İzmir Institute of Technology

The Graduate School of Engineering and Natural Sciences

TRAJECTORY PREDICTION OF MOVING OBJECTS BY MEANS OF NEURAL NETWORKS

A Thesis in

Computer Engineering

by Hakan BARIŞIK

Submitted in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

June 1997

We approve the thesis of **Hakan BARIŞIK**

Date of Signature

09/07/1997

Prof. Dr. İ. Sıtkı AYTAÇ
Thesis Advisor
Chairman of Department of Computer Engineering

09/07/1997

Prof. Dr. Halis PÜSKÜLCÜ
Committee member
Department of Computer Engineering

09/07/1997

Prof. Dr. Şaban EREN
Committee member
Ege University
Department of Computer Engineering

ACKNOWLEDGEMENTS

I wish to express my sincerely gratitude to

- my advisor, Professor İ. Sıtkı Aytaç, for his constant support and trust.
- Prof. Dr. Halis Püskülcü, for his attention to equipment requirement of the study.
- Prof. Dr. Erol Emre, for encouraging me to study subject of motion prediction.
- Associate Professor Yusuf Öztürk, for the course of Advanced Artificial Neural Networks,
- Mr. Mete Eminağaoğlu for his work in edition of the thesis.
- my friends in IIT, for their helps ,
- Mr. Bahattin Tayanç, for maintaining necessary equipment and software for the study.
- Computer Center of İzmir Institute of Technology, for providing necessary computing environments and tools.

Abstract

Estimating the three-dimensional motion of an object from a sequence of object positions and orientation is of significant importance in variety of applications in control and robotics. For instance, autonomous navigation, manipulation, servo, tracking, planning and surveillance needs prediction of motion parameters. Although "motion estimation" is an old problem (the formulations date back to the beginning of the century), only recently scientists have provided with the tools from nonlinear system estimation theory to solve this problem. Neural Networks are the ones which have recently been used in many nonlinear dynamic system parameter estimation context. The approximating ability of the neural network is used to identify the relation between system variables and parameters of a dynamic system.

The position, velocity and acceleration of the object are estimated by several neural networks using the **n** most recent measurements of the object coordinates as input to the system. Several neural network topologies with different configurations are introduced and utilized in the solution of the problem. Training schemes for each configuration are given in detail. Simulation results for prediction of motion having different characteristics via different architectures with alternative configurations are presented comparatively.

Öz

Hareket halindeki bir üç boyutlu nesnenin zamana göre uzaydaki ardaşık çizgisel ve açısal konumlarından yararlanarak hareketinin tahmin edilmesi kontrol ve robotik uygulamaları açısından son derece önemlidir. Örneğin, seyir halindeki araçların otomatik yönlendirilmesi, süreç yönlendirme, hareketli nesnelerin takibi, planlama ve gözetleme gibi uygulamalar hareket parametrelerinin tahminine ihtiyaç duyarlar. Hareket kestirimi problemi eski bir problem olmakla birlikte, (formülasyonu bu yüzyılın başlarına dayanmaktadır) bilim adamları bu problemin çözümü için ancak son zamanlarda, doğrusal olmayan sistemler teorisi yardımıyla tasarlanmış araçlara kavuşabilmişlerdir. Yapay sinir ağları, doğrusal olmayan dinamik sistemlerin parametrelerinin kestirilmesinde kullanılan araçların en yenilerindendir.

Cisimlerin konum, hız ve ivmeleri bu hareket parametrelerinin önceki n sayıda ölçüm değerinin ayrı yapay sinir ağlarına girdi olarak verilmesiyle kestirilmektedir. Birkaç farklı yapay sinir ağ modelinin değişik düzenlemeri göz önüne alınarak tartışılmış ve problemin çözümünde nasıl kullanıldığı anlatılmıştır . Her düzenleme için yapay sinir ağlarının nasıl eğitildiğine ilişkin yöntemlerden ayrıntılı şekilde bahsedilmiştir. Farklı özellikteki hareketlerin değişik ağ modellerinin farklı düzenlemeriyle yapılan kestirimlerin başarımları karşılaştırılmalı olarak sunulmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Motivation

The problem of predicting the trajectory of moving objects is encountered in industrial robotics, servo systems and some aeronautical applications like target tracking where interaction with these moving parts is required. From the point of automatic systems' view, the trajectories of the parts are frequently unknown. For instance, parts that are moving on conveyor belt are tried to be grasped firmly to be placed into boxes. For a robot to grasp a part while the part is moving, it should be provided with the instant motion information of the part. The situation becomes more complicated when simple slide onto which products are thrown is used in the production line. On such a device, parts are free to rotate and deviate from perfect linear path. In such an application, the manipulator control system has to anticipate, at any instant, parts' position, orientation, velocity , and acceleration with the highest possible accuracy in order to plan an appropriate path for a manipulator to successfully grasp and pick up the part without collision or sliding.

Although a guided missile has capability to track a target motion, an intelligent pilot has the chance to deceive the missile chasing behind. For a guided missile prediction of the target behavior helps to make necessary maneuver to keep on the route to the target. If a missile is armed with an intelligent mechanism to identify the behavior of the target it is less probable to be deceived by the intelligent target .

The researches in the physics showed that many objects around us display chaotic behavior[Gulick,1992]. Chaotic dynamics of a phenomena can be extracted by different tools with of course different accuracy. Artificial neural networks are the ones which have recently been employed in many applications of this kind. The aim of this study is to give alternative solutions of trajectory prediction methods by

means of neural network having different architectures and configurations . Their learning capabilities on the same set of data are presented in comparative manner.

1.2 Motion Measurement and Prediction

Motion of an object is characterized by set of parameters called "motion parameters". These parameters give information both about the motion of the center of gravity of the object which is called transitional motion and object's rotational motion.

Both transitional motion and rotational motion have three main parameters; **position**, **velocity**, and **acceleration**. Rotational position of object is defined as "**orientation**". "**Angular velocity**" is the terminology used for rotational speed. In the case of rotational motion, change in rotational velocity in regard to time is defined as "**angular acceleration**".

Motion of center of gravity, namely transitional motion, can be described by the parameters which are position, velocity and acceleration.

Consequently, it can be said that provided with the motion parameters mentioned above, one have complete information about the motion of an object. In order to describe the motion in detail, each parameter must be defined with its x , y, z components in the three dimensional space. For instance, velocity can be given in form of triplet (*Vx, Vy, Vz*) where *Vx, Vy, Vz* are the components of the velocity vector in x, y, and z directions respectively. Transitional acceleration a has also components in x, y and z directions and denoted as ax, ay and az respectively.

The situation is no different for the rotational motion. Let an imaginary axis passing through an object is predefined such that this axis does not change in the course of motion. The angles between the defined axis and x-y plane , y-z plane and z-x plane represent the object's rotational position information simply named as orientation. Orientation has three components denoted as θxy, θyz, θzx. As the angles changes, orientation is said to be changing. The rate of change in each angle with respect to time is a component of angular velocity of object. Components of the

angular velocity are denoted as ω_{xy}, ω_{yz}, and ω_{zx}. Rate of change in each angular velocity components is the angular acceleration with possible indices xy, yz, zx.

As a total, a moving object has 9 transitional motion parameters at any instant of the motion. As for the rotational motion, 9 motion parameters can be counted just as in the case of rotational motion. In other words , number of parameter in the system is 18.

Any motion of sixth degrees of freedom needs 18 parameters to be estimated[Payeur et. al. , 1995]. The phrase "six degrees of freedom" refers to how many ways a body can move in space, including all translations and rotations.

The 6-DOF(degree of freedom) motion system can translate along and rotate around three axes: it can translate east-west, north-south, and up-down, and it can also rotate about those same axes. The translations and rotations are independent of each other. Six is the minimum number of degrees of freedom required to simulate the motion of a free-floating body in space.

Using image processing and pattern recognition techniques, an object can be recognized and its motion can be detected. If the the object is moving in a volume, its motion can be detected from the two series of frames recorded by two video cameras which are placed such that they are not face to face or not looking in the same direction. In other words, planes in which objects are viewed are not parallel to each other. This is the basic condition that needs to be satisfied for stereo-imaging. Alternatively, some different imaging techniques like radar-imaging can be used to obtain instantaneous position and orientation of an object. However, this study does not concern with the performance and use of imaging techniques. Rather it concentrates on predicting the value of motion parameters.

Estimation of motion parameters can be performed by the method outlined below.

Collect last n position (X_{0x}, X_{0y}, X_{0z} , X_{1x}, X_{1y}, X_{1z}, ... , Xn_x, Xn_y, Xn_z) and orientation information (θ_{0x}, θ_{0y}, θ_{0z} , θ_{1x}, θ_{1y}, θ_{1z}, ..., θn_x, θn_y, θn_z)

Calculate other motion parameters

Transitional motion V_x, V_y, V_z α_{xy}, α_{yz}, α_{zx}

Rotational motion $\omega_{xy}, \omega_{yz}, \omega_{zx}$ $\alpha_{xy}, \alpha_{yz}, \alpha_{zx}$

Estimate the next motion parameters using the estimation model .

1.3 Non-linear System Identification and Time Series Prediction

A system is the connected units or parts that form a whole and operate together [Lim Ni Fu, 1994]. The function of the system depends not only on the functions of parts constituting the system but also on how they connected to each other. Boundary separating the system from the rest of the world is referred as environment. An open system is a system that is affected by its environment, whereas a closed system is one isolated from its environment. The assumption of closed system under certain restrictions is a means to simplify the analysis of a system behavior[Lim Ni Fu, 1994].

System theory is the study of system behavior. The behavior refers to input/output characteristics and internal state changes. Mathematics is the language for expressing quantitative theory. Given a physical system, system identification aims to construct its mathematical model on the basis of a set of examples encoding the input/output behavior of the system.

Predicting the future in real-time (i.e. updating prediction "on the fly") has potential applications in such fields as adaptive control and system modeling. In fact, the problem of predicting the future is extremely general. The problem has potential application in any field that involves working with time series, i.e. with sequences of measurements or observations made on an unknown non-linear dynamic process [Chichocki and Unbehauen, 1993].

The most powerful approach to the problem of prediction is to find a law underlying the given dynamic process or phenomenon [Chichocki and Unbehauen, 1993]. If such a law can be discovered and analytically described, e.g. by a set of ordinary differential equations, then by solving them we can predict the future, if the initial conditions are completely specified. Unfortunately, the information about a dynamic process under investigation is often only partial and incomplete; so the prediction cannot be based on a known analytical model. The second less powerful approach is to attempt to discover some strong empirical regularities in the observation of the time series. For example, a time series consisting

of samples of a periodic process can be modeled by the superimposition of sinusoids generated by a set of second-order differential equations. Unfortunately, in many real-world problems some regularities such as periodicity are masked by noise and even some dynamic processes (phenomena) are described by chaotic time series in which the data containing hidden periodicity seem random [Chichocki and Unbehauen, 1993]. It must be stressed here that for chaotic time series long-term predictions are not possible, since the error (uncertainty of the prediction) increases exponentially in time. Note that for chaotic time series a long-term prediction is impossible even when the equations of motion are known exactly, because any error in the specification of the initial conditions will grow exponentially fast with time on average. Though chaos precludes any long-term predictability a short-time prediction is possible and very promising results have been obtained by using a neural network approach [Chichocki and Unbehauen, 1993].

1.4 Artificial Neural Networks

Given a physical system, a neural network can model it on the basis of a set of examples encoding the input/output behavior of the system. The neural network learns to map an input to a desired output by self-adaptation. The modeling capability of the neural networks can be described by its ability to learn the mathematical function underlying the system operation. If the network is designed and trained properly , it can perform generalization rather than curve fitting[Lim Ni Fu, 1994].

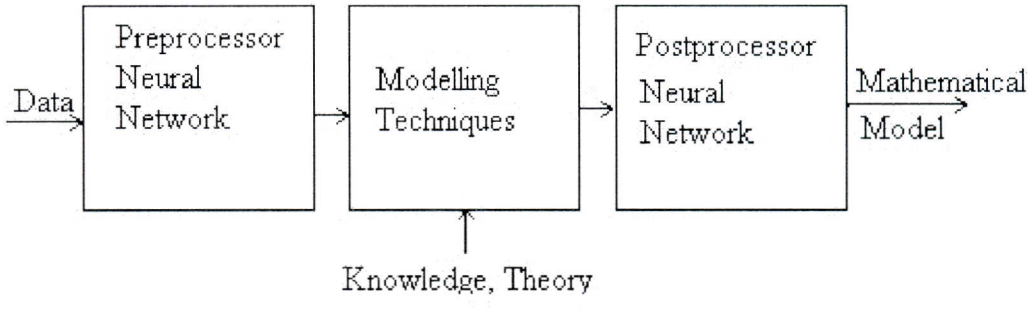


Figure 1.1 Neural networks in mathematical modeling

A neural network can be used in mathematical modeling in several ways:

- As a preprocessor
- As a post processor
- As a mathematical model
- As a baseline control

Figure 1.1 shows how neural networks are used in system modeling.

The use of neural networks as a system model is increasingly getting common. This is because a neural network can model a system by example mapping . If accuracy issue is concerned for a model, neural network may be a good one.

For the purpose of training, a set of training examples are selected from the domain. Each example is represented by ([x], [y]) where [x] and [y] vectors having dimensions m and n respectively. The goal is that after training,

$$[y] = f([x])=F_{NeuralNetwork}([x]) \qquad (1.1)$$

for every x in the domain, not limited to training set.

The main issue is generalization. In other words, the neural network is required to make generalization about the entire domain from the training examples. The design of the network architecture and training algorithm is aimed at improving this capability. Another related issue is overtraining, which refers to deterioration of the network performance against test examples beyond a certain number of training cycles. When the network is overtrained, it tends to fit the training data so closely but performance on the test data degrades [Lim Ni Fu, 1994].

An important theorem that illuminates the capability of multilayer neural networks as proven by Kolmogorov and is described in [Lorenz, 1976]. This theorem states that any continuos function can be represented in terms of nonlinear and continuously increasing functions of only one variable[Lim Ni Fu, 1994].

1.5 Contribution

In this study several hypothetical motion types have been studied and different neural network topologies have been used for predicting the motion

parameters for these motion types. Performance of each network model has been compared on the basis of prediction errors.

