that of the model dynamics. Identification of linear and nonlinear processes using neural networks is described in section 5.2.

If the inputs to a neural network are formed by a regressor vector, as in linear identification, the neural network approach is sometimes denoted as e.g. NARX or NARMAX method to indicate the nonlinear or neural approach.
Chapter 4

Framework

Chapter 1 introduced the concept of predicting future values of a dynamical system to reduce disturbances. This outline will now be extended after which the problems expected will be highlighted and approaches for the solutions of them will be discussed. The setup of disturbance reduction is done with a neural network in the feedback loop. This neural network has to be trained with a chaotic signal to be able to predict future values. Depending on the process involved, which can possess delays, one or more future values have to be predicted. If knowledge of the chaotic disturbance is present measurements of this chaotic signal can be used to train the neural network. If no distinct measurements of the chaotic disturbance are available, the residuals of the complete setup have to be used to train the neural network. The possibility of predicting future values can be divided into the following two categories:

- Cascading neural networks with one-step ahead prediction.

- Multi-step prediction.

In case the (chaotic) disturbance is of order one, i.e. \( v(k+1) = F[v(k)] \), the one-step ahead prediction (calculation of \( v(k+1) \) using only \( v(k) \)) is in this case done by the approximation of \( F \) by the neural network. If more than one future value has to be predicted cascading the neural networks can be applied. Figure 4.1 shows this setup. The consecutive neural networks are considered identical, each predicting the one-step ahead value. In this way, several future values can be calculated using only the current value \( x(k) \) or values until time instant \( k \).

The value of \( v(k) \) in case of the population model can be considered the output or the state of the system, i.e. \( y(k) = x(k) \). There is one state equal to the output. This will not be the case in all situations. Observability of all states is not guaranteed but in this report it will be assumed that all states can be reconstructed using measurements of the output.

Deviations from the real dynamics \( (F) \) will influence the prediction quality and this deviation is passed on to consecutive neural predictors leading most likely to erroneous results at predictions laying further in the future. The quality of approximation by the neural network will have a great influence on the prediction horizon.

A change in point of view at this scheme could improve the performance of the cascading. This would involve the minimization of the sum of squared errors of all outputs \( (\hat{x}(k + 1), \hat{x}(k + 2) \) and \( \hat{x}(k + 3)) \). Because the predictions are based on past measurement until time instant \( k \), the future values are denoted as \( \hat{x}(k+i \mid k) \). The criterium to be minimized then
reads:

$$J = \sum_{k=1}^{N} \sum_{i=1}^{H_p} \| \hat{x}(k+i | k) - x(k+i | k) \|^2$$

(4.1)

where $H_p$ represents the prediction horizon (the number of future outputs to be predicted), $N$ is the length of the dataset used to train the network and $\hat{x}$ is the approximated value of $x$. The computations described in this report have only minimized the cost function for one network. Adaptation of the program is needed to minimize the cost function for cascaded networks.

Application of the cascading implies that each network can be kept rather simple and the computational effort is reduced in relation to the possibility of trying to compute several predictions with just one network. The network layout would in this case become quite large in magnitude, i.e. a large number of nodes and multiple hidden layers.

If only the present value ($x(k)$) is used, successive delays can provide the network with past values. This makes it easy to enlarge the vector of past states if necessary. The higher the order of the function the more past states will usually be required. Figure 4.3 shows a neural network that approximates the one-step ahead value using a regressor vector with length $p$.

Combinations of this scheme and the layout given in figure 4.1 can be thought of. The use of delays is in fact no more than was given in figure 4.2 where the inputs $x(k)$, $x(k-1)$ and $x(k-2)$ act as regressor vector. Combining the two could result in a scheme given in figure 4.4.

The prediction of the population model (2.4) is quite simple. The underlying dynamics of this model is of first order and the approximation can be achieved most accurate as was
Figure 4.3: Neural network with consecutive delays to provide \( p \) inputs.

Figure 4.4: Cascade of neural networks. Each network determines the one-step ahead prediction using past values.
shown in chapter 3. Higher order models like the Henon map (4.2) use two past values to calculate the new value.

\[ x(k + 1) = 1 - ax^2(k) + bx(k - 1) \]  

(4.2)

The Henon map shows chaotic behavior for e.g. \( a = 1.4 \), \( b = 0.3 \). Function \( F \) to be approximated now is \( x(k + 1) = F[x(k), x(k - 1)] \). The Henon map is a good example to introduce state space models. Two states can be used to describe the Henon map. This is done in the following equations:

\[
x_1(k + 1) = 1 - ax_1^2(k) + x_2(k) \quad (4.3)
\]

\[
x_2(k + 1) = bx_1(k) \quad (4.4)
\]

If state space models are used and not all states are available, state estimators (or observers) have to be applied that will be used to design a proper control law.

The simulations that will be done try to show that disturbance reduction is possible using predictive neural networks. A linear process will be used that is disturbed by a chaotic process. Objective in this setup is to follow a reference signal by application of the inverse model of the process as controller. It will be assumed that inverse modeling is possible and does not lead to instability to avoid problems concerning limited performance of the controller to be designed. Main interest is the contribution of the predicted value of the chaotic disturbance. The disturbance and its reduction are of main interest, not the control strategies of the plant itself.

The model of the process will be obtained by an identification session using an input value that is persistently exciting [2] and output measurements that are disturbed (the desired output of the process is not available due to disturbances and measurement noise). The disturbance is in first approach be taken of first order (i.e. the population model) and initially a rather ideal situation will be considered. Initially, no measurement noise will be added to the output but gradually more and more deviations from the idealized picture are introduced. These deviations include the, in practice inevitable, measurement noise, unmodeled process dynamics and non-accurate prediction of future disturbances by the neural network.

In chapter 6 the configuration, training and results of several configurations of the neural network predictions will be discussed after which the predictive properties are built in as feedback in the controlled system.

As described in section 2.3 the prediction of a chaotic signal will highly depend on the initial condition. Deviations from the real value will affect the quality of the prediction and reduce the prediction horizon. Expected is therefore that influence of measurement noise, model estimation error and the like will deteriorate the controlled setup as shown in figure 1.2 in chapter 1.
Chapter 5

Process identification

5.1 Linear identification method

To obtain a controlled system a first step is to obtain a model of the process to be controlled. This chapter describes some methods to obtain a model of the process in question. These models can either be linear or nonlinear. At the same time the process to be identified can be linear or nonlinear. Both combinations will be described in the following sections to show the differences, advantages and drawbacks of these methods.

Identification using an Output Error (OE) model structure is applied and a suitable test signal has to be designed. The test signal has to be persistently exciting in order to activate all system dynamics. As said, the identification structure used is the linear Output Error (OE) model [2]. This method makes use of independent parametrized plant and noise models. The OE model is represented by the following equations:

\[ F(q)w(k) = B(q)u(k) \]  \hspace{1cm} (5.1)
\[ y(k) = w(k) + v(k) \]  \hspace{1cm} (5.2)

where the plant model equals \( G(q) = B(q)/F(q) \) (see figure 5.1) and the noise model \( H(q) = 1 \).