1.6 Organization

Chapter 2 : Basic theory of rigid body motion is presented.

Chapter 3 : Time series concept is explained. Besides, methods used to analyze and process time series data are presented.

Chapter 4 : Artificial neural network concept is introduced and fields in which artificial neural networks are used are listed. In addition, several famous neural network topology and their operation principles are presented .

Chapter 5: Theory of learning algorithms are explained mathematically.

Chapter 6: A literature survey on time series prediction and specifically motion prediction fields is given.

Chapter 7: Different motion types are analyzed. Experiments on estimation of these parameters with different neural network topologies are made and their performances are presented in comparative manner.

Chapter 8: The result of the study is summarized and works to be carried out in order to obtain a better performance are mentioned.

Chapter 2

MOTION OF A RIGID BODY

2.1 Introduction

A body can be subject to translational as well as rotational motion. In order to describe the motion of a rigid body at a time , one has to give parameter values for each motion type. "What are the characteristic parameters in each motion type?" and "What are relationships between these parameters?" are the questions that are going to be answered throughout the following sections.

2.2 Transitional Motion along a Straight Line

Motion of a point object along a straight line is the basics of more complicated motions. Translational motion on a plane and motions in a volume are the translational motions in higher dimensions. Motion in one dimension can be further generalized to motion in higher dimensions.

2.2.1 Position and Velocity of a Point Object along a Straight Line

Let A be a particle moving along Ox axis as in the Figure 2.1 . Position of the point object can be determined with respect to a reference point [Sears et. al., 1987]. In the figure below reference point is the origin denoted by O.

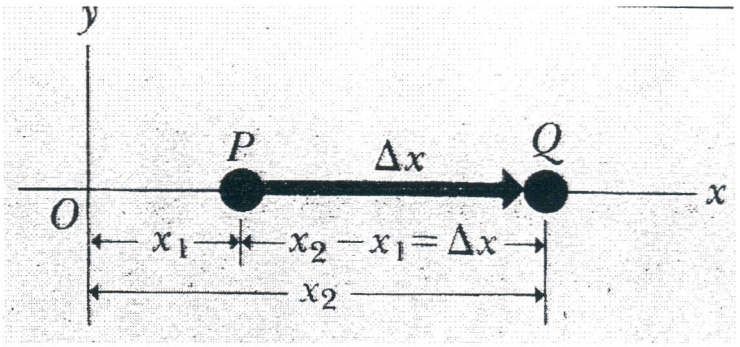


Figure 2.1 A Point Object moving on the x-axis

Position of a point object is defined as distance between the current point of an object and reference point. In the figure, the point P where the object is located at time t_1 has the distance of $|X_1|$ [1] unit from origin. At the time t_2 object has moved to the point Q which is $|X_2|$ units far from origin. That is, the object has the position X_2 at the time t_2. In this situation object is said to be undergone a displacement which is simply a position change and denoted as ΔX. Equation (2.1) shows how ΔX is obtained.

$$\Delta X = X_2 - X_1 \qquad (2.1)$$

The average velocity of the point object is defined as the ratio of the displacement ΔX to the time interval Δt, which is obtained as in equation (2.2).

$$\Delta t = t_2 - t_1 \qquad (2.2)$$

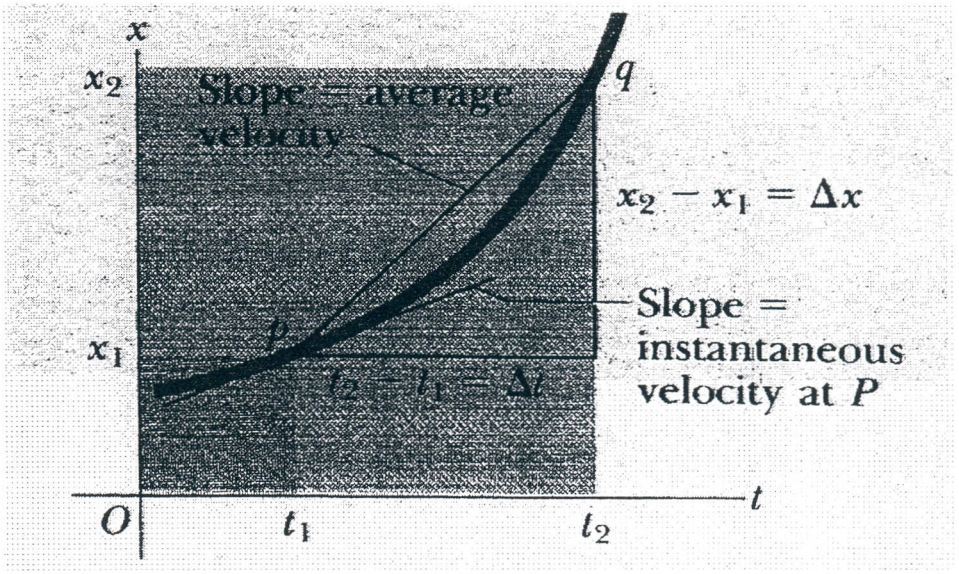


Figure 2.2 Coordinate-time graph of the motion in Figure 2.1

[1] $|X_1|$ denotes magnitude of the vector X_1.

The average velocity is represented by the letter V with subscript "av" to signify average value[Sears et. al., 1987]. Thus

$$v_{av} = \frac{x_2 - x_1}{t_2 - t_1} = \frac{\Delta x}{\Delta t} \qquad (2.3)$$

Even when the velocity of a moving particle varies, we can still define a velocity at any specific instant of time or at one specific point in the path. Such a velocity is called instantaneous velocity[Sears et. al., 1987]. Instantaneous velocity is defined as

$$v = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} = \frac{dx}{dt} \qquad (2.4)$$

2.2.2 Acceleration of Point Object Along a Straight Line

When the velocity of a moving object changes with time, it is said that object has an acceleration. Just as velocity is a quantitative description of the rate of change of position with time , so acceleration is quantitative description of the rate of change of velocity with time[Sears et. al., 1987].

Considering again the motion of a particle along x-axis, suppose that at time t_1 the particle is at point P and has velocity v_1, and that at a later time t_2 it is at point Q and has velocity v_2.

The average acceleration a_{av} of the particle as it moves from P to Q is defined as the ratio of the change in velocity to the elapsed time[Sears et. al., 1987].

$$a_{av} = \frac{v_2 - v_1}{t_2 - t_1} = \frac{\Delta v}{\Delta t} \qquad (2.5)$$

Instantaneous acceleration can be defined by following the same procedure used for instantaneous velocity[Sears et. al., 1987].

$$a = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt} \qquad (2.6)$$

The acceleration $a = dv/dt$ can be expressed in various ways. Since $v = dx/dt$, it follows that

$$a = \frac{dv}{dt} = \frac{d}{dt}\left(\frac{dx}{dt}\right) = \frac{d^2x}{dt^2}. \qquad (2.7)$$

2.3 Rotational Motion on a Plane

The motion of real-world bodies can be very complex. A body can have rotational as well as translational motions. In rotational motion angle is the fundamental unit of measurements. Position, velocity and acceleration which are translational motion quantities get the description "angular" in rotational motion domain. Detailed definitions of these quantities are given in the section below.

2.3.1 Orientation and Angular Velocity

Let A be a rigid body rotating about a stationary axis as in the *Figure 2.3*. Line OP is fixed in the body of and rotates with it . The angle between this line and horizontal line in the figure is θ in radians. Once the position of the axis of the rotation is known, θ describes the position of the body completely [Sears et. al., 1987]. Thus θ serves as a coordinate to describe the rotational position of the body. This has another name known as orientation of rigid body.

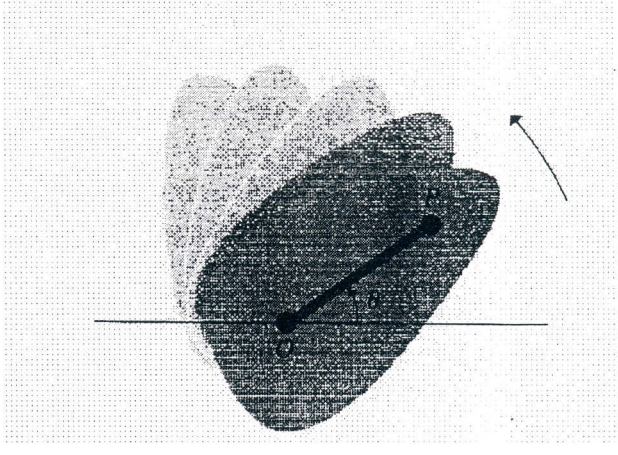


Figure 2.3 Body rotating about a fixed axis through point O.

Rotational motion of a body can be described in terms of the rate of the change of θ. In *Figure 2.4*, a reference line OP in a rotating body makes an angle θ_1 with the reference line Ox, at time t_1. At a later time t_2 the angle has changed to θ_2.

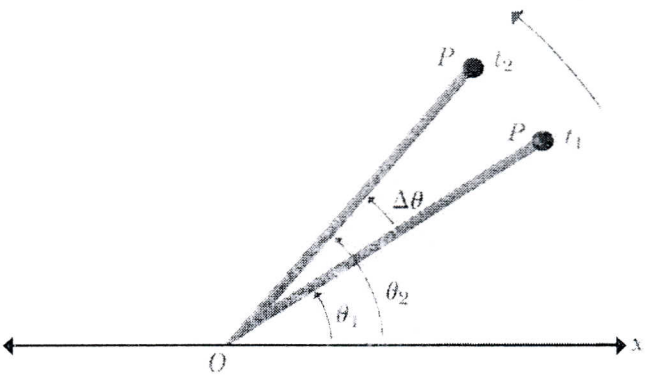


Figure 2.4 Angular displacement $\Delta\theta$ of a rotating body

We define **average angular velocity** of the body, ω_{av}, in the time interval $\Delta t = t_2 - t_1$, as the ratio of angular displacement $\theta_2 - \theta_1$ or $\Delta\theta$, to Δt [Sears et. al., 1987]:

$$\omega_{av} = \frac{\Delta\theta}{\Delta t} \qquad (2.8)$$

The **instantaneous angular velocity** ω is defined as the limit of this ratio as Δt approaches zero, that is, the derivative of θ with respect to t [Sears et. al., 1987]:

$$\omega = \lim_{\Delta t \to 0} \frac{\Delta\theta}{\Delta t} = \frac{d\theta}{dt} \qquad (2.9)$$

Because the body is rigid, all lines in it rotate through the same angle in the same time, and the angular velocity is the characteristic of the body as a whole. If the angle is in radians, the unit of angular velocity is one radian per second (1 rad . s^{-1} or 1 s^{-1}) [Sears et. al., 1987].

2.3.2 Angular Acceleration

When the angular velocity of a body changes, it has an angular acceleration. If ω_1 and ω_2 are the instantaneous angular velocities at times t_1 and t_2, we define the angular acceleration α_{av} as

$$\alpha_{av} = \frac{\omega_2 - \omega_1}{t_2 - t_1} = \frac{\Delta\omega}{\Delta t}, \qquad (2.10)$$

and the instantaneous angular acceleration α as the limit of ratio as $\Delta t \to 0$:

$$\alpha = \lim_{\Delta t \to 0} \frac{\Delta\omega}{\Delta t} = \frac{d\omega}{dt} \qquad (2.11)$$

The unit of angular acceleration is 1 rad.s^{-2} or 1 s^{-2}. Angular velocity and angular acceleration are exact analogous to linear velocity and acceleration. In each case, velocity is the time derivative of position, and acceleration is the time derivative of velocity.

Because $\omega = d\theta/dt$, the angular acceleration can also be written:

$$\alpha = \frac{d}{dt}\frac{d\theta}{dt} = \frac{d^2\theta}{d^2t} \qquad (2.12)$$

2.3 Transitional and Angular Motion in Three-Dimension

Three dimensional motion can be considered as three simultaneous transitional motion. From this point of view 3-D motion can be described by 3n parameters where n is the number of parameter describing a motion along a line.

2.4 Data Flow of Motion Parameters While Object is Moving in Three-Dimensional Space

When object is moving in space, motion parameters can be either measured or calculated using the measured quantities. Measuring fundamental motion parameter

is performed by means of different techniques. Stereo imaging via two video camera is a technique to measure displacement of object in space. By means of the same method, orientation difference can be extracted. The other motion parameters like average velocity and average acceleration in each direction for the duration of the interval can be calculated from this information.

Chapter 3

TIME SERIES

3.1 Introduction

Throughout scientific research, time series have been the basis for characterising an observed system and for predicting its future behaviour. A number of new techniques such as state-space reconstruction and neural networks promise insights that traditional approaches to these very old problems cannot provide.

Learning the motion behaviour of an object is nothing but the time series analysis of motion parameters. Thus, time series constitute the basis of this study. Powerful tools that can successfully extract the dynamics of a system have been used in this study.

This chapter gives basic properties of time series and mentions evolution of time series analysis methods.

3.2 History of Time Series Analysis

The desire to predict the future and understand the past, drives the search for the laws that explain the behavior of the observed phenomena. Examples range from the irregularity in a heartbeat to the volatility of a US $-TL exchange rate. If there are known underlying deterministic equation, in principle they can be solved to forecast the outcome of an experiment based on knowledge of initial conditions. In order to make prediction when the underlying equations explaining the system dynamics are not known, one must find the rules governing the system evolution and actual state of the system. For example, the motion of pendulum or the rhythm of the season are typical dynamic systems whose future behaviours can be predicted without insight into the underlying mechanism. The terms "understanding " and "learning" are used to refer to

system such as hydrodynamics flows and chemical reactions. In this context, this parabola is called logistic map or quadratic map. The value X_t depends on previous value X_{t-1}. λ is a parameter that controls the qualitative behaviour, ranging from a fixed point (for small values of λ) to deterministic chaos. For example, for $\lambda=4$, each iteration destroys one bit of information. Consider that, by plotting X_t against X_{t-1}, each value of X_t has two equally likely predecessors or equally well, the absolute average slope is two. Thus, if we know the location within $\in$ before the iteration, we will on average know it within $2\in$ afterwards This exponential increase in uncertainty is the sign of deterministic chaos .

Two crucial developments occurred around 1980; both were enabled by the general availability of powerful computers that permitted much longer time series to be recorded, more complex algorithms to be applied to them, and the data and the results of these algorithms to be interactively visualised. The first development , state space reconstruction by time-delay embedding, drew ideas from differential topology and dynamical systems to provide a technique for recognizing when a time series has been generated by deterministic governing equations and, if so, for understanding the geometrical structure underlying the observed behaviour. The second development was the emergence of the field of machine learning , typified by neural networks, that can adaptively explore a large space of potential models. With the shift in artificial intelligence from rule-based methods towards data-driven methods, the field was ready to apply itself to time series.

Global computer networks now offer a mechanism for disjoint communities to attack common problems through the widespread exchange of data and information. Santa Fe Time Series Prediction and Analysis Competition under the auspices of Santa Fe Institute was held during the fall of 1991. The goal was to provide a structure for researchers from the many relevant disciplines to compare quantitatively the result of their analyses of group of data sets. To explore the results of the competition, a NATO Advanced Research Workshop was held in spring of 1992.

3.3 Linear Time Series Models

Linear time series models have two particularly desirable features: they can be understood in detail and they are easy to implement. The drawback of the convenience is that they may not be unsuitable for even moderately complicated systems. Some of the most well-known models are given below.

3.3.1 Moving Average (MA) models

Given an external input series $\{e_t\}$, if someone wants to modify it to produce another series $\{x_t\}$, assuming linearity of the system and causality (the present value of x is influenced by the present and N past values of the input series e), the relationship between the input and output is

$$x_t = \sum_{n=0}^{N} b_n e_{t-n} = b_0 e_t + b_1 e_{t-1} + \cdots + b_N e_{t-N} . \qquad (3.2)$$

Statisticians and econometricians call this an N^{th} -order moving average model, MA (N). Engineers call this a finite impulse response (FIR) filter , because the output is guaranteed to go to zero at N time steps after the input becomes zero.

For a linear system , the response of the filter is independent of the input. A characterization focuses on properties of the system, rather than on properties of the time series. The series b_0 , ..., b_N is thus the impulse response of the system. The response of arbitrary input can be computed by superimposing the responses at arbitrary delays, weighted by the respective input values. Thus, the transfer function completely describes a linear system.

Sometimes it is more convenient to describe the filter in the frequency domain. This is useful because a convolution in the time domain becomes a product in frequency domain. If the MA model is an impulse (which has a flat power spectrum), the discrete Fourier transform of the output is given by[Weighend and Gershenfeld, 1994]

$$\sum_{n=0}^{N} b_n e^{-i2\pi nf} \qquad (3.3)$$

The power spectrum is given by the squared magnitude of this:

$$\left|1 + b_1 e^{-i2\pi 1f} + b_2 e^{-i2\pi 2f} + \cdots + b_N e^{-i2\pi Nf}\right|^2 \qquad (3.4)$$

The third way of representing yet again the same information is, in terms of the autocorrelation coefficients, defined in terms of the mean $\mu=\langle x_t \rangle$ and the variance $\sigma^2=E\{(x_t-\mu)^2\}$ by

$$\rho_\tau \equiv \frac{1}{\sigma^2} E\{(x_t - \mu)(x_{t-1} - \mu)\} \qquad (3.5)$$

where $E\{X\}$ means expected value of X.

The autocorrelation coefficients describe how much, on average, two values of a series that are τ time steps apart co-vary with each other. If the input to the system is a stochastic process with inputs values at different times uncorrelated, $\langle e_i e_j \rangle = 0$ for $i \neq j$, then all of the cross terms will disappear from the expectation value in equation 3.5, and resulting autocorrelation coefficients are

$$\sigma_\tau = \begin{cases} \dfrac{1}{\sum_{n=0}^{N} b_n^2} \sum_{n=\tau}^{N} b_n b_{n-\lfloor\tau\rfloor} & |\tau| \leq N \\ 0 & |\tau| > N \end{cases} \qquad (3.6)$$

3.3.2 Autoregressive (AR) Models

FIR filters operate in an open loop without feedback; they can only transform an input that is applied to them. If we do not want to drive the series externally, we need to provide some feedback (or memory) in order to generate internal dynamics:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + e_t . \qquad (3.7)$$

This is called M^{th} -order autoregressive model (AR(M)) or an infinite impulse response (IIR) filter. Depending on the applications, e_t can represent either a controlled input to the system or noise. As before, if e is white noise , the autocorrelations of the output series x can be expressed in terms of the model coefficients. Here, however, due to the

feedback coupling of previous steps, we obtain a set of linear equations rather than just a single equation for each autocorrelation coefficient. By multiplying equation (3.7) by $x_{t-\tau}$, taking expectation values, and normalizing the autocorrelation coefficients of an AR model are found by solving the set of linear equations, traditionally called Yule-Walker equations,

$$\rho_\tau = \sum_{m=1}^{M} a_m \rho_{\tau - m} , \quad \tau > 0 . \qquad (3.8)$$

Unlike th MA case, the autocorrelation coefficient need not vanish after M steps. Taking the Fourier transform of both sides of equation (3.6) and rearranging terms shows that the output equals the input times $\left(1 - \sum_{m=1}^{M} a_m e^{-i2\pi m}\right)^{-1}$. The power spectrum of the output is thus that of input times

$$\frac{1}{\left|1 - a_1 e^{-i2\pi 1 f} - a_2 e^{-i2\pi 2 f} - \cdots - a_M e^{-i2\pi M f}\right|^2} .$$

To generate a specific realization of the series, initial conditions must be specified, usually by the first M values of series x. Beyond that, the input term e_t is crucial for the life of an AR model.

The next step is to allow both AR and MA parts in the model; this is called an ARMA(M,N) model:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + \sum_{n=0}^{N} b_n e_{t-n} \qquad (3.9)$$

Its output is most easily understood in terms of z-transform, which generalizes the discrete Fourier transform to the complex plane:

$$X(z) \equiv \sum_{t=-\infty}^{\infty} x_t z^t \qquad (3.10)$$

On the unit circle , ($z=e^{-i2\pi f}$), the z-transform reduces to the discrete Fourier transform. Off the unit circle , z-transform measures the rate of divergence or convergence of a series. Since the convolution of two series in the time domain corresponds to the multiplication of their z-transforms, the z-transform of the output of an ARMA model is

$$X(z) = A(z)X(z) + B(z)E(z) \qquad (3.11)$$

$$= \frac{B(z)}{1 - A(z)} E(z)$$

The input z-transform E(z) is multiplied by transfer function that is unrelated to it; the transfer function will vanish at zeros of the MA term (B(z)=0) and diverge at poles (A(z)=1) due to the AR term. As A(z) is Mth-order complex polynomial, and B(z) is Nth-order, there will be M poles and N zeros. Therefore, the z-transform of a time series produced by equation (3.9) can be decomposed into a rational function and remaining part due to the input . The number of poles and zeros determines the number of degrees of freedom of the system (the number of the previous states that the dynamics retains). It should be noted that since only the ratio enters, there is no unique ARMA model. In the extreme cases, a finite-order AR model can always be expressed by an infinite-order MA model, and vice versa.

ARMA models have dominated all areas of time series analysis and discrete-time signal processing for than half a century . For example, in speech recognition and synthesis, Linear Predictive Coding compresses speech by transmitting the slowly varying coefficients for a linear model rather than the origin signal. If the model is good, it transforms the signal into a small number of coefficients plus residual white noise.

3.4 Fitting a Linear Model to Given Time Series

3.4.1 Fitting the coefficients.

Set of differential equations (3.8) permits to express the autocorrelation coefficients of a time series in terms of the AR coefficients that generated it. But there is a second reading of same equations: they also permits to estimate the coefficients as a regression problem: expressing the next value as a function of M previous values. This can be done by minimizing the squared errors: the parameters are determined such that the squared difference between the model output and the observed value, summed over all time steps in the fitting region, is as small as possible. There is no comparable conceptually simple expression for finding MA and full ARMA coefficients from

observed data. For all cases, however, standard techniques exist, often expressed as efficient recursive procedure.

Although there is no reason to expect that an arbitrary signal was produced by a system that can be written in the form of equation (3.9), it is reasonable to attempt to approximate a linear system's true transfer function (z-transform) by aratio of polynomials, i.e., an ARMA model. This is a problem of function approximation , it is well known that a suitable sequence of ratios of polynomials converges faster than a power series for arbitrary functions.

3.4.2 Selecting the order of the model

So far we have dealt with the question of how to estimate the coefficients from data for an ARMA model of order (M,N), but have not addressed the choice for the order of the model. There is not a unique best choice for the order of the model is increased, the fitting error decreases, but the test error of the forecast beyond the training set will usually start to increase at some point because the model will be fitting extraneous noise in the in the system. A simple approach is to hold back some of the training data and use these to evaluate the performance of competing models.

It is said that chaos is characterized by the sensitive dependence on the initial conditions. According to this feature it is suggested that long term of behavior of chaotic systems cannot be predicted. However it may be possible to forecast behavior locally after characterizing the deterministic rules of dynamic system from its complex behavior.

3.5 Nonlinear Time Series Prediction Models

Time series to be predicted mostly do not display linear characteristics. In such situations techniques or tools used in the prediction should be able to recognize nonlinear geometrical structure of the output produced by the system under consideration. In the following subsections models that are used to predict nonlinear dynamical structures are presented.

3.5.1 State Space Reconstruction

Yule's original idea for forecasting was that future predictions can be improved by using immediately predicted values. ARMA model, equation (3.9) can be rewritten as dot product between vectors of the time-lagged variables and coefficients:

$$x_t = A \cdot X_{t-1} + B \cdot E_t , \qquad (3.12)$$

where $X_t=(x_t, x_{t-1},\ldots, x_{t-(d-1)})$, and $A=(a_1, a_2,\ldots, a_d)$. These vectors are also called tapped delay lines.

In fact, there is a deep connection between time-lagged vectors and underlying dynamics. This connection was proposed in 1980 by [Weighend and Gershenfeld, 1994]. Delay vectors of sufficient length can recover the full geometrical structure of a nonlinear system. These results address the general problem of inferring the behaviour of intrinsic degrees of freedom of a system when a function of the state of the system is measured .

There are four relevant spaces and dimensions for the discussion of time series prediction:

1. The configuration space of the system is the space " where the equation live". It specifies the values of all the potentially accessible physical degrees of freedom of the system. For example, for a fluid governed by the Navier-Stokes partial differential equations, these are the infinite dimensional degrees of freedom associated with the continuous velocity, pressure, and temperature fields.
2. The solution manifold is where "the solution lives," that is, the part of the configuration space that the system actually explores as its dynamics unfolds. Due to unexcited or correlated degrees of freedom , this can be much smaller than the configuration space; the dimension of solution manifold is the number of parameters that are needed to uniquely specify a distinguishable state of overall system. For example, in some regimes, the infinite physical degrees of freedom of convecting fluid reduce to small set of coupled ordinary differential equations for a mode expansion.

3. The observable is a usually one dimensional function of the variables of configuration. In an experiment, this might be the temperature or a velocity component at a point in the fluid.
4. The reconstructed state space is obtained from that observable by combing past values of it to form a lag vector.

Given a time series measured from such a system- no other information about the origin of the time series- the question is what can be deduced about underlying dynamics.

Let y be state vector on the solution manifold, let dy/dt=f(y) be the governing equations, and let the measured quantity be x_t= x(y(t)). The result to be cited here also apply to systems that are described by iterated maps. Given a delay time τ and a dimension d, a lag vector can defined,

lag vector: $X_t=(x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau})$.

The central result is that the behavior of x and y will differ only by a smooth local inevitable change of coordinates. For almost every possible choice of f(y), x(y), and , τ as long as d is large enough, x depends on the at least some of the components of y, and the remaining components of y are coupled by the governing equations to the ones that influence X.

Time delay embedding differs from traditional experimental measurement in three fundamental respects:

1. It provides detailed information about the behavior of degrees of freedom other than the one that is directly measured.
2. It rests on probabilistic assumptions and - although it has been routinely and reliably used in practice-it is not guaranteed to be true for any system.
3. It allows precise questions only about quantities that are invariant under such a transformation, since the reconstructed dynamics have been modified by an unknown smooth change of coordinates.

Using embedding for forecasting appears to be very similar to Yule's original AR model :a prediction function is sought based on time-lagged vectors. The crucial difference is that understanding embedding reduces forecasting tom recognizing and representing the underlying geometrical structure, and once the number of lags exceeds the minimum embedding dimension, this geometry will not change. A global linear

model (AR) must do this with a single hyperplane. Since this may be a very poor approximation, there is no fundamental insight into how to chose the number of delays and related parameters.

3.5.2 Neural Networks

Theoretical work in connectionism ranges from reassuring proofs that neural networks with hidden units can essentially fit any well-behaved function and its derivative to results on the ability to generalize [Weighend, Gershenfeld, 1994]. These properties of neural networks are exploited to predict motion prediction of motion parameters in the way time series are predicted.

Theory of neural networks is going to be presented in detail in the following chapters.

Chapter 4

ARTIFICAL NEURAL NETWORKS

4.1 Introduction

Before discussing the solution methods of real world problems using neural networks, a definition of neural networks need to be presented. It is important to know the conditions under which this style of problem solving performs better and what its limitations are.

At the core of neural computation are the concepts of distributed, adaptive and nonlinear computing. Neural networks perform computation in a very different way than conventional computers, where a single central processing unit sequentially dictates every piece of the action. Neural networks are built from a large number of very simple processing elements that individually deal with pieces of a big problem. A processing element (PE) simply multiplies an input by a set of weights, and nonlinearly transforms the result into an output value (table lookup). The principles of computation at the PE level are deceptively simple. The power of neural computation comes from the massive interconnection among the PEs which share the load of the overall processing task, and from the adaptive nature of the parameters (weights) that interconnect the PEs.ss

Figure 4.1 illustrates a simple MLP. The circles are the PEs arranged in layers. The left row is the input layer, the middle row is the hidden layer, and the right row is the output layer. The lines represent weighted connections (i.e., a scaling factor) between PEs.