The model can be written as difference equations as in the following example that is chosen to be proper (direct feedthrough from input to output):

\[ w(k) = b_1 u(k) + b_2 u(k-1) - f_1 w(k-1) - f_2 w(k-2) \]  \hspace{1cm} (5.3)
\[ y(k) = w(k) + v(k) \]  \hspace{1cm} (5.4)
An additive model estimation error ($\Delta M$) will propagate the control signal in such manner that the residue contains an error induced by $\Delta M$. The residue is passed to the neural network which predicts future values of the residue (Fig. 1.2).

5.1.1 Linear process

The process to be identified in this section is a linear process with two poles at $z = 0.3$ and $z = 0.2$ respectively:

$$H_p(z) = \frac{z^2}{z^2 - 0.5z + 0.06} \quad (5.5)$$

The process is excited with a Zero Mean White Noise signal (uniform distribution with values between $-2$ and $+2$) and the output is disturbed by a chaotic signal, i.e. the population model with parameter $r = 4$. The signals that in practice will be available are the input and the disturbed output. These signals will be used to model the process and are summed up in table 5.1.

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Process identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification method</td>
<td>Output Error</td>
</tr>
<tr>
<td>Data length</td>
<td>5000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 0.5z + 0.06$</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed [-2..2]</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model, $r = 4$</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed [-0.1..0.1]</td>
</tr>
</tbody>
</table>

The advantage now is that the chaotic disturbance is considered as being noise by the identification method which is based on correlation (See section 2.2). The resulting model depends on the combination of the amount of noise added and the length of the dataset but mainly on the correct choice of regressors (the order of the model). In the MATLAB Identification Toolbox these variables are entered as $[n_b \ n_f \ n_k]$ where $n_b$ and $n_f$ correspond with the number of variables in the polynomials $B(q)$ and $F(q)$ in (5.1) respectively. The delay of the input to the output is set by $n_k$ which in case of the process used here (5.5) is zero. Two identifications are carried out. In the first session the correct number of regressors is used (i.e., the correct model order) resulting in

$$H_{m_1} = \frac{0.9979z^2}{z^2 - 0.4950z + 0.0616} \quad (5.6)$$

in which case no severe estimation mismatch is present. The length of the dataset was taken 5000 samples. Validation of the process (5.5) and the corresponding model (5.6) yields the time signals drawn in figure 5.2.
5.1 Linear identification method

Figure 5.2: Simulation data: Disturbed output (solid) of process $H_p$ and corresponding model ($H_{m_1}$) output (dashed). The model is obtained using an Output Error identification. The process order is correctly estimated.

In case the process order is underestimated (which in practice can easily occur because the order of the process is not known and the process will probably be nonlinear so a real order cannot be assigned) as in the next identification session the resulting model yields the transfer as in 5.7. The experiment data is given in table 5.1. The validation data is drawn in figure 5.3.

\[ H_{m_2} = \frac{1.0104z}{z - 0.4452} \] (5.7)
The process used so far (5.5) had two poles at $z = 0.3$ and $z = 0.2$ respectively. The process behavior caused by these poles can be considered to possess quite moderate characteristics. Change of the process to

$$G_p(z) = \frac{z^2}{z^2 - 1.15z + 0.19}$$

(5.8)

which contains the same pole at $z = 0.2$ but the second pole is shifted to $z = 0.95$ which leads to slower dynamics. The identification of this process (5.8) is again done with correct and underestimated process order. Experiment data is given in table 5.2.

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Process identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification method</td>
<td>Output Error</td>
</tr>
<tr>
<td>Data length</td>
<td>5000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 1.15z + 0.19$</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed [-2..2]</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model, $r = 4$</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed [-0.1..0.1]</td>
</tr>
</tbody>
</table>
5.1 Linear identification method

The correct estimation leads to

\[ G_{m1} = \frac{1.0014z^2}{z^2 - 1.1467z + 0.1855} \]  

(5.9)

whereas underestimation yields

\[ G_{m2} = \frac{1.2046z}{z - 0.9537} \]  

(5.10)

In this case, only the slow characteristics of the process are modeled leading to large errors between the real and approximated output. Comparison of the time signals in figures 5.4 and 5.5 illustrates this.

![Real (solid) and estimated (dashed) output](image)

Figure 5.4: Simulation data: Disturbed output (solid) of process \( G_p \) and corresponding model (\( G_{m1} \)) output (dashed). The model is obtained using an Output Error identification with correct order estimation.
Resuming this section gives an indication of the problems that can be expected if the process is to be controlled in the complete disturbance reduction setup. Model estimation mismatch results in deviations from the desired output which in turn leads to a residue (to be passed to the neural network) strongly deviating from the real chaotic disturbance. Consequences will be described in chapter 7.

5.1.2 Nonlinear process

The linear process could easily be identified by the OE model identification method. More complicated process behavior will probably lead to higher order OE models. This section shows the difference between linear and nonlinear identification methods. The linear OE model and the neural network approach are both used to identify a linear and a nonlinear system. The nonlinear process (in discrete time domain) to be identified is:

\[ y(k + 1) = \frac{y(k)}{1 + y^2(k)} + u(k) \]  

Identification of this process (5.11) which can be written as

\[ y(k + 1) = F[y(k), u(k)] \]  

is done again with a chaotic disturbance at the output (population model with parameter \( r = 4 \)). Experiment data is given in table 5.3.
5.1 Linear identification method

Table 5.3: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Process identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification method</td>
<td>Output Error</td>
</tr>
<tr>
<td>Data length</td>
<td>5000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 0.5z + 0.06$</td>
</tr>
<tr>
<td>Model</td>
<td>$b_0/1 - a_1z^{-1}$</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed $[-0.5..0.5]$</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model, $r = 4$</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed $[-0.1..0.1]$</td>
</tr>
</tbody>
</table>

The OE model obtained is given by the transfer function

$$F_m = \frac{0.8963}{z - 0.7405} \quad (5.13)$$

or written in the discrete time domain as

$$y(k + 1) = 0.7405y(k) + 0.8963u(k) \quad (5.14)$$

The validation of this model is drawn in figure 5.6. As expected the resulting time signals have quite a great deviation from the desired response. Model order increase will improve the model fit but the results as depicted in figures 5.2 to 5.4 will not be achieved. To compare the linear identification method with a nonlinear method the next section will shortly highlight identification using neural networks.

![Figure 5.6: Simulation data: Validation of the model obtained with an OE structure for a nonlinear process ($F_m$). Solid refers to the real process, dashed to the model output.](image-url)
5.2 Identification using neural networks

5.2.1 Linear process

Identification of the process to be controlled is done with linear identification tools in the previous section. The application of the neural network has so far only been applied to capture the chaotic dynamics of the disturbance. As is done more and more last years and most likely in future the process dynamics can also be identified using neural networks ([10], [11] and [12]). Especially the nonlinear properties can be very useful in identifying real processes that are most likely nonlinear. More and more attention is paid to applications and investigation is done in the area of state space domain, observability and controllability [13], [14].

The linear process used throughout this report will now be identified using a neural network. It is expected that the neural network will try to approximate both the linear transfer function of the process and the chaotic disturbance present in the output signal. Input and output signals are used that have been obtained with a MATLAB process simulation. The process (5.5) has been excited with a uniform distributed white noise signal with values between -1 and +1 input signal and a chaotic disturbance has been added to the output. Main problems in this scheme are the choice of the number of past inputs and outputs and the neural network architecture. Table 5.4 shows all relevant experiment data.

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Process identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification method</td>
<td>Neural network approach</td>
</tr>
<tr>
<td>Data length</td>
<td>2000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 0.5z + 0.06$</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed [-1..1]</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model, $r = 4$</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed [-0.1..0.1]</td>
</tr>
<tr>
<td>Neural network</td>
<td>3 inputs, 1 hidden layer (4 nodes), 1 output</td>
</tr>
<tr>
<td>Additional</td>
<td>Prediction error method</td>
</tr>
</tbody>
</table>

A view at the transfer function of the process to be identified (which in practice is not known) gives knowledge of the number of inputs to be used.

$$\frac{Y(z)}{U(z)} = \frac{z^2}{z^2 - 0.5z + 0.06}$$  \hspace{1cm} (5.15)

which can alternatively be written in the (discrete) time domain as

$$y(k) = 0.5y(k-1) - 0.06y(k-2) + u(k)$$  \hspace{1cm} (5.16)

The network architecture will be a three-input, single output feedforward network. The three inputs will be $u(k)$, $y(k-1)$ and $y(k-2)$ while the output is the corresponding process output $y(k)$. The network will try to approximate the parameters 0.5, 0.06 and 1. Expected problem is the chaotic disturbance present in the output signal that is given as the network output signal ($y(k)$). As seen in section 5.1 the chaotic disturbance was considered a white noise.
signal in the linear identification method and could be eliminated using correlation analysis. However, as said before, the neural network will possibly try to approximate this chaotic disturbance in combination with the (linear) process. Results of the neural network identification is given in figure 5.7.

The estimation set was created with a process simulation in the MATLAB Identification Toolbox using *idsim.m*. The process has been excited with a uniform distributed white noise signal with values between $-1$ and $+1$ and a chaotic disturbance was added (i.e. the population model with $r = 4$) to the output.

Each configuration (i.e. neural network architecture) has been optimized twenty times which means that the initial weights in the network are chosen randomly resulting in different final network weights and cost function. The cost function appears to be a complicated function (i.e. the sum of squared errors between the desired output and the neural network approximated output as a function of the network weights). No global minimum can be found for the network. After the optimization the network is chosen which has the smallest final cost function. For the linear process to be identified this resulted in a four-node hidden layer network with three input nodes and one output node. Figure 5.7 shows the time signals of the validation of the network, the experiment data is given in table 5.5.

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Neural network validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network layout</td>
<td>3 inputs, 1 hidden layer (4 nodes), 1 output</td>
</tr>
<tr>
<td>Data length</td>
<td>2000 samples</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed [-1..1]</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model, $r = 4$</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed [-0.1..0.1]</td>
</tr>
</tbody>
</table>

The input signal again is a ZMWN sequence. As in section 5.1.1 the identification is quite good. The neural network identified both the linear transfer function as the chaotic disturbance. However, comparison with the OE model is not fair because the neural network setup equals that of a prediction error estimate. In fact, an OE structure had to be applied involving recurrent backpropagation.
5.2.2 Nonlinear process

The nonlinear process (5.11) is now to be identified by the neural network. The signals \( u(k) \) and \( y(k) \) serve as input value of the network, \( \hat{y}(k+1) \) determines the output. The initial value of \( y(k) \) is in each simulation set to zero. The input signal \( u(k) \) is a uniform distributed white noise signal with values between \(-2\) and \(+2\). The population model again serves as chaotic disturbance. Also an additional noise term (with the same characteristics as \( u(k) \) but values between \(-0.1\) and \(+0.1\)) is added to the output. Experiment data is given in table 5.6.

Table 5.6: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Neural network training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network layout</td>
<td>2 inputs, 1 hidden layer, 1 output</td>
</tr>
<tr>
<td>Data length</td>
<td>2000 samples</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed [-1..1]</td>
</tr>
<tr>
<td>Process</td>
<td>( y(k+1) = y(k)/1 + y^2(k) + u(k) )</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model, ( r = 4 )</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed [-0.1..0.1]</td>
</tr>
<tr>
<td>Additional</td>
<td>Nonlinear process identification</td>
</tr>
</tbody>
</table>

A feedforward neural network is used with one hidden layer. A different number of nodes
is set in each training to be able to select the best network architecture.

The nonlinear process to be identified (5.11) can be identified quite good as seen in figure 5.8. Best result was obtained with a five-node hidden layer feedforward neural network.

Figure 5.8: The results of the neural network identification of a nonlinear process. The neural network consisted of four nodes. Inputs are $u(k)$ and $y(k)$. Output is $y(k + 1)$ and was disturbed by a chaotic signal.

Comparing figures 5.6 and 5.8 shows that the identification capabilities of the neural network is quite good but comparison with the linear method resulting in an OE model cannot be made. The neural network is trained with a prediction error estimate. Main drawback is here that the computational effort required for the neural network identification is quite large. Especially when complicated functions have to be approximated, datasets and neural network architectures grow larger the computation time may be a limiting factor that reduces the identification quality. The time required in this experiment easily reaches 100 times that of the linear method.

The method used here to optimize the neural network is a prediction error estimate. The linear method used an Output Error method whereas this would lead here to a recurrent optimization method. Recurrent backpropagation must then be used. A program that applies this method is available but yet not used.
Chapter 6

Prediction

6.1 One-step ahead

6.1.1 First order systems

In case no delay is present in the transfer from reference signal to output signal the prediction of future values of the disturbance suffices with one step in the future to reduce the output disturbance. The value of \( \hat{v}(k+1) \) was predicted using only \( v(k) \). The population model can therefore be considered of first order. In chapter 3 it was shown that training of a one layer network could fairly approximate this one-step ahead prediction. Expansion of the prediction horizon (multi-step prediction) could be necessary in e.g. a MPC setup or in case the process possesses delays. In case of the population model the prediction of more than one value will require several transfer functions to approximate from input to output. The one-step ahead prediction is carried out with a dataset of 2000 samples of the population model. The mean value of the signal has been removed. A neural network consisting of one input, one output and a four-node hidden layer is trained to approximate the population model.

![Figure 6.1: Transfer function of the population model. Real (thick dots) and approximated (thin dots). Approximation is done by a four-node hidden layer feedforward neural network](image)
Figure 6.1 shows the approximation of the transfer function of the model \( x(k) \rightarrow \hat{x}(k+1) \) and figure 6.2 shows the corresponding time signals.

![Figure 6.2: Time signals of the population model. Real (solid) and approximated (dashed). Approximation is done by a four-node hidden layer feedforward neural network](image)

Figures 6.1 and 6.2 clearly indicate that the first order population model characteristics are well approximated by the neural network. Even a small network (one hidden layer, four nodes) is capable of capturing the process dynamics of this model.