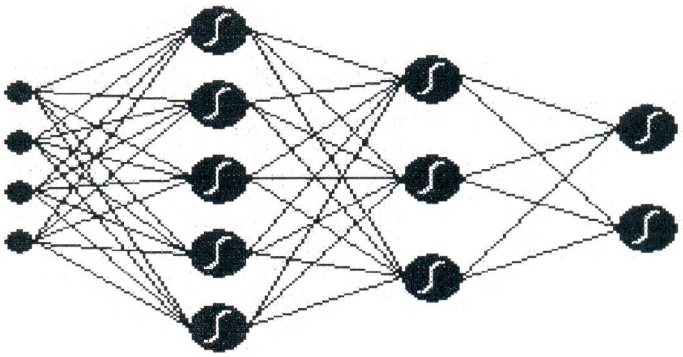


Figure 4.1 Simple multilayer perceptron

By adapting its weights, the neural network works towards an optimal solution based on a measurement of its performance. For supervised learning, the performance is explicitly measured in terms of a desired signal and an error criterion. For the unsupervised case, the performance is implicitly measured in terms of a learning law and topology constraints[Principe et. al., 1996].

4.2 Neural Network History

Neural Networks are an expanding and interdisciplinary field bringing together mathematicians, physicists, neurobiologists, brain scientists, engineers, and computer scientists. That is bringing a tremendous momentum to neural network research and creating many challenges.

Neural network theory started with the first discoveries about brain cellular organization, by Ramon Y. Cajal and Charles S. Sherrington [Principe et. al., 1996] at the turn of the century. The challenge was immediately undertaken to discover the principles that would make a complex interconnection of relatively simple elements to produce information processing at an intelligent level. This challenge is still with us today.

The work of the neuroanatomists has grown into a very rich field of science.. The work of McCulloch and Pitts [Principe et. al., 1996] on the modeling of the neuron as a threshold logic unit, and Caia-niello on neurodynamics merit special mention because they respectively led to the analysis of neural circuits as switching devices and as

nonlinear dynamic systems. More recently, brain scientists began studying the underlying principles of brain function and even implications to philosophy [Principe et. al., 1996].

The theoretical neurobiologists' work also interested computer scientists and engineers. The principles of computation involved in the interconnection of simple elements led to cellular automata, were present in Norbert Wiener's work on cybernetics and laid the ground for artificial intelligence [Principe et. al., 1996]. This branch is often referred to as artificial neural networks..

An initial goal in neural network development was modeling memory as the collective property of a group of processing elements [Principe et. al., 1996]. Caianiello, Grossberg and Amari studied the principles of neural dynamics. Rosenblatt created the perceptron for data driven (nonparametric) pattern recognition, and Fukushima proposed the neocognitron[Principe et. al., 1996]. Widrow's adaline (adaptive linear element) found applications and success in communication systems. Hopfield's analogy of computation as a dynamic process captured the importance of distributed systems. Rumelhart and McClelland's compilation of papers in the PDP (Parallel Distributive Processing) book, opened up the field for a more general audience. From the first International Joint Conference on Neural Networks held in San Diego, 1987, the field exploded [Principe et. al., 1996].

4.3 Neural Information Processing

4.3.1 Distinctive Properties and Application Areas of Artificial Neural Networks

Neural computation has a style. Unlike more analytically based information processing methods, neural computation effectively explores the information contained within input data, without further assumptions. Statistical methods are based on assumptions about input data ensembles (i.e. a priori probabilities, probability density functions, etc.). Artificial intelligence encodes a priori human knowledge with simple IF THEN rules, performing inference (search) on these rules to reach a conclusion. Neural networks, on the other hand "discover" relationships in the input data sets through the iterative presentation of the data and the intrinsic mapping characteristics of neural topologies (normally referred to as learning). There are two basic phases in neural

network operation. The training or learning phase where data is repeatedly presented to the network, while its weights are updated to obtain a desired response; and the recall or retrieval phase, where the trained network with frozen weights is applied to data that it has never seen. The learning phase is very time consuming due to the iterative nature of searching for the best performance. But once the network is trained, the retrieval phase can be very fast, because processing can be distributed.

In general, neural networks offer viable solutions when there are large volumes of data to train the neural network. When a problem is difficult (or impossible) to formulate analytically and experimental data can be obtained, then a neural network solution is normally appropriate.

The major applications of ANNs are the following:

Pattern classifiers: The necessity of a data set in classes is a very common problem in information processing. We find it in quality control, financial forecasting, laboratory research, targeted marketing, bankruptcy prediction, optical character recognition, etc. ANNs of the feedforward type, normally called multilayer perceptrons (MLPs) have been applied in these areas because they are excellent functional mappers (these problems can be formulated as finding a good input-output map).

Associative memories: Human memory principles seem to be of this type. In an associative memory, inputs are grouped by common characteristics, or facts are related. Networks implementing associative memory belong generally to the recurrent topology type, such as the Hopfield network or the bidirectional associative memory. However, there are simpler associative memories such as the linear or nonlinear feedforward associative memories.

Optimizers: The neural network model of computation proposed by Hopfield appears to have a variety of interesting applications, among which is the solution of combinatorial optimization problems. Some mechanisms are combined with Hopfield neural network model to improve its performance in optimization.

Feature extractors: This is also an important building block for intelligent systems. An important aspect of information processing is simply to use relevant information, and discard the rest. This is normally accomplished in a pre-processing stage. ANNs can be used here as principal component analyzers, vector quantizers, or

clustering networks. They are based on the idea of competition, and normally have very simple one layer topologies.

Dynamic networks: A number of important engineering applications require the processing of time-varying information, such as speech recognition, adaptive control, time series prediction, financial forecasting, radar/sonar signature recognition and nonlinear dynamic modeling. To cope with time varying signals, neural network topologies have to be enhanced with short term memory mechanisms. This is probably the area where neural networks will provide an undisputed advantage, since other technologies are far from satisfactory. This area is still in a research stage.

It should be noticed that a lot of real world problems fall in this category, ranging from classification of irregular patterns, forecasting, noise reduction and control applications. Humans solve problems in a very similar way. They observe events to extract patterns, and then make generalizations based on their observations.

4.3.2 Basic Operation Principles of Neural Networks

Biologial neurons transmit electrochemical signals over neural pathways. Each neuron receives signals from other neurons through special junctions called synapses. Some inputs tend to excite the neuron; others tend to inhibit it. When the cumulative effect exceeds a threshold, the neuron fires and sends a signal down to other neurons . An artificial neuron receives a set of inputs . Each input is multiplied by a weight analogous to a synaptic strength.. The sum of all weighted inputs determine the degree of firing called the activation level[Makhoul, 1992]. Notationally, each input X_i is modulated by weight W_i and the total output is expressed as $\sum_i X_i W_i$ or in vector form, **X.W** where $\mathbf{X}=[X_1, X_2, \ldots, X_n]$ and $\mathbf{W}=[W_1, W_2, \ldots, W_n]$. This is depicted in the Figure 4.2.

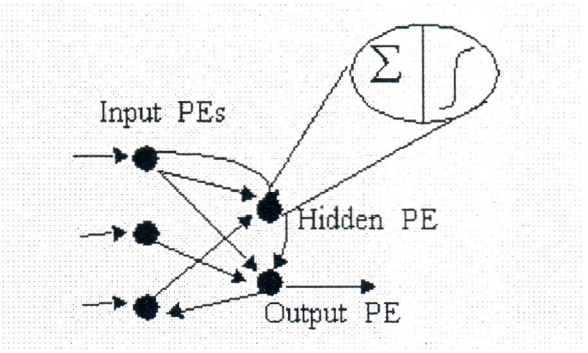
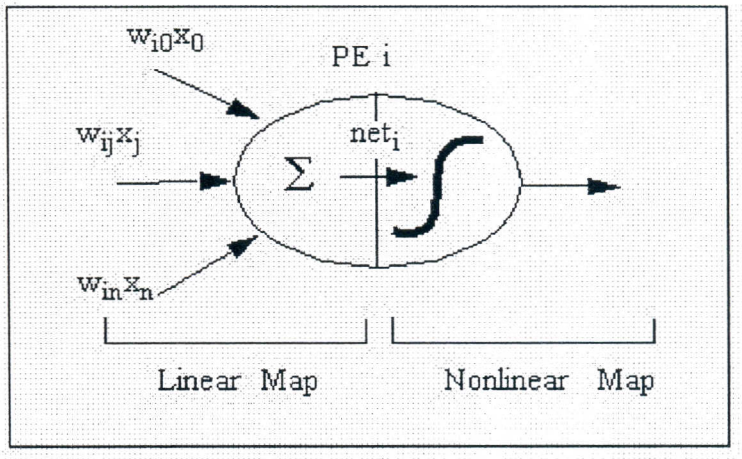


Figure 4.2.a. The building blocks of artificial neural networks.



X_j are the inputs to unit j.

X_i is the output of unit i.

Wij are the weights that connect unit j to unit i.

Ψ is a nonlinearity.

Figure 4.2.b The building blocks of artificial neural networks.

The input signal is further processed by an activation function to produce the output signal. The activation function can be threshold function or or smooth function . Some of the famous activation functions are ploted in Figure 4.3.

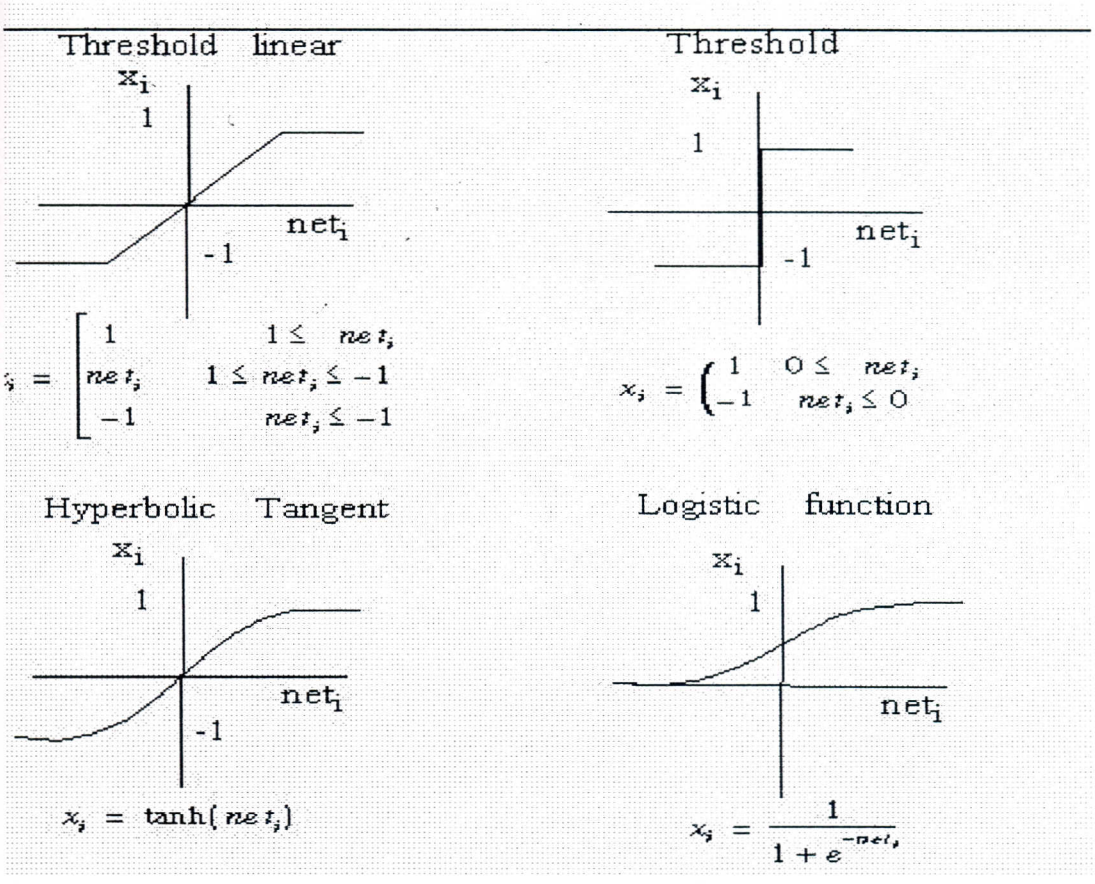


Figure 4.3 Some commonly used activation functions

4.3.3 Distributed Computation via Neural Networks

Distributed computation has the advantages of reliability, fault tolerance, high throughput (division of computation tasks) and cooperative computing, but generates problems of locality of information, and the choice of interconnection topology.

Adaptation is the ability to change a system's parameters according to some rule (normally, minimization of an error function). Adaptation enables the system to search for optimal performance, but adaptive systems have trouble responding in a repeatable manner to absolute quantities.

Nonlinearity is a blessing in dynamic range control for unconstrained variables and produces more powerful computation schemes (when compared to linear processing) such as feature separation. However, it complicates theoretical analysis tremendously.

These features of distributed processing, adaptation and nonlinearity are the hallmark of biological information processing systems. ANNs are therefore working with the same basic principles as biological brains, but probably the analogy should stop here. We are still at a very rudimentary stage of mimicking biological brains, due to the rigidity of the ANN topologies, restriction of PE dynamics and timid use of time (time delays) as a computational resource.

4.4 Learning Pradigms

The process of modifying network parameters to improve performance is normally called learning. Learning in ANN's can also be thought of as a second set of dynamics, because the network parameters will evolve in time according to some rules. Consider the 6 PE MLP in A Multilayer Perceptron and its Weight Matrix figure as depicted in Figure 4.4. Assume that inputs are currents, and the weights are potentiometers that the user can control. For this example, the PEs can be thought of simply as being transistors. The goal is to obtain a value of 1 volt at the output, when several different currents are presented to the network. What the user would do is the following: start by connecting one of the inputs, and check the value at the output. If the value is not 1 volt, then some of the potentiometers will have to be changed until the goal state is reached. Then a second input is presented, and the process repeated until the desired response is obtained for all the inputs. When a neural network is trained, this very process of changing the weights is automated.

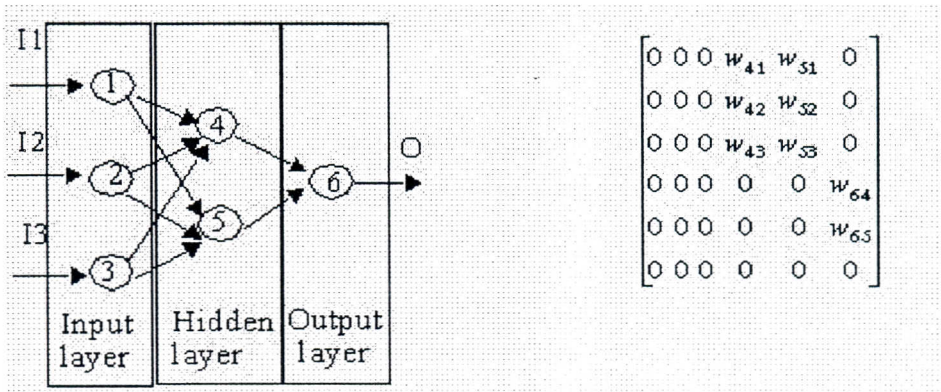


Figure 4.4 A multilayer perceptron and its weight matrix

Learning requires several ingredients. First, as the network parameters change, the performance should improve. Therefore, the definition of a measure of performance is required. Second, the rules for changing the parameters should be specified. Third, this procedure (of training the network) should be done with known data.

The application of a performance measure produces another important taxonomic division in ANNs. When the performance function is based on the definition of an error measure, learning is said to be supervised. Normally the error is defined as the difference of the output of the ANN and a pre-specified external desired signal. In engineering applications where the desired performance is known, supervised learning paradigms become very important.

The other class of learning methods modify the network weights according to some pre-specified internal rules of interaction (unsupervised learning). There is therefore no "external teacher". This is the reason unsupervised learning is also called self-organization. Self-organization may be very appropriate for feature discovery (feature extraction) in complex signals with redundancy. A third intermediate class of learning is called reinforcement learning. In reinforcement learning the external teacher just indicates the quality (good or bad) of the response. Reinforcement learning is still in a research phase, but it may hold the key to on-line learning (i.e. with the present sample).

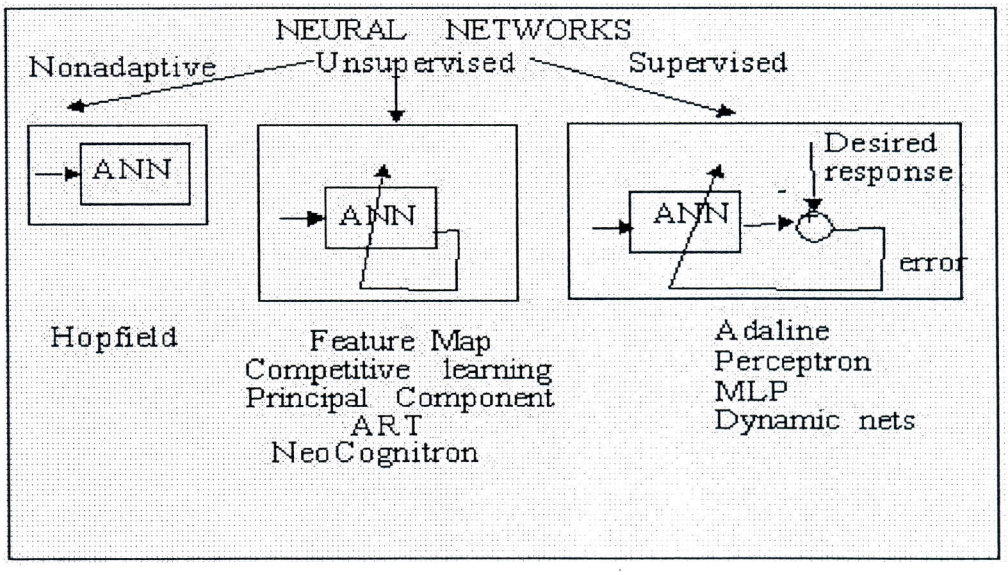


Figure 4.5 A taxonomy for artificial neural networks

For the class of supervised learning there are three basic decisions that need to be made: choice of the error criterion, how the error is propagated through the network,

and what constraints (static or across time) one imposes on the network output. The first issue is related to the formula (the cost function) that computes the error. The second aspect is associated with mechanisms that modify the network parameters in an automated fashion. After some investigation one can easily notice that gradient descent learning is the most common in supervised learning schemes. The third aspect is associated with how we constrain the network output versus the desired signal. One can specify only the behavior at the final time (fixed point learning), i.e. we do not constrain the values that the output takes to reach the desired behavior. Or, the intermediate values can be constrained and trajectory learning can be performed. Note that a feedforward network, since it is an instantaneous mapper (the response is obtained in one time step), can only be trained by fixed point learning. Recurrent networks, however, can be trained by specifying either the final time behavior (fixed point learning) or the behavior along a path (trajectory learning).

Learning requires the specification of a set of data for training the network. This is normally called the training set. Learning performance should be checked against a different set of data called the test set. It is of fundamental importance to choose an appropriate training set size, and to provide representative coverage of all possible conditions. During learning, the network is going to discover the best mapping between the input data and the desired performance. If the data used in the training set is not representative of the input data class, we can expect poor performance with the test set, even though performance can be excellent with the training set.

4.5 Famous Neural Network Topologies

4.5.1 Perceptron Based Topologies

4.5.1.1 Perceptron

The perceptron was probably the first successful neurocomputer. Rosenblatt [Principe et. al., 1996] constructed the MARK I for binary image classification. The perceptron is nothing but a feedforward neural network with no hidden units. Its information processing abilities are limited. It can only discriminate among linearly separable classes, i.e. classes that could be separated by hyperplanes. The appeal of the perceptron was Rosenblatt's proof that it is trainable (for linearly separable classes) in a finite number of steps. The perceptron learning rule is very simple:

Present a pattern. If the output is the desired output, do nothing. If the response is wrong, from the units that are active, change their weights towards the desired response. Repeat the process until all the units have acceptable outputs.

The delta rule (simplified backpropagation) can also be applied to the perceptron, but perceptron learning is faster and more stable if the patterns are linearly separable. Even today (more than 30 years later), the perceptron and its learning rule are still popular. Recently the perceptron learning rule was revisited to provide acceptable results even when the patterns were not linearly separable.

4.5.1.2 Multilayer Perceptron (MLP)

The multilayer perceptron (MLP) is one of the most widely implemented neural network topologies. Generally speaking, for static pattern classification, the MLP with two hidden layers is a universal pattern classifier. In other words, the discriminant functions can take any shape, as required by the input data clusters. Moreover, when the weights are properly normalized and the output classes are normalized to 0/1, the MLP achieves the performance of the maximum a posteriori receiver, which is optimal from a classification point of view [Makhoul, 1992]. In terms of mapping abilities, the MLP is believed to be capable of approximating arbitrary functions. This has been important in

the study of nonlinear dynamics [Principe et. al., 1986], and other function mapping problems.

MLPs are normally trained with the backpropagation algorithm [Principe et. al., 1986]. In fact the renewed interest in ANNs was in part triggered by the existence of backpropagation. The LMS learning algorithm proposed by Widrow[Principe et. al., 1986] can not be extended to hidden PEs, since we do not know the desired signal there. The backpropagation rule propagates the errors through the network and allows adaptation of the hidden PEs.

Two important characteristics of the multilayer perceptron are: its nonlinear processing elements (PEs) which have a nonlinearity that must be smooth (the logistic function and the hyperbolic tangent are the most widely used); and their massive interconnectivity (i.e. any element of a given layer feeds all the elements of the next layer).

The multilayer perceptron is trained with error correction learning, which means that the desired response for the system must be known. In pattern recognition this is normally the case, since we have our input data labeled, i.e. we know which data belongs to which experiment.

The backpropagation learning rule based error correction is going to be presented in the following chapter in detail.

4.5.2 Madaline

Madaline is an acronym for multiple adalines, the ADAptive LINear Element proposed by Widrow. The adaline is nothing but a linear combiner of static information, which is not very powerful. However, when extended to time signals, the adaline becomes an adaptive filter of the finite impulse response class. This type of filter was studied earlier by Wiener. Widrow's contribution was the learning rule for training the adaptive filter. Instead of numerically solving the equations to obtain the optimal value of the weights, Widrow proposed a very simple rule based on gradient descent learning (the least mean square rule LMS). The previous adaptive theory was essentially statistical (it required expected value operators), but Widrow took the actual value of the product of the error at each unit and its input as a rough estimate of the gradient. It turns out that

this estimate is noisy, but unbiased, so the number of iterations over the data average the estimate and make it approach the true value[Principe et. al., 1996].

The adaptive linear combiner with the LMS rule is one of the most widely used structures in adaptive signal processing. Its applications range from echo cancellation, to line equalization, spectral estimator, beam former in adaptive antennas, noise canceller, and adaptive controller. The adaline is missing one of the key ingredients for our definition of neural networks (nonlinearity at the processing element), but it possesses the other two (distributed and adaptive).

4.5.3 Radial Basis Function Network

Radial basis functions networks have a very strong mathematical foundation rooted in regularization theory for solving ill-conditioned problems. Suppose that we want to find the map that transforms input samples into a desired classification. Due to the fact that only have a few samples are present, and that they can be noisy, the problem of finding this map may be very difficult (mathematicians call it ill-posed). The mapping is performed by decreasing the error between the network output and the desired response, also a constraint relevant to our problem is aded . Normally this constraint is smoothness.

One can show that such networks can be constructed in the following way: Bring every input component (p) to a layer of hidden nodes. Each node in the hidden layer is a p multivariate Gaussian function

$$G(x:x_i) = e^{-\frac{1}{2\sigma^2}\sum_{k=1}^{p}(x_k - x_{ik})^2} \qquad (4.1)$$

of mean (each data point) and variance . These functions are called radial basis functions[Chichocki and Unbehauen, 1993]. Finally, linearly weight the output of the hidden nodes to obtain

$$F(x) = \sum_{j=1}^{N} w_j (G(x:x_j)) \qquad (4.2)$$

The problem with this solution is that it may lead to a very large hidden layer (the number of samples of your training set).

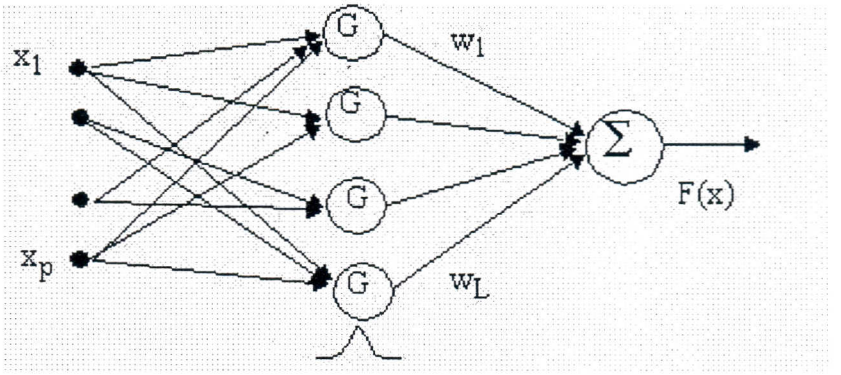


Figure 4.6 Radial Basis Function (RBF) network

This solution can be approximated by reducing the number of PEs in the hidden layer, but cleverly position them over the input space regions, i.e. where we have more input samples. This means that we have to estimate the positions of each radial basis function and its variance (width), as well as compute the linear weights w_i .

4.5.3.1 Estimation of the Centers and Widths

The most widely used method of estimating the centers and widths is to use an unsupervised technique called the k-nearest neighbor rule. The input space is first discretized into k clusters and the size of each is obtained from the structure of the input data. The centers of the clusters give the centers of the RBFs, while the distance between the clusters provide the width of the Gaussians. The definition of the width is nontrivial. Each width is set proportional to the distance between the center and its nearest neighbor. Conscience can be used to make sure that all the RBF centers are brought into the data clusters. However, conscience also brings the problem of confining the centers too close together. Scheduling of the conscience may be necessary for a good coverage of the data clusters.

4.5.3.2 Computing the Output Weights

The output weights in turn are obtained through supervised learning. The error correction learning described in the multilayer perceptron section is normally used, but this problem is easier because the output unit is normally linear, so convergence is faster. In practical cases, an MLP can be superior to the linear network, because it may take advantage of nonlinearly separable data clusters produced by too few RBFs.

4.5.4 Associative Memories

Steinbuch was a cognitive scientist and one of the pioneering researchers in distributed computation. His interests were in associative memories, i.e. devices that could learn associations among dissimilar binary objects. He implemented the learnmatrix, where a set of binary inputs is fed to a matrix of resistors, producing a set of binary outputs. The outputs are 1 if the sum of the inputs is above a given threshold., zero otherwise. The weights (which were binary) were updated by using several very simple rules based on Hebbian learning. But the interesting thing is that the asymptotic capacity of this network is rather high and easy to determine [Principe et. al., 1996].

The linear associative memory was proposed by several researchers [Principe et. al., 1996]. It is a very simple device with one layer of linear units that maps N inputs (a point in N dimensional space) onto M outputs (a point in M dimensional space). In terms of signal processing this network does nothing but a projection operation of a vector in N dimensional space to a vector in M dimensional space.

This projection is achieved by the weight matrix. The weight matrix can be computed analytically: it is the product of the output with the pseudo inverse of the input [Principe et. al., 1996]. In terms of linear algebra, what we are doing is computing the outer product of the input vector with the output vector. This solution can be approximated by Hebbian learning and the approximation is quite good if the input patterns are orthogonal. Widrow's LMS rule can also be used to compute a good approximation of W even for the case of non-orthogonal patterns [Principe et. al., 1996]

4.5.5 Jordan/Elman Network

The theory of neural networks with context units can be analyzed mathematically only for the case of linear PEs. In this case the context unit is nothing but a very simple lowpass filter. A lowpass filter creates an output that is a weighted (average) value of some of its more recent past inputs. In the case of the Jordan context unit, the output is obtained by summing the past values multiplied by the scalar τ^n as shown in the figure below.