The circumstances under which the training is carried out expects that proper knowledge is available of the chaotic process. If this is not the case, the only available signals are the (simulation) residuals which in turn must be used to train the neural network. These residuals contain besides the chaotic disturbance the error due to the model estimation mismatch and measurement noise. Natural question now is, if the neural network is able to capture the chaotic dynamics that is more or less hidden in the residue. The process used is

\[
H_p(z) = \frac{z^2}{z^2 - 0.5z + 0.06}
\]  

(6.1)

and is disturbed by the chaotic population model. All identification information is given in table 6.1

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Process identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification method</td>
<td>Output Error</td>
</tr>
<tr>
<td>Data length</td>
<td>5000 samples</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed [-1..1]</td>
</tr>
<tr>
<td>Process</td>
<td>( z^2/z^2 - 0.5z + 0.06 )</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model (( r = 4 ))</td>
</tr>
<tr>
<td>Noise</td>
<td>not added</td>
</tr>
</tbody>
</table>
An Output Error model is obtained using a dataset of 5000 samples that reads

\[
H_m(z) = \frac{1.0086z^2}{z^2 - 0.5066z + 0.0629}
\]  

(6.2)

In figure 6.3 the transfer function of the validation data and the resulting approximated transfer function is drawn. Measurement noise has not been added to the output. The accompanying time signals are shown in figure 6.4. As is shown in these figures the neural network is capable of capturing the chaotic dynamics from the residue. Condition for this result however is that the noisy part in the residue must be uncorrelated.

![Figure 6.3: Transfer functions of the validation data (left) and the approximated data (right). The (four-node) network is trained with the residue of the simulation.](image-url)
After taking this residue to train the network an additional measurement noise has been added to the output signal. Uniform distributed zero mean white noise is added to the output and used to train the network. Values of the input are between $-2$ and $+2$. Experiment information is given in table 6.2.

Table 6.2: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Neural network training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training goal</td>
<td>One-step ahead prediction</td>
</tr>
<tr>
<td>Data length</td>
<td>2000 samples</td>
</tr>
<tr>
<td>Network layout</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training data</td>
<td>Simulation residual</td>
</tr>
<tr>
<td>Additional</td>
<td>Residual contains chaotic population model ($r = 4$) and measurement noise (ZMWN, uniform distributed, $[-0.2..0.2]$) plus a small error due to model estimation mismatch</td>
</tr>
</tbody>
</table>

The transfer function ($\xi(k+1)$ as a function of $\xi(k)$) from the residual is shown in the left part of figure 6.5. The residue clearly contains an additional noise term when comparing this figure with figure 6.3. As can be concluded from this figure the neural network still is very well capable of capturing the chaotic part of the residue and sorting out the measurement noise. The transfer function of the validation of the network is shown in the left part of
6.1 One-step ahead

The corresponding time signals are drawn in figure 6.6. The deviation from the real value grows larger as seen from this figure. The amount of noise with respect to the chaotic disturbance is approximately 20%.

Figure 6.5: Transfer function of the validation data (left) and the approximated data (right). The (four-node) neural network has been trained with the residue of the simulation. The residue contained an error due to model estimation mismatch, a chaotic part and an additional measurement noise.

Figure 6.6: Real (solid) and approximated (dashed) time signals of the residual prediction
6.1.2 Higher order systems

As was seen, the population model can be approximated using a one-input, one-output neural network. A higher order model was already given in chapter 4, the Henon map (4.2). Training with this model can be done with two different neural network architectures. As indicated in (4.2) and (4.3) the Henon map can be written in two ways. The first leads to a neural network with two inputs and one output (i.e. \( x(k - 1), x(k) \) and \( \hat{x}(k + 1) \), respectively) while the second, the state space representation that was introduced in chapter 4 leads to a two-input, two-output network. Inputs then are \( x_1(k) \) and \( x_2(k) \), outputs of the network are \( \hat{x}_1(k + 1) \) and \( \hat{x}_2(k + 1) \). Note here, that only a difference in representation is introduced. The system remains the same. However, the network architecture is slightly changed. In the first case the hidden layer was composed of six nodes while the latter case leads to an eight node-layer with best performance. Both states must be available. Table 6.3 gives the information of the neural network training.

Table 6.3: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Neural network training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training goal</td>
<td>One-step ahead prediction</td>
</tr>
<tr>
<td>Data length</td>
<td>2000 samples</td>
</tr>
<tr>
<td>Network layout</td>
<td>2 inputs, 8 hidden nodes, 2 outputs</td>
</tr>
<tr>
<td>Training data</td>
<td>Henon map ((a = 1.4, b = 0.3))</td>
</tr>
<tr>
<td>Additional</td>
<td>Setup in state space</td>
</tr>
</tbody>
</table>

The resulting time evolution of the validation of the first network (six nodes) is given in figure 6.7.

![Figure 6.7: Neural network approximation of the Henon map. Real value (solid) and approximated value (dashed). Training is carried out using a neural network with two inputs, one output and one hidden layer with six nodes.](image-url)
Experiment data is given in table 6.4.

**Table 6.4: Experiment data**

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Neural network training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training goal</td>
<td>One-step ahead prediction</td>
</tr>
<tr>
<td>Data length</td>
<td>2000 samples</td>
</tr>
<tr>
<td>Network layout</td>
<td>2 inputs, 6 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training data</td>
<td>Henon map (a = 1.4, \ b = 0.3)</td>
</tr>
<tr>
<td>Additional</td>
<td>Two past values are used to calculate the next value</td>
</tr>
</tbody>
</table>

The validation data of the network in state space (the eight-node network) is shown in figure 6.8.

![Figure 6.8: Approximation of the Henon map in state space. Upper plot shows state \(x_1(k)\) (solid) and its approximated value (dashed), the lower plot shows \(x_2(k)\)](image)

From these figures (figs. 6.7 and 6.8) it can be seen that a good approximation can be achieved. The step to state space is quite easy. The state \(x_2(k)\) is a factor \(b\) times the delayed first state \((x_2(k) = b.x_1(k - 1))\) which can also be seen from figure 6.7. Both states can be reconstructed using past output values (e.g. \(y(k) = Cx_1(k)\)).

The dataset to train the network was taken 2000 samples, the parameters \(a\) and \(b\) had been set to 1.4 and 0.3 respectively. In figure 6.9 the approximation of the Henon attractor is drawn. The transfer \(x(k - 1) \rightarrow x(k + 1)\) and \(x(k) \rightarrow x(k + 1)\) is combined in this figure \([x(k - 1), x(k)] \rightarrow x(k + 1)\) to create a trajectory. The data is taken from the first network approximation. Again, a good approximation is achieved. However, it must be noted that the training is done for very particular trajectories. In case different startvalues are used for the Henon map different trajectories will turn up eventually converging to the attractor
drawn in the figure or completely erratic behavior will arise. In the next section the one-step ahead prediction will be used to predict more than one future output by cascading identical networks.