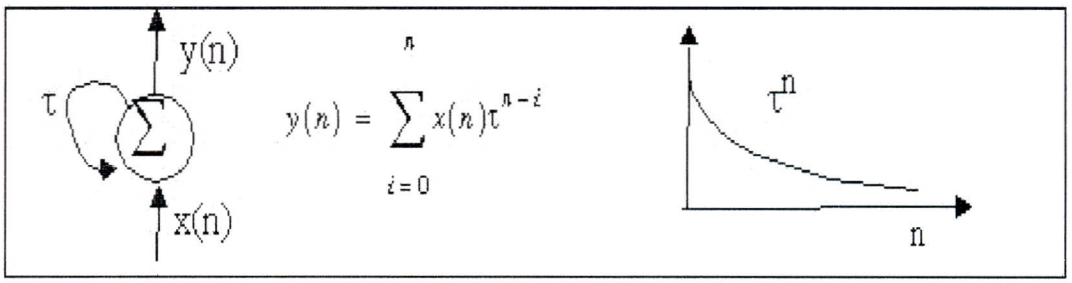


Figure 4.7 Context unit response

Notice that an impulse event x(n) (i.e. x(0)=1, x(n)=0 for n>0) that appears at time n=0, will disappear at n=1. However, the output of the context unit is t1 at n=1, t2 at n=2, etc. This is the reason these context units are called memory units, because they "remember" past events. t should be less than 1, otherwise the context unit response gets progressively larger (unstable).

The Jordan[Jordan, 1986] network and the Elman[Elman, 1990] network combine past values of the context units with the present inputs to obtain the present net output. The input to the context unit is copied from the network layer, but the outputs of the context unit are incorporated in the net through adaptive weights. One issue in these nets is that the weighting over time is kind of inflexible since we can only control the time constant (i.e. the exponential decay). Moreover, a small change in τ is reflected in a large change in the weighting (due to the exponential relationship between time constant and amplitude). In general, how large the memory depth should be is not known, so this makes the choice of τ problematic, without a mechanism to adapt it.

In linear systems the use of the past of the input signal creates what is called the moving average (MA) models. They represent well signals that have a spectrum with sharp valleys and broad peaks. The use of the past of the output creates what is called the autoregressive (AR) models. These models represent well signals that have broad valleys and sharp spectral peaks. In the case of nonlinear systems, such as neural nets, these two topologies become nonlinear (NMA and NAR respectively). The Jordan[Jordan, 1986] net is a restricted case of an NAR model, while the configuration with context units fed by the input layer are a restricted case of NMA. Elman's[Elman, 1990] net does not have a counterpart in linear system theory. As can be understood from this simple discussion, the supported topologies have different processing power, but the question of which one performs best for a given problem is left to experimentation.

4.5.6 Hopfield Neural Network

The Hopfield network is a recurrent neural network with no hidden units, where the weights are symmetric ($w_{ij}= w_{ji}$). The PE is an adder followed by a threshold nonlinearity. The model can be extended to continuous units. The processing elements are updated randomly, one at a time, with equal probability (synchronous update is also possible). The condition of symmetric weights is fundamental for studying the information capabilities of this network. It turns out that when this condition is fulfilled the neurodynamics are stable in the sense of Lyapunov, which means that the state of the system approaches an equilibrium point. With this condition Hopfield was able to explain to the rest of the world what the neural network is doing when an input is presented. The input puts the system in a point in its state space, and then the network dynamics (created by the recurrent connections) will necessarily relax the system to the nearest equilibrium point (point P1 in the figure below) [Principe et. al., 1996].

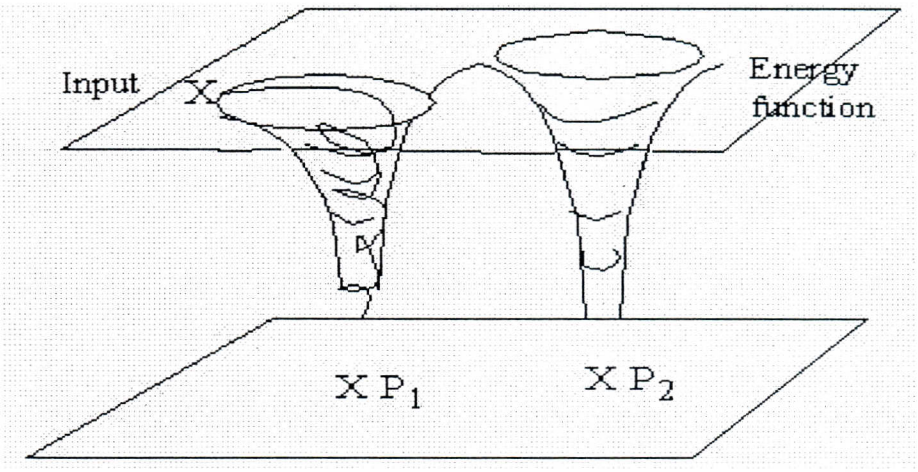


Figure 4.8 Relaxation to the nearest fixed point

If the equilibrium points were pre-selected (for instance by hardcoding the weights), then the system could work as an associative memory. The final state would be the one closest (in state space) to that particular input. We could then classify the input or recall it using content addressable properties. In fact, such a system is highly robust to noise, also displaying pattern completion properties. Probably, biological memory is based on identical principles. The structure of the hippocampus is very similar to the wiring of a Hopfield net (outputs of one unit fed to all the others). In a Hopfield net if one asks where the memory is, the answer has to be in the set of weights. The Hopfield net therefore implements a nonlinear associative memory, which is known to have some of the features of human memory (e.g. highly distributed, fault tolerance, graceful degradation, finite capacity).

Most Hopfield net applications are in optimization, where a mapping of the energy function to the cost function of the user's problem must be established and the weights need to be pre-computed. The weights in the Hopfield network can be computed using Hebbian learning, which guarantees a stable network. Recurrent backpropagation can also be used to compute the weights, but in this case, there is no guarantee that the weights are symmetric (hence the system may be unstable).

4.5.7 Principle Component Analysis Neural Network

The fundamental problem in pattern recognition is to define data features that are important for the classification (feature extraction). One wishes to transform our input samples into a new space (the feature space) where the information about the samples is retained, but the dimensionality is reduced. This will make the classification job much easier.

Principal component analysis (PCA) also called Karhunen-Loeve transform of Singular Value Decomposition (SVD) is a technique to perform the task mentioned above. PCA finds an orthogonal set of directions in the input space and provides a way of finding the projections into these directions in an ordered fashion. The first principal component is the one which has the largest projection (we can think that the projection is the shadow of our data cluster in each direction). The orthogonal directions are called the eigenvectors of the correlation matrix of the input vector, and the projections the corresponding eigenvalues[Principe et. al., 1996].

Since PCA orders the projections, we can reduce the dimensionality by truncating the projections to a given order. The reconstruction error is equal to the sum of the projections (eigenvalues) left out. The features in the projection space become the eigenvalues. Note that this projection space is linear.

PCA is normally done by analytically solving an eigenvalue problem of the input correlation function. However, Sanger and Oja demonstrated that PCA can be accomplished by a single layer linear neural network trained with a modified Hebbian learning rule[Principe et. al., 1996].

Let us consider the network shown in the figure below. Notice that the network has p inputs (we assume that our samples have p components) and m<p linear output PEs. The output is given by

$$y_j(n) = \sum_{i=0}^{p-1} w_{ij}(n) x_i(n), \quad j = 0,1,\cdots,m-1 \qquad (4.3)$$

To train the weights, the following modified Hebbian rule is used

$$\Delta w_{ji}(n) = \eta\left[y_i(n)x_i(n) - y_i(n)\sum_{k=0}^{j} w_{ki}(n)y_k(n) \right] \begin{matrix} i = 0,\cdots,p-1 \\ j = 0,\cdots,m-1 \end{matrix} \qquad (4.4)$$

where η is the step size.

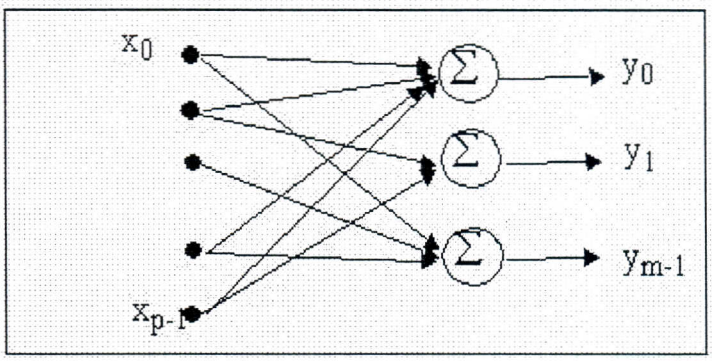


Figure 4.9 PCA network

What is interesting in this network is that the eigenvectors of the correlation function of the input are computed without ever computing the correlation function. Sanger showed that this learning procedure converges to the correct solution, i.e. the weights of the PCA network approach the first m principal components of the input data matrix. The outputs are therefore related to the eigenvalues and can be used as input to another neural networks for classification[Principe et. al., 1996].

PCA networks can be used for data compression, providing the best m linear features. They can also be used for data reduction in conjunction with multilayer perceptron classifiers. In this case, however, the separability of the classes is not always guaranteed. If the data clusters are sufficiently separated, yes, but if the classes are on top of each other, the PCA will get the largest projections, but the separability can be in some of the other projections. Another problem with linear PCA networks are outlying data points. Outliers will distort the estimation of the eigenvectors and create skewed data projections. Nonlinear networks are better able to handle this case.

The importance of PCA analysis is that the number of inputs for the MLP classifier can be reduced a lot, which positively impacts the number of required training patterns, and the training times of the classifier.

4.5.8 Kohonen Self-Organizing Feuture Map

As was stated previously, one of the most important issues in pattern recognition is feature extraction. Since this is such a crucial step, different techniques may provide a better fit to our problem. An alternative to the PCA concept is the self-organizing feature map.

The ideas of SOFM are rooted in competitive learning networks. These nets are one layer nets with linear PEs but use a competitive learning rule. In such nets there is one and only one winning PE for every input pattern (i.e. the PE whose weights are closest to the input pattern). In competitive nets, only the weights of the winning node get updated. Kohonen proposed a slight modification of this principle with tremendous implications. Instead of updating only the winning PE, in SOFM nets the neighboring PE weights are also updated with a smaller step size. This means that in the learning process (topological) neighborhood relationships are created in which the spatial locations correspond to features of the input data. In fact one can show that the data points that are similar in input space are mapped to small neighborhoods in Kohonen's SOFM layer. Our brain has several known topographic maps (visual and auditory cortex).

The SOFM layer can be a one or two dimensional lattice, and the size of the net provides the resolution for the lattice. The SOFM algorithm is as follows:

- Initialize the weights with small different random values for symmetry breaking.
- For each input data find the winning PE using a minimum distance rule, i.e.

$$\vec{i}(x) = \arg_j \min \left\| \vec{x}(n) - w_j \right\| \qquad (4.5)$$

- For the winning PE, update its weights and those in its neighborhood L(n) by

$$w_j(n+1) = w_j(n) + \eta(n)\left[x(n) - w_j(n)\right] \qquad (4.6)$$

Note that both the neighborhood and the learning rate are dependent on the iteration, i.e. they are adaptive. Kohonen suggests the following Gaussian neighborhood

$$\Lambda_{j,j^*}(n) = e^{-\left[\frac{\left|r_j - r_{j^*}\right|^2}{2\sigma^2(n)}\right]} \qquad (4.7)$$

where j^* is the winning PE and $|rj-rj^*|$ is the spatial distance from the winning node to the j-th PE [Principe et. al., 1996]. The adaptive standard deviation controls the size of the neighborhood through iterations. The neighborhood should start as the full output space and decrease to zero (i.e. only the winning PE), according to

$$\sigma(n) = \frac{1}{(c_\sigma - d_\sigma n)} \qquad (4.8)$$

where c_σ and d_σ are constants. The step size $\eta(n)$ should also be made adaptive. In the beginning the step size should be large, but decrease progressively to zero, according to

$$\eta(n) = \frac{1}{(a_\eta - b_\eta n)} \qquad (4.9)$$

where a_η and b_η are also problem dependent constants.

The idea of these adaptive constants is to guarantee, in the early stages of learning, plasticity and recruitment of units to form local neighborhoods and, in the later stages of learning, stability and fine tuning of the map. These issues are very difficult to study theoretically, so heuristics have to be included in the definition of these values.

Once the SOFM stabilizes, its output can be fed to an MLP to classify the neighborhoods. Note that in so doing we have accomplished two things: first, the input space dimensionality has been reduced and second, the neighborhood relation will make the learning of the MLP easier and faster because input data is now structured.

4.5.9 Adaptive Resonance Theory Network

Adaptive resonance theory proposes to solve the stability-plasticity dilemma present in competitive learning. Grossberg and co-workers [Principe et al., 1996] add a new parameter (vigilance parameter) that controls the degree of similarity between stored patterns and the current input. When the input is sufficiently dissimilar to the

stored patterns, a new unit is created in the network for the input. There are two ART models, one for binary patterns and one for continuous valued patterns. This is a highly sophisticated network which achieves good performance, but the network parameters need to be well tuned.

4.5.10 Fukishima's Neocognitron

Fukushima [Principe et. al., 1996] proposed the Neocognitron, a hierarchical network for image processing that achieves rotation, scale, translation and distortion invariance up to a certain degree. The principle of a Fukushima network is a pyramid of two layer networks (one, feature extractor and the other, position readjusting) with specific connections that create feature detectors at increasing space scales. The feature detector layer is a competitive layer with neighborhoods where the input features are recognized.

In the model of artificial neuron suggested by Fukushima all synaptic weights and all input and output signals are nonnegative[Principe et. al., 1996]. In this model the inputs and corresponding synaptic weights are separated in to two groups: excitatory and an inhibitory one. The excitory effects e_j which is the weighted sum of all the excitatory inputs, is suppressed by the inhibitory effects h_j , which is the weighted sum of all the inhibitory inputs in a shunting manner. The output of the neuron can be described by the expression

$$y_j = \Psi\left[\frac{1+n\sum_{i=1}^{n} a_{ji}x_i}{1+\sum_{i=1}^{m} b_{ji}v_i} - 1\right], \qquad (4.10)$$

where

$$\Psi(u_j) = \begin{cases} u_j & if \quad u \geq 0, \\ 0 & otherwise \end{cases}$$

In this expression the a_{ji} mean the the excitatory synaptic weights and b_{ji} are the inhibitory synaptic weights.

4.5.11 Time Lagged Reccurent Network

TLRNs with the memory layer confined to the input can also be thought of as input preprocessors. But now the problem is representation of the information in time instead of the information among the input patterns, as in the PCA network. When we have a signal in time (such as a time series of financial data, or a signal coming from a sensor monitoring an industrial process) we do not know a priori where, in time, the relevant information is. Processing of the signal can be used here in a general sense, and can be substituted for prediction, identification of dynamics, or classification[Principe et. al., 1996].

A brute force approach is to use a long time window. But this method does not work in practice because it creates very large networks that are difficult or impossible to train (particularly if the data is noisy). TLRNs are therefore a very good alternative to this brute force approach. The other class of models that have adaptive memory are the recurrent neural networks. However, these nets are very difficult to train and require more advanced knowledge of neural network theory.

The most studied TLRN network is the gamma model. The gamma model is characterized by a memory structure that is a cascade of leaky integrators, i.e. an extension of the context unit of the Jordan and Elman nets[Principe et. al., 1996].

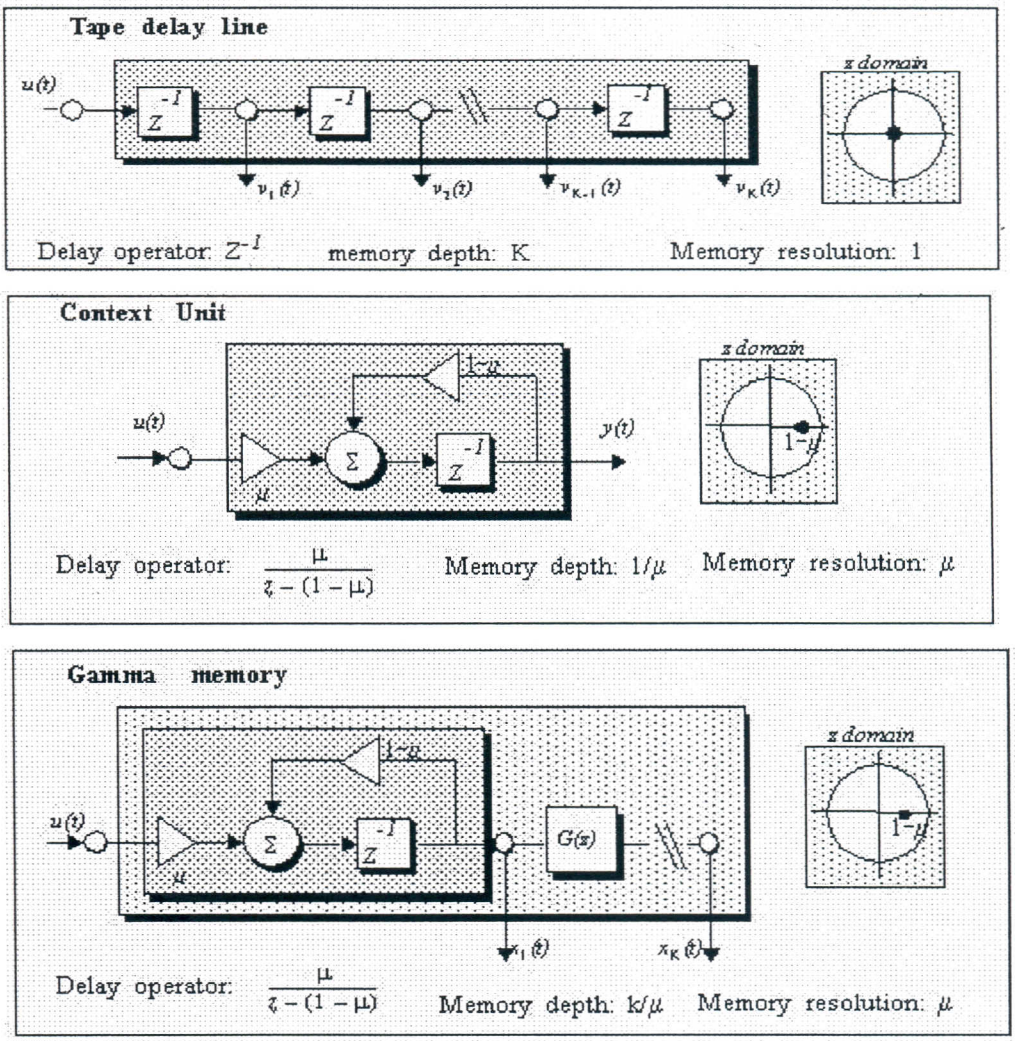


Figure 4.10 Connectionist memory structures, and the frequency domain location of the pole

The signal at the taps of the gamma memory can be represented by

$$x_0(n) = u(n) \qquad (4.11.a)$$

$$x_k(n) = (1-\mu)x_k(n-1) + \mu x_{k-1}(n-1) \quad k = 1,\ldots,K \qquad (4.11.b)$$

Note that the signal at tap k is a smoothed version of the input, that holds the voltage of a past event, creating a memory[Principe et. al., 1996].

Note that the point in time where the response has a peak is approximately given by k/μ, where μ is the feedback parameter. This means that the neural net can control the depth of the memory by changing the value of the feedback parameter, instead of changing the number of inputs. The parameter μ can be adapted using gradient descent

procedures just like the other parameters in the neural network. But since this parameter is recursive, a more powerful learning rule needs to be applied.

Memories can be appended to any layer in the network, producing very sophisticated neural topologies very useful for time series prediction and system identification and temporal pattern recognition

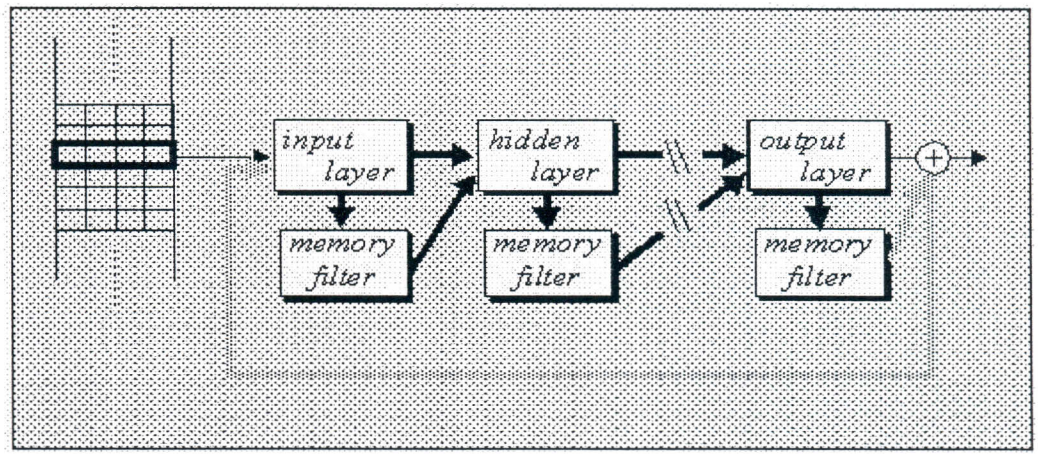


Figure 4.11 Use of gamma kernels in an MLP architecture

Instead of the gamma memory there are other memory structures that recently have been applied with some advantages. One of these is the Laguarre memory, based on the Laguarre functions. The Laguarre functions are an orthogonal set of functions that ares built from a lowpass filter followed by a cascade of all pass functions.

This family of functions constitutes an orthogonal span of the gamma space, so they have the same properties as the gamma memories, but they may display faster convergence for some problems. The equation for the Laguarre functions is

$$L(z,\mu) = \sqrt{1-(1-\mu)^2}\,\frac{(z^{-1}-(1-\mu))^{j-1}}{(1-(1-\mu)z^{-1})^{j}} \qquad (4.12)$$

Notice that this gives a recursion equation of the form

$$x_0(n) = (1-\mu)x_0(n-1) + \sqrt{1-(1-\mu)^2}\,u(n) \qquad (4.13.a)$$

$$x_k(n) = (1-\mu)x_k(n-1) + x_{k-1}(n-1) - (1-\mu)x_{k-1}(n) \qquad (4.13.b)$$

where u(n) is the input signal.

Chapter 5

LEARNING ALGORITHMS FOR ARTIFICAL NEURAL NETWORKS

5.1 Back-Propagation Learning Algorithms

In this section supervised learning from examples is described in detail, and most widely used learning procedure, the back- propagation algorithm, also called the generalized delta rule is introduced.

The back-propagation algorithm was developed first by Werbos in 1971, but his achievement remained almost unknown. The technique was rediscovered by Parker in 1982 and independently in 1986 by Rumelhart, Hinton and Williams [Cichocki and Unbehauen , 1993]. The multilayer perceptron networks were not used in the past because of lack of an effective learning algorithm. This has recently changed, mainly owing to Rumelhart and his coworkers who have popularized the back-propagation algorithm among the scientific community. Recently the back-propagation algorithm has also been adapted to feedback (recurrent) neural networks [Cichocki and Unbehauen , 1993]. In this section, discussion is limitted to back-propagation algorithms for the feedforward multilayer perceptron (MLP). The MLP is supposed to perform a specific nonlinear mapping or association task (e.g. classification, diagnosis, pattern recognition, etc.) which can be expressed in terms of a given input/output pattern (pairs). These input/output relations will be called a set of learning examples. Learning of the MLP consists in the adaptation of all synaptic weights in such a way that the difference between the actual output signals and the desired signals, averaged over all learning examples (input patterns), is as small as possible. The back-propagation learning algorithm can be considered as an unconstrained optimization problem of a suitably constructed error function (cost function).

5.1.1 Learning of the Single Layer Perceptron

In order to explain the back-propagation algorithm in its basic form, let us first consider the learning of a single neuron (located in the output layer of a multilayer perceptron (MLP)) Let us assume that the nonlinear activation function is chosen to be the hyperbolic tangent function,

$$y_j = \Psi(u_j) = \tanh(\gamma_j, u_j) = \frac{1-e^{-2\gamma_j u_j}}{1+e^{-2\gamma_j u_j}} \qquad (5.1)$$

where

$$u_j = \sum_{i=1}^{n} w_{ji} x_i + \Theta_j, \gamma_j > 0 \qquad (5.2)$$

The aim of learning is to minimize the instantaneous squared error of the output signal

$$E_j = \frac{1}{2}(d_j - y_j)^2 = \frac{1}{2} e_j^2 \qquad (5.3)$$

by modifying the synaptic weights w_{ji}.The problem of learning can be formulated as follows. Given the current set of synaptic weights w_{ji} , and the bias $\Theta_j = w_{ji}$ we need to determine how to increase or decrease them in order to decrease the local error function E_j. This can be done using a steepest-descent gradient rule expressed as

$$\frac{dw_{ji}}{dt} = -\mu \frac{\partial E}{\partial w_{ji}} \qquad (5.4)$$

where μ is a positive learning parameter determining the speed of convergence to the minimum. Taking into account that the instantaneous error can be expressed as

$$e_j = d_i - y_j = d_j - \Psi(u_j), \qquad (5.5)$$

where

$$u_j = \sum_{i=0}^{n} w_{ji} x_i \qquad (5.6)$$

with $w_{j0}, = \Theta_j$, and $x_0, = +1$, and applying the chain rule

$$\frac{dw_{ji}}{dt} = -\mu \frac{\partial E_j}{\partial e_j} \frac{\partial e_j}{\partial t} \qquad (5.7)$$

equation (5.8) is obtained .

$$\frac{dw_{ji}}{dt} = -\mu e_j \frac{\partial \cdot e_j}{\partial \cdot w_{ji}} = -\mu e_j \frac{\partial \cdot e_j}{\partial \cdot u_j} \frac{\partial \cdot u_j}{\partial \cdot w_{ji}} \qquad (5.8)$$

Furthermore,

$$\frac{d \cdot w_{ji}}{d \cdot t} = \mu \cdot e_j \frac{d \cdot \Psi(u_j)}{d \cdot u_j} x_i = \mu \cdot e \cdot \Psi'(u_j) \cdot x_i \qquad (5.9)$$

where δ_j, is called the learning signal or local error , is expressed as

$$\delta = e_j \Psi'(u_j) = -\frac{\partial E_j}{\partial u_j} \qquad (5.10)$$

If the activation function is hyperbolic tangent function given by equation (5.1) then the derivation $\Psi'(u_j)$ is given by

$$\Psi'(u_j) = \frac{d \cdot \Psi(u_j)}{du_j} = \gamma_j \left[1 - \tanh^2(\gamma_j u_j)\right] = \gamma_j (1 - y_j^2) \qquad (5.11)$$

In this case equation 5.9 can be written in the form

$$\frac{d \cdot w_{ji}}{d \cdot t} = \mu \gamma_j e_j (1 - y_j^2) x_i = \mu_j e_j (1 - y_j^2) x_i, \qquad (5.12)$$

with $\mu_j > 0$. Weight update stabilizes if y_j approaches -1 or +1 since the derivative $\partial y_i / \partial u_i$, equal to $(1-y_j^2)\gamma_j$, reaches its maximum for $y_j = 0$ and its minima for ± 1 [Cichocki and Unbehauen , 1993].

However, if the sigmoid activation function is unipolar that is described by

$$y_j = \Psi(u_j) = \frac{1}{1 + \exp(-\gamma_j u_j)} \qquad (5.13)$$

then

$$\frac{d\Psi(u_j)}{du_j} = \gamma_j y_j (1 - y_j) \qquad (5.14)$$

and equation (5.9) takes the form

$$\frac{dw_{ji}}{dt} = \mu \gamma_j e_j y_j (1 - y_j) x_i = \mu_j \cdot e_j y_j (1 - y_j) x_i \qquad (5.15)$$

It should be noted that in this case the derivative $\partial y_i / \partial u_i$ reaches its maximum for $y_j=1/2$ and, since $0 \leq y_j \leq 1$,approaches its minima as the output y_j approaches "0" or "1" [Cichocki and Unbehauen , 1993].