![Real and approximated Henon attractor](image)

**Figure 6.9:** Real (thick dots) and approximated (thin dots) of the Henon attractor. Approximation is done with a six-node hidden layer feedforward neural network.

### 6.2 Multi-step prediction

#### 6.2.1 Cascading neural networks

In chapter 4 the application of a number of neural networks in series was mentioned. The higher order functions that have to be approximated can be obtained using several neural networks in a series connection. Deviations from the real function that has to be approximated will decide the number of successive networks to predict more than one future value at the same time instant $k$. Figure 4.1 shows the idea of this setup. Experimentally, the population model again is used to show the restriction on the number of neural networks to be concatenated. First, a training of the neural network has been carried out. The population model is of first order, therefore the neural network is designed to predict $x(k+1)$ using only $x(k)$. The training of the network was done using several network lay-outs, varying from four to ten hidden nodes. The network (with one input and one output) performed best with four nodes in one hidden layer. A series connection of four identical of these networks must then be capable of predicting $x(k+1), x(k+2), x(k+3)$ and $x(k+4)$. The transfer functions of each
predicted value as a function of the input value \( x(k) \) is drawn in figure 6.10. All functions are
drawn as a function of the present value of the population \( x(k) \). From this figure it becomes
clear that the prediction quality is poor for values of \( x(k) \) around 0.5. It will become clear
later what the reason is for this behavior.

![Figure 6.10: Transfer functions of predicted values of the population model using a series connection of four identical neural networks, each calculating the one-step ahead prediction. Upper left: \( x(k) \) to \( \hat{x}(k+1 \mid k) \), upper right: \( x(k) \) to \( \hat{x}(k+2 \mid k) \), lower left: \( x(k) \) to \( \hat{x}(k+3 \mid k) \) and lower right: \( x(k) \) to \( \hat{x}(k+4 \mid k) \).](image)
For comparison the error between the real and approximated transfer function is plotted in figure 6.11. The deviations mainly appear for values around \( x(k) = 0.5 \). The histogram for the population model for which the network was trained is shown in figure 6.12. It can be concluded from this figure that the network is mainly trained for values near the points 0 and 1 and how serious the lack of training data affects the prediction.

As can be seen from figure 6.10 the distortion from the desired function to be approximated grows larger for predictions further in future particular for past values around \( x(k) = 0.5 \). The
corresponding time signals are given in figures 6.13 to 6.16. The data used in this experiment is given in table 6.5.

Table 6.5: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Increasing prediction horizon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Cascaded one-step ahead prediction</td>
</tr>
<tr>
<td>Data length</td>
<td>100 samples</td>
</tr>
<tr>
<td>Network layout</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training data</td>
<td>Population model ($r = 4$)</td>
</tr>
<tr>
<td>Additional</td>
<td>Optimization is done for one neural network.</td>
</tr>
<tr>
<td></td>
<td>Four identical networks are connected in series.</td>
</tr>
</tbody>
</table>

Figure 6.13: Real and predicted population after one neural network, i.e. $\hat{x}(k + 1) = F[x(k)]$
Figure 6.14: Real and predicted population after two neural networks, i.e. 
\[ \hat{x}(k + 2) = F^2[x(k)] \]

Figure 6.15: Real and predicted population after three neural networks, i.e. 
\[ \hat{x}(k + 3) = F^3[x(k)] \]
6.2 Multi-step prediction

Figure 6.16: Real and predicted population after four neural networks, i.e.
\[ \hat{x}(k + 4) = F^4[x(k)] \]

It may be clear that the small deviations grow higher and higher with time. The divergence of nearby trajectories plays an important role in this situation (see section 2.3).

6.2.2 Increased horizon

The naming of multi-step prediction may be confusing when comparing this with the cascading of neural networks. In the latter case, identical networks, each predicting one-step ahead, were used. With increased horizon prediction is meant that a single network predicts several future values as indicated in figure 4.2. During optimization the cost function will be minimized (i.e. the sum of squared errors) for more than one future output but the required computation time and most likely the network size will increase considerably. An example was already given in figure 3.7. Figure 6.17 shows the validation of a nine-node hidden layer network that predicts \( x(k + 1) \) and \( x(k + 2) \) based on \( x(k) \). The prediction of the first value is still quite good but the second has a greater deviation. Training of the network could improve the quality of both the predictions but the extension of the prediction horizon with one sample implied already an increase of the network with five nodes in the hidden layer. However, it is not proven that this network architecture has the best properties. Multi-layer networks could be used or extension of the datasets for training but no guarantee can be given that this will indeed improve the results.
Concluding this chapter some remarks can be given. The prediction of one sample in future can be achieved quite good. If a series connection is made of several identical networks the prediction horizon can be enlarged but after a few steps severe errors appear. Optimization of cascaded networks has not been carried out but may lead to better results. The optimization is in this setup done for one neural network and the network with the best performance has been copied to obtain a series connection. Training could also be done with the networks already connected and minimizing the total cost function for all predicted values.
Chapter 7

Disturbance reduction

After training neural networks to predict future values (chapter 6) and identification of the process to be controlled (chapter 5) the results can be brought together to complete the setup given in chapter 4 of the complete system of disturbance reduction given in figure 1.2. To show that the idea given in the introduction leads to good results under ideal circumstances this will be shown with the help of some simulations and the ensuing figures. The experiment information is given in table 7.1.

Table 7.1: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Controlled system simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Show effect of disturbance reduction</td>
</tr>
<tr>
<td>Data length</td>
<td>100 samples</td>
</tr>
<tr>
<td>Network layout</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 0.5z + 0.06$</td>
</tr>
<tr>
<td>Model</td>
<td>$z^2/z^2 - 0.5z + 0.06$</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model ($r = 4$)</td>
</tr>
<tr>
<td>Noise</td>
<td>Not added</td>
</tr>
</tbody>
</table>

Under ideal circumstances is meant that the process is exactly modeled and the model is invertible. Also is assumed that the neural network can predict future values of the chaotic disturbance with high accuracy. Thus:

$$H_m = H_p$$  \hspace{1cm} (7.1)

and assume $H_m^{-1}$ exists. For the neural network approximation:

$$F_{NN}[v(k)] = F[v(k)]$$  \hspace{1cm} (7.2)

where $F[v(k)]$ represents the chaotic disturbance function and $F_{NN}[v(k)]$ the approximation by the neural network. The measurement noise will initially be set to zero ($\eta = 0$). Figure 7.1 shows the setup.

The process used has a second order linear transfer function (7.3) in discrete time:

$$H_p(z) = \frac{z^2}{z^2 - 0.5z + 0.06}$$  \hspace{1cm} (7.3)
Disturbance reduction

Suppose that the model has been determined identical to the process and is invertible (resulting in $H_c(z) = z^2 - 0.5z + 0.06/z^2$ which is causal) the feedthrough from input to output is unity ($H_cH_p = 1$). The disturbance at the output is given to be the population model with parameter $r = 4$. The controlled system has been simulated with and without the predictive feedback (i.e. the neural network) and yields the time evolutions as given in figure 7.2. Data used in this experiment is given in table 7.1.

Figure 7.1: Controlled system for disturbance reduction. The controller $H_c$ is assumed to be the inverse of the model $H_m$ to achieve signal tracking from input to output. A neural network is added to reduce the effect of disturbances at the output of the system.

Figure 7.2: SIMULINK simulation of second order linear process with chaotic disturbance at the output. Solid curve refers to the reference, dashed curve is the output with predictive feedback and dotted without feedback.
It may be clear that under ideal circumstances the prediction of the chaotic disturbance almost completely suppresses the disturbance. The dashed curve (output with disturbance reduction) has only a small deviation from the reference signal (it almost exactly coincides with the reference signal) and is caused by the neural network imperfection.

7.1 Model estimation mismatch

7.1.1 First order disturbances

The results given in figure 7.2 will in practice never be achieved. The process probably will be nonlinear and can only be approximated by linear identification techniques. The estimation mismatch will give rise to an additional component in the residue (the difference between the real and model output) that is passed to the neural network causing divergent predictions.

To put this notion more formal:

\[
H_m = H_p + \Delta H
\]  

(7.4)

The influence of this factor \(\Delta H\) will now be examined. Chapter 5 already indicated the deviation of the model from the real process. Now the result of this estimation mismatch with respect to the controlled system will be looked at.

In practice, the measured output does not equal the desired process output. The measurements are obscured by a disturbance, in this case considered chaotic. Identification of the linear process (7.3) using the data given in table 7.2 yields the following model:

\[
H_{m1}(z) = \frac{0.9985z^2}{1.0z^2 - 0.4985z + 0.0612}
\]  

(7.5)

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Process identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Output Error</td>
</tr>
<tr>
<td>Data length</td>
<td>10000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>(z^2/z^2 - 0.5z + 0.06)</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed, ([-2..2])</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model ((r = 4))</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed, ([-0.2..0.2])</td>
</tr>
</tbody>
</table>

The identification session is carried out using an Output Error structure (5.1) with parameters \([1 \ 2 \ 0]\), a dataset of 50000 samples, a uniform distributed white noise signal with values between \(-2\) and \(+2\) as input signal the same kind of signal as additional noise but with values between \(-0.2\) and \(+0.2\). Figure 7.3 shows that also in this case predictive feedback reduces the disturbance at the output.
Disturbance reduction

Figure 7.3: SIMULINK simulation of second order linear process with chaotic disturbance at the output. Solid curve refers to the input, dashed curve is the output with predictive feedback and dotted without feedback. The model $H_{m_1}$ (obtained using an identification session with OE structure) contains a small estimation error.

A difference from the ideal situation can hardly be distinguished. A reason is that the linear identification method cannot discriminate chaotic behavior from random behavior (recall section 2.2). The chaotic disturbance additive to the output has been considered a noise signal and can be eliminated in case this signal is not correlated with the input signal (the dataset must be sufficiently large). The effect of disturbance reduction is made more clear in case the error signals are plotted as a function of time as in figure 7.4. The reference signal is subtracted from the output signal both with and without disturbance reduction feedback to show the effect of the predictive feedback.
7.1 Model estimation mismatch

Figure 7.4: Error between reference and output signals. Solid curve shows the effect of disturbance reduction in relation with the system without this reducing measure (dashed curve).

The controller $H_c$ is taken to be the inverse of the model to make the total transfer function unity and to achieve a reference tracking system. The error between reference and output is thus preferably zero. Figure 7.4 shows that this effect can almost be reached in case the process is accurately modeled. Note here that still no sensor noise has been added to the output. This effect will be discussed in section 7.2.

The model that has been used so far had two poles at $z = 0.2$ and $z = 0.3$ respectively. In the model (5.8) the pole at $z = 0.3$ is shifted to $z = 0.95$ and improper modeling resulted in capturing only the slow dynamics of the process. When the model is incorporated in the controlled system, the error due to the model estimation mismatch is passed on to the neural network which gives rise to the following system behavior. Recall the processes (5.5, 5.8) and models (5.6, 5.7 and 5.9) from section 5.1. Table 7.3 shows all simulation data.

Table 7.3: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Disturbance reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Show effect of predictive feedback</td>
</tr>
<tr>
<td>Data length</td>
<td>1000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2 / z^2 - 1.15z + 0.19$</td>
</tr>
<tr>
<td>Model</td>
<td>$1.2046z / z - 0.9537$</td>
</tr>
<tr>
<td>Neural network</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training</td>
<td>With chaotic signal</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed, [-2..2]</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model ($r = 4$)</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed, [-0.2..0.2]</td>
</tr>
<tr>
<td>Additional</td>
<td>Training done with real chaotic signal, not residual information</td>
</tr>
</tbody>
</table>
As can be concluded from figure 7.5 the predictive feedback can better be left out. The model estimation mismatch has such influence on the controlled system that a severe distortion from the desired behavior occurs. To show the reason that causes this behavior the transfer function of the neural network is plotted in figure 7.6.

The neural network is trained with an estimation set of values between zero and one; \( x_{est} \in [0,1] \). The training was carried out with the real chaotic disturbance, not with the residue. The population model (2.4) is only valid for these values. Figure 7.6 shows what
7.1 Model estimation mismatch

... happens if this constraint is "violated". The quadratic curve of the population model is still present in the interval [0, 1] but deviations from this interval lead to extreme values that cause the controlled system to behave more or less in an unstable sense. It is expected that this type of behavior will also occur in case noise deteriorates the measurements unless maybe the training is done with the residual information.

Another cure to narrow this "instability" could be to include a saturating nonlinearity in the feedback loop. The predicted values will in this way always be kept within the interval [0, 1] but it cannot be expected that this will lead to correct disturbance reduction. Experiment data is given in table 7.4.

Table 7.4: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Disturbance reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Show effect of additional saturating element</td>
</tr>
<tr>
<td>Data length</td>
<td>1000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 1.15z + 0.19$</td>
</tr>
<tr>
<td>Model</td>
<td>$1.2046z/z - 0.9537$</td>
</tr>
<tr>
<td>Neural network</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training</td>
<td>With chaotic signal</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed, [-2..2]</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model ($r = 4$)</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed, [-0.2..0.2]</td>
</tr>
<tr>
<td>Additional</td>
<td>Training done with real chaotic signal, not residual information</td>
</tr>
<tr>
<td></td>
<td>Extra saturating element included in feedback</td>
</tr>
</tbody>
</table>

Figure 7.7 shows what happens if a saturation is included in the feedback when the training is done with values for the population model between zero and one.
The previous sections describe disturbance reduction with training of the neural network with the chaotic disturbance itself. The following simulations are carried out with training of the network with the residue of the underestimated process order. Is the network able to separate the quite large error due to the model estimation mismatch from the chaotic disturbance in the output signal?

The process used here is

$$H_p(z) = \frac{z^2}{z^2 - 1.15z + 0.06} \quad (7.6)$$

In case the process is correctly estimated the residue contains mainly measurement noise and the chaotic disturbance. Correct predictions can be expected and disturbance reduction is possible in case the amount of noise is not too large. However, as seen in previous chapters, the underestimation of the process dynamics lead to residuals that grow quite large. In case the aforementioned process is underestimated the following model is obtained

$$H_m(z) = \frac{1.2057z}{z - 0.9523} \quad (7.7)$$

The dataset used was taken 10000 samples. Uniform distributed white noise ranging from $-2$ to $+2$ serves as reference signal and the same signal characteristics but values between $-0.1$ and $+0.1$ disturb the process output as measurement noise. The population model serves as chaotic disturbance. The transfer function of the residue is drawn in figure 7.8. From this data the neural network should separate the chaotic dynamics from the noise. No good results must be expected. All experiment data is given in table 7.5.
Table 7.5: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Disturbance reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Show residual</td>
</tr>
<tr>
<td>Data length</td>
<td>10000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$z^2/z^2 - 1.15z + 0.19$</td>
</tr>
<tr>
<td>Model</td>
<td>$1.2057z/z - 0.9523$</td>
</tr>
<tr>
<td>Neural network</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training</td>
<td>With chaotic signal</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed, $[-2..2]$</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model ($r = 4$)</td>
</tr>
<tr>
<td>Noise</td>
<td>ZMWN, uniform distributed, $[-0.1..0.1]$</td>
</tr>
<tr>
<td>Additional</td>
<td>Residual used to train network</td>
</tr>
</tbody>
</table>

Figure 7.8: Transfer function of the residual left over from underestimated process dynamics. Output is disturbed by measurement noise and a chaotic signal.

The network would have to predict future values of a real noise signal which is not really possible. After finding an optimum value for the weights in the network the cost function is still unacceptable. Simulations have been carried out but no useful results have been obtained.

7.1.2 Higher order disturbances

The disturbances in the previous subsection was considered of order one. The population model served as the disturbance additive to the output. The residue in this case could be fed to the neural network resulting in predictions of the disturbance in question. The extension to disturbances of higher order is quite simple. Depending upon the order of the (nonlinear) disturbance the residue must be delayed several times and these delayed values must serve as inputs to the neural network. Figure 4.3 gives an idea of how this can be done.
The disturbance used for this purpose is the Henon map (4.2). The number of delays can be restricted to one. Using the nomenclature of figure 1.2 this would mean that $\xi(k)$ and $\xi(k-1)$ (which is the delayed version of $\xi(k)$) are fed to the neural network. Figure 7.9 shows this idea.

Table 7.6: Experiment data

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Disturbance reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Show effect of disturbance reduction</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Henon map ($a = 1.4$, $b = 0.3$)</td>
</tr>
<tr>
<td>Data length</td>
<td>10000 samples</td>
</tr>
<tr>
<td>Process</td>
<td>$\frac{z^2}{z^2} - 0.5z + 0.06$</td>
</tr>
<tr>
<td>Model</td>
<td>$0.9977z^2/z^2 - 0.5025z - 0.0618$</td>
</tr>
<tr>
<td>Neural network</td>
<td>2 inputs, 6 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training</td>
<td>With residuals</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed, $[-2..2]$</td>
</tr>
<tr>
<td>Noise</td>
<td>Not added</td>
</tr>
<tr>
<td>Additional</td>
<td>Chaotic disturbance is of second order</td>
</tr>
</tbody>
</table>

Table 7.6 shows experiment information. Signal tracking and disturbance reduction is achieved as shown in figure 7.10.

Figure 7.10: Disturbance reduction with a small model estimation error. The disturbance is of order two. Reference (solid) and output with and without disturbance reduction (dashed and dotted, respectively)
Figure 7.9: Controlled setup for disturbance reduction. The chaotic disturbance is of higher order therefore an additional delay is added to the feedback to provide the network with the current and past values of the residue.
7.2 Measurement noise

As was said in the previous section, the deviation from the quadratic curve could lead to instability in the controlled system. The same behavior is expected in case the output measurements are obscured by sensor noise. The influence of this noise will be added to the residue (real output minus model output) and passed on to the neural network predictor. Again, cascading of neural networks for extending the prediction horizon will be limited by the addition of sensor noise.

The process to be controlled (5.8) now is disturbed by an additional noise term at the output (besides the chaotic disturbance). The model used is \( H_m(z) = \frac{1.0053z^2}{z^2 - 1.1399z + 0.1804} \). The controller is the inverse model \( (H_c = H_m^{-1}) \). The simulations done with this setup is carried out with a ZMWN sequence added to the output. Deviation from the desired chaotic residue will increase and predictions will no longer have proper value. The time signal of the output process is plotted along with the model reference trajectory that has to be tracked. The plot with reference and output signals is followed by a plot with error signals (the deviation from the reference signal). This gives a better insight to the tracking error. The amount of measurement noise added to the output is increased with each experiment. Input is a uniform distributed zero mean white noise signal with values between -2 and +2. The chaotic disturbance is the population model with parameter \( r = 4 \) and the neural network is trained with the residuals from the previous simulations. A network with four nodes in one hidden layer is used. The noise signal is also a zero mean white noise signal with uniform distribution. The amplitude of the signal varies in each simulation. The amplitude range is taken \([-0.05..+0.05]\), \([-0.1..+0.1]\), \([-0.2..+0.2]\) and \([-0.5..+0.5]\), respectively. Table 7.7 summarizes the data for the next four experiments.

<table>
<thead>
<tr>
<th>Experiment type</th>
<th>Disturbance reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Show effect of measurement noise</td>
</tr>
<tr>
<td>Disturbance</td>
<td>Population model ( (r = 4) )</td>
</tr>
<tr>
<td>Data length</td>
<td>500 samples</td>
</tr>
<tr>
<td>Process</td>
<td>( z^2/z^2 - 1.15z + 0.19 )</td>
</tr>
<tr>
<td>Model</td>
<td>( 1.0053z^2/z^2 - 1.1399z - 0.1804 )</td>
</tr>
<tr>
<td>Neural network</td>
<td>1 input, 4 hidden nodes, 1 output</td>
</tr>
<tr>
<td>Training</td>
<td>With residuals</td>
</tr>
<tr>
<td>Input signal</td>
<td>ZMWN, uniform distributed, ([-2..2])</td>
</tr>
<tr>
<td>Exp.1: Noise</td>
<td>ZMWN, uniform distributed, ([-0.05..0.05])</td>
</tr>
<tr>
<td>Exp.2: Noise</td>
<td>ZMWN, uniform distributed, ([-0.1..0.1])</td>
</tr>
<tr>
<td>Exp.3: Noise</td>
<td>ZMWN, uniform distributed, ([-0.2..0.2])</td>
</tr>
<tr>
<td>Exp.4: Noise</td>
<td>ZMWN, uniform distributed, ([-0.5..0.5])</td>
</tr>
</tbody>
</table>

The results of the complete setup is given in the following eight figures. The first figure of each simulation shows the reference signal and the output signals with and without predictive feedback. The second figure is made with the same data to show the influence of the measurement noise to the prediction quality. The error between the reference and output signal is drawn again for the setup with and without the neural network in the feedback loop.
Figures 7.11 to 7.18 show the output and error signals.

![Figure 7.11: Reference signal (solid) and output with (dashed) and without (dotted) predictive feedback. Process \( H_p(z) = \frac{z^2}{z^2 - 1.15z + 0.19} \) is disturbed by a chaotic signal and measurement noise (ZMWN, uniform distribution, \([-0.05..+0.05]\)). Model is taken \( H_m(z) = 1.0053z^2/z^2 - 1.1399z + 0.1804 \).](image)

![Figure 7.12: Tracking error between reference and output signal. Setup with predictive feedback (solid) and without (dotted). Measurement noise in the range \([-0.05..+0.05]\).](image)
Figure 7.13: Reference signal (solid) and output with (dashed) and without (dotted) predictive feedback. Process \( H_p(z) = z^2 / z^2 - 1.15z + 0.19 \) is disturbed by a chaotic signal and measurement noise (ZMWN, uniform distribution, \([-0.1.. + 0.1]\)). Model is taken \( H_m(z) = 1.0053z^2 / z^2 - 1.1399z + 0.1804 \).

Figure 7.14: Tracking error between reference and output signal. Setup with predictive feedback (solid) and without (dotted). Measurement noise in the range \([-0.1.. + 0.1]\).
7.2 Measurement noise

Figure 7.15: Reference signal (solid) and output with (dashed) and without (dotted) predictive feedback. Process \( H_p(z) = z^2/z^2 - 1.15z + 0.19 \) is disturbed by a chaotic signal and measurement noise (ZMWN, uniform distribution, \([-0.2..+0.2]\)). Model is taken \( H_m(z) = 1.0053z^2/z^2 - 1.1399z + 0.1804 \).

Figure 7.16: Tracking error between reference and output signal. Setup with predictive feedback (solid) and without (dotted). Measurement noise in the range \([-0.2..+0.2]\).
As shown in previous sections the neural network may be capable of capturing chaotic dynamics from a noisy signal. Even if the noise term is considerably large the chaotic term can be separated. However, this fact does not lead to good predictions if the noise term grows too large. Figures 7.11 to 7.18 show this. In case of the population model as chaotic disturbance the neural network approximates the quadratic curve but the residual information determines the future value. If the residue deviates considerably from the chaotic disturbance
the prediction of the disturbance is very likely not equal to the real chaotic disturbance at time instant $k + 1$.

Conclusions that can be drawn from the previous figures are that even for noise amplitudes ranging from $-0.2$ to $+0.2$ a slight improvement can be observed. For noise amplitudes ranging from $-0.5$ to $+0.5$ the residual deviates too much from the chaotic disturbance and feedback deteriorates the disturbance reduction.
Chapter 8

Conclusions and recommendations

8.1 Conclusions

The conclusions that can be drawn from this report are summed up in the same ordering as the chapters.

- Considering residual information and disturbances as being white noise cannot always be done. More insight in the dynamics of the processes and disturbances reveals that a lot of signals formerly considered to be stochastic can be taken (partially) deterministic. The chaotic dynamics have been proven to be a special case of dynamic behavior.

- The application of neural networks have been studied in the area of prediction and system identification. In both cases, good results have been obtained. The nonlinear properties of the neural network appear to be powerful in capturing and approximation of system dynamics.
  
  - Main drawback of the application of these neural network is the computational effort involved. Certainly in cases where on-line optimization is required powerful machinery has to be available.

  - Another drawback of the application of neural networks is the reproducibility that is very poor. The optimization is a very complicated task work, certainly when large neural networks are involved. The large amount of weights involved gives rise to a complicated cost function of which a global minimum is hard to find. However, optimization of a cost function in case of estimation of an Output Error model also involves complicated cost function. Each optimization described in this report is done with a randomly chosen initial weight vector. Consecutive optimizations will very likely lead to different solutions which can nevertheless yield good results. The complex architecture of the neural network makes solutions not unique.

- One of the properties of the chaotic signals was the sensitivity to initial conditions. The prediction of future value is therefore limited. As was shown in section 2.3 already after four cascaded neural networks large deviations from the desired trajectory could be observed. It must be noted that the limitation is also reached due to the non-ideal approximation of the neural network. More accurate approximation, if possible, either by a different neural network architecture or better optimization techniques will most
likely increase the number of predicted values, either by cascading neural networks or by introduction of a more complex network architecture. Measurements have limited accuracy which gives rise to divergence of nearby trajectories.

- Prediction of disturbances and negative feedback of these values can lead to reduced disturbances at the measured output. The objective of the setup drawn in figure 1.2 is that of signal tracking. Other configurations can be thought of. The feedback of the predicted disturbance has to be entered at another point in the controlled system.

  - Quality of the disturbance reduction is highly dependent on the estimation of the process. Mismatch of the process model results in residues that considerably deviate from the chaotic disturbance.

  - Measurement noise also affects the residue. Small amounts of noise are acceptable and influence the disturbance reduction but can still lead to reduction of the disturbance. In case the noise leads to small signal to noise ratios predictions of future disturbances can be considered meaningless and it has no use to apply the predictive feedback.

  - The neural network has been trained both with the chaotic process itself and with residual information. The first method is in fact only possible if this data is available. The latter is more sensible because measurements most often include the disturbance. It turns out that in case the amount of noise is not too large and uncorrelated the neural network is capable of capturing the chaotic dynamics and sorting out "real" noise. This more realistic setup also enlarges the range of sample data for the neural network training. If the network is capable of separating these two signals the chaotic dynamics are still present in the neural network but still no correct predictions can be made.

8.2 Recommendations

If future investigation is done in the area of chaotic disturbance reduction the following recommendations are given:

- This report describes only two types of chaotic disturbances. The population model and the Henon map serve as first and second order discrete time disturbances. Research still has to be done in the area of continuous time processes, sampling of these signals and the influence on the system control and learning of the neural network.

- Inverse modeling has so far been considered possible. The model $H_m$ is exactly inverted to achieve a unity transfer from reference to output. In practice, inverse problems will turn up, deteriorating the disturbance reduction. The discrete time processes used here could be inverted and in case model inversion would lead to non-causal controllers extra delays could be introduced resulting in proper disturbance reduction but in delayed signal tracking. It mainly depends on the demands of the system behavior whether these delays can be used or other solutions have to be found.

- In case the process to be controlled contains delays the prediction horizon has to be enlarged. A start has already been made with this setup but the results have not been described. Future work can easily be continued.
8.2 Recommendations

- No experiments have been done with real-life measurements. There will be a lot of experimental data available that could be used to train a neural network. As said in the introduction, weather is a splendid example of a chaotic system. However, the complexity of this system will probably lead to a very complex neural network and training will be time consuming. Application of DSP systems may be a solution.

- The design of a neural network architecture is a complicated task. Only feedforward single-layer networks have been used so far. Optimization is quite easy compared to multi-layer perceptrons. In fact, different neurons, connection layouts or activation functions can be applied. However, the application of a neural network as a tool is more important than to look for better performing networks.
References