The synaptic weights are usually changed incrementally and the neuron gradually converges to a set of weights which solve the specific problem. On the basis of the algorithm given by equation (5.9) an incremental (discrete-time) change of the weight w_{ji} can be determined by the formula

$$\Delta w_{ji}(k) = \Delta w_{ji} = \Delta w_{ji}\left[(k+1)\tau\right] - \Delta w_{ji}(k\tau) = \eta_j \delta_j x_i \qquad (5.16)$$

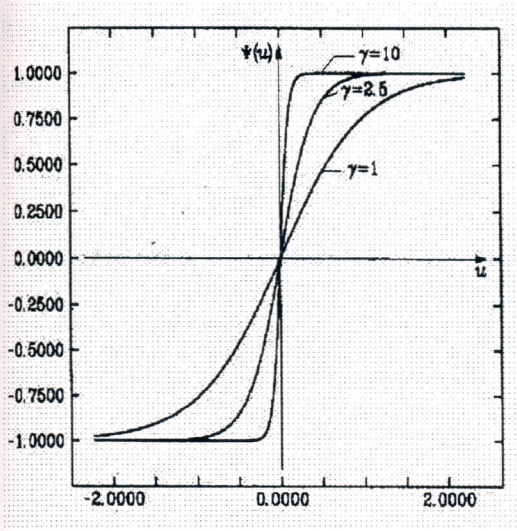
where

$$\delta_j := \delta_j(k\tau) = e_j(k\tau)\Psi'\left[u_j(k\tau)\right] = e_j(k\tau)\frac{d\Psi(u_j)}{du_j}|_{u_j=u_j(k\tau)}$$

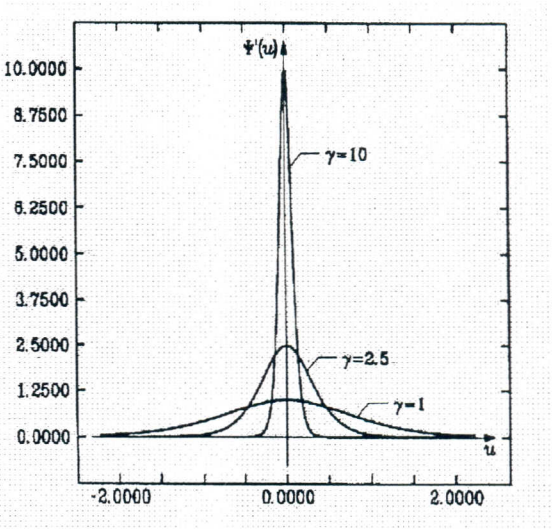
$$x_i := x(k\tau), \qquad \eta_j = \tau\mu > 0$$

Usually $\eta_j = \eta$ for j = 1, 2,..., n.

The implementation of the above algorithm requires an accurate realization of the sigmoid activation function and its derivative function. These functions are plotted in the Figure 5.1 . It can easily be noticed that shape of the derivative of sigmoid nonlinearity strongly depend on the value of the parameter γ. In VLSI implementations of neural networks practical, accurate realizations of such a function may be difficult [Cichocki and Unbehauen , 1993].



Activation functions $\Psi(u)=\tanh(\gamma u)$ for different vaues of γ.



Derivatives $\Psi'(u)$ for different values of γ.

Figure 5.1 Sigmoid functions and their derivatives

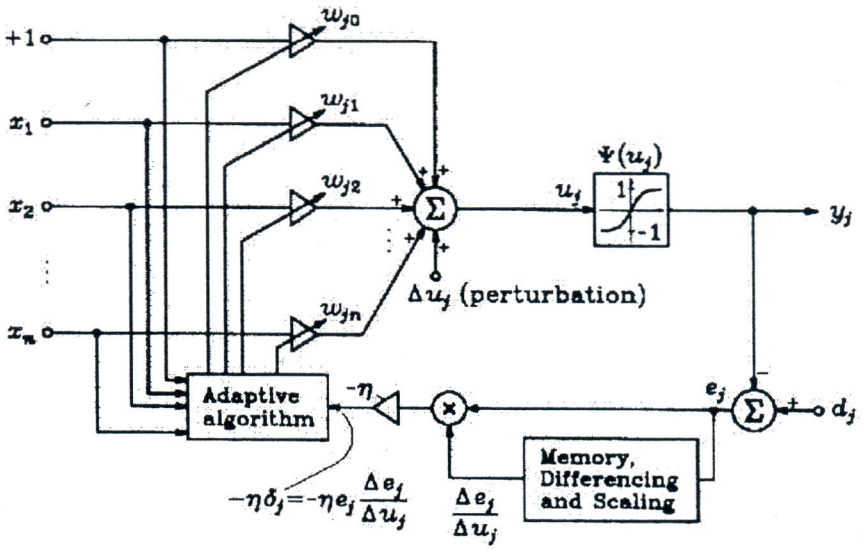


Figure 5.2 Implementation of modified back-propagation algorithm

An alternative implementation of the back-propagation algorithm which avoids the use of the derivative of the sigmoid activation function is illustrated in Figure 5.2. The network uses a small perturbation signal Δu_j which is added to the internal signal u_j in order to estimate the instantaneous gradient of the error

function. The effect of this small perturbation upon the output error $e_j = d_j - y_j$ is registered. The components of the instantaneous gradient can be expressed as [Cichocki and Unbehauen , 1993]

$$\frac{\partial E_j}{\partial w_{ji}} = \frac{1}{2}\frac{\partial e^2}{\partial w_{ji}} = \frac{1}{2}\frac{\partial e^2}{\partial u_j}\frac{\partial u_j}{\partial w_{ji}} = -e_j \frac{\partial e_j}{\partial u_j} x_i \qquad (5.17)$$

For small perturbations it can be writen as

$$\frac{\partial e_j^2}{\partial u_j} \cong \frac{(\Delta e_j^2)}{\Delta u_j} \qquad (5.17.a)$$

or

$$\frac{\partial e_j^2}{\partial u_j} \cong 2e_j \frac{\partial e_j}{u_j} \cong 2e_j \frac{\Delta e_j}{\Delta u_j} \qquad (5.17.b)$$

Hence, taking into account (5.3) and (5.16) the following two forms of an alternative algorithm for updating the synaptic weigths can be obtained:

$$\Delta w_{ji} = -\frac{1}{2}\eta \cdot e_j \frac{(\Delta e_j)^2}{\Delta u_j} x_i \qquad (5.18.a)$$

or

$$\Delta w_{ji} = -\eta \cdot e_j \left[\frac{\Delta e_j}{\Delta u_j}\right] x_i \qquad (5.18.b)$$

For small perturbations Δu_j these two forms of the algorithm are the same. In order to implement the algorithm the variation of the error e_j caused by a smalll perturbation signal Δu_j must be registered and the quantity $(\Delta e_j/\Delta u_j)$ or $((\Delta e_j^2)/(\Delta u_j))$ must be computed at each iteration step.

5.1.2 Standard Back-Propagation Algorithm for the Multilayer Perceptron

The above described approach of the adaptive learning (updating) of synaptic weights can be extended to the multi-layer perceptron (MLP). For simplicity of our further considerations let us assume that the MLP consists of three

layers of neurons: the first hidden layer with n_0 inputs and n_1 units, the second hidden layer with n_2 units, and the output layer with n_3 units.

The network behaviour should be determined on the basis of a set of input/output pairs. Each learning example is composed of n_0 input signals x_i ($i = 1, 2,..., n_0$) and n_3 corresponding desired output signals d_j ($j = 1, 2,..., n_3$). The input/output pairs are expressed as stable states of neurons which are usually represented by + 1 (ON) and - 1 (OFF). Learning of the MLP for a specific task is equivalent to finding the values of all synaptic weights such that the desired output is generated for the corresponding input. More explicitly, learning of the MLP means adjusting all weights such that the error measure between the desired output signals d_{jp} and the actual output signals y_{jp} averaged over all learning examples p will be minimal [Cichocki and Unbehauen , 1993]. The standard backpropagation algorithm uses the steepest-descent gradient approach to minimize the mean-squared error function. Such a local error function for the p-th learning example can be formulated as

$$E_p = \frac{1}{2}\sum_{j=1}^{n_3} (d_{jp} - y_{jp})^2 \qquad (5.19)$$

and the global error function as

$$E = \sum_p E_p = \frac{1}{2}\sum_p \sum_j (d_{jp} - y_{jp})^2, \qquad (5.20)$$

where d_{jp} and y_{jp} are the desired and actual output signal of the j-th output neuron for the p-th learning example, respectively.

There are two basic approaches to find the minimum of the global error function E. The first technique is the on-line or per example learning in which the training patterns are presented sequentially, usually in random order. For each learning example the synaptic weights $w_{ji}^{[s]}$ ($s = 1, 2, 3$) are changed by an amount $\Delta w_{ji}^{[s]}$ proportional to the respective negative gradient of the local error function E_p, which can be written mathematically as

$$\Delta w_{ji}^{[s]} = -\eta \frac{\partial E_p}{\partial w_{ji}^{[s]}}, \eta > 0 \qquad (5.21)$$

It has been proved that, if the learning parameter η is sufficiently small, this procedure minimizes the global error function $E = \Sigma E_p$ [Cichocki and Unbehauen , 1993]. Of

course, the above discrete gradient descent rule can be replaced by the continuous differential equations

$$\frac{dw_{ji}^{[s]}}{dt} = -\mu \frac{\partial E_p}{\partial w_{ji}^{[s]}}, \mu > 0 \qquad (5.22)$$

In the second approach, sometimes called "batch learning", the total error function E is minimized in such a way that the weight changes are accumulated over all learning examples before the weights are actually changed [Cichocki and Unbehauen , 1993].

At first we shall discuss the on-line learning approach in which the gradient search in the synaptic weight space is carried out on the basis of a local error function E_p. Let us first determine an updating formula for the synaptic weights $w_{ji}^{[s]}$ (s = 3) of the output layer. Using the chain rule for equation (5.19) $\Delta w_{ji}^{[3]}$ can be written as

$$\Delta w_{ji}^{[3]} = -\eta \frac{\partial E_p}{\partial w_{ji}^{[3]}} = -\eta \frac{\partial E_p}{\partial u_j^{[3]}} \frac{\partial u_j^{[3]}}{\partial w_{ji}^{[3]}} \qquad (5.23)$$

Taking into account that

$$u_j^{[3]} = \sum_{i=1}^{n_3} w_{ji}^{[3]} x_i^{[3]} = \sum_{i=1}^{n_3} w_{ji}^{[3]} o_i^{[3]} \qquad (5.24)$$

and defining the local error, called delta, by

$$\delta_j^{[3]} = -\frac{\partial E_p}{\partial u_j^{[3]}} = -\frac{\partial E_p}{\partial e_{jp}} \frac{\partial e_{jp}}{\partial u_j^{[3]}} = e_{jp} \frac{\partial \Psi_j^{[3]}}{\partial u_j}, \qquad (5.25)$$

a general formula for updating the weights in the output layer as

$$\Delta w_{ji}^{[3]} = \eta \delta_j^{[3]} x_i^{[3]} = \eta \delta_j^{[3]} o_i^{[3]} \qquad (5.26)$$

where

$$\delta_j^{[3]} = e_{jp} (\Psi_j^{[3]})' = (d_{jp} - y_{jp}) \frac{\partial \Psi_j^{[3]}}{\partial u_j^{[3]}}. \qquad (5.27)$$

Updating the synaptic weights in the hidden layers is little more complicated . For the second hidden layer the following equation can be written.

$$\Delta w_{ji}^{[2]} = -\eta \frac{\partial E_p}{\partial w_{ji}^{[2]}} = -\eta \frac{\partial E_p}{\partial u_j^{[2]}} \frac{\partial u_j^{[2]}}{\partial w_{ji}^{[2]}} = \eta \delta_j^{[2]} x_j^{[2]} = \eta \delta_j^{[2]} o_j^{[2]} . \qquad (5.28)$$

The local error for the second hidden layer is defined as

$$\delta_j^{[2]} = -\frac{\partial E_p}{\partial u_j^{[2]}}, (j = 1,2,\ldots,n_2) . \qquad (5.29)$$

However, this local error cannot be directly evaluated like local errors in the output layer . Instead, it can be represented in terms of quantities which are already known, and other quantities which are easily evaluated. Using the chain rule

$$\delta_j^{[2]} = -\frac{\partial E_p}{\partial u_j^{[2]}} = -\frac{\partial E_p}{\partial o_j^{[2]}} \frac{\partial o_j^{[2]}}{\partial u_j^{[2]}} \qquad (5.30)$$

is obtained.

Taking into account that

$$o_j^{[2]} = \psi_j^{[2]}(u_j^{[2]}) \qquad (5.31)$$

the local error can be written as

$$\delta_j^{[2]} = -\frac{\partial E_p}{\partial o_j^{[2]}} \frac{\partial \Psi^{[2]}}{\partial u_j} . \qquad (5.32)$$

The factor $-\partial E_p/\partial o_j^{[2]}$ can be evaluated as

$$-\frac{\partial E_p}{\partial o_j^{[2]}} = -\sum_{i=1}^{n_3} \frac{\partial E_p}{\partial u_j^{[3]}} \frac{\partial u_i^{[3]}}{\partial o_j^{[2]}} = \sum_{i=1}^{n_3} \left[-\frac{\partial E_p}{\partial u_j^{[3]}} \right] \frac{\partial}{\partial o_j^{[2]}} \left[\sum_{k=1}^{n_3} w_{jk}^{[3]} x_k^{[3]} \right]$$

$$= \sum_{i=1}^{n_3} \delta_i^{[3]} \frac{\partial}{\partial o_j^{[2]}} \left[\sum w_{ik}^{[3]} o_k^{[2]} \right] = \sum_{i=1}^{n_3} \delta_i^{[3]} w_{ij}^{[3]} . \qquad (5.33)$$

Thus the local error in the second layer can be evaluated by using the formula

$$\delta_j^{[2]} = \frac{\partial \Psi_j^{[2]}}{\partial u_j^{[2]}} \sum_{i=1}^{n_3} \delta_i^{[3]} w_{ji}^{[3]} . \qquad (5.34)$$

Similarly, an updating formula for the first hidden layer obtained as

$$\Delta w_{ji}^{[1]} = \eta \delta_j^{[1]} x_i^{[1]} = \eta \delta_j^{[1]} o_i^{[0]} = \eta \delta_j^{[1]} x_i . \qquad (5.35)$$

The local error is determined as

$$\delta_j^{[1]} = \frac{\partial \Psi_j^{[1]}}{\partial u_j^{[1]}} \sum_{i=1}^{n_3} \delta_i^{[2]} w_{ji}^{[2]} \qquad (5.36)$$

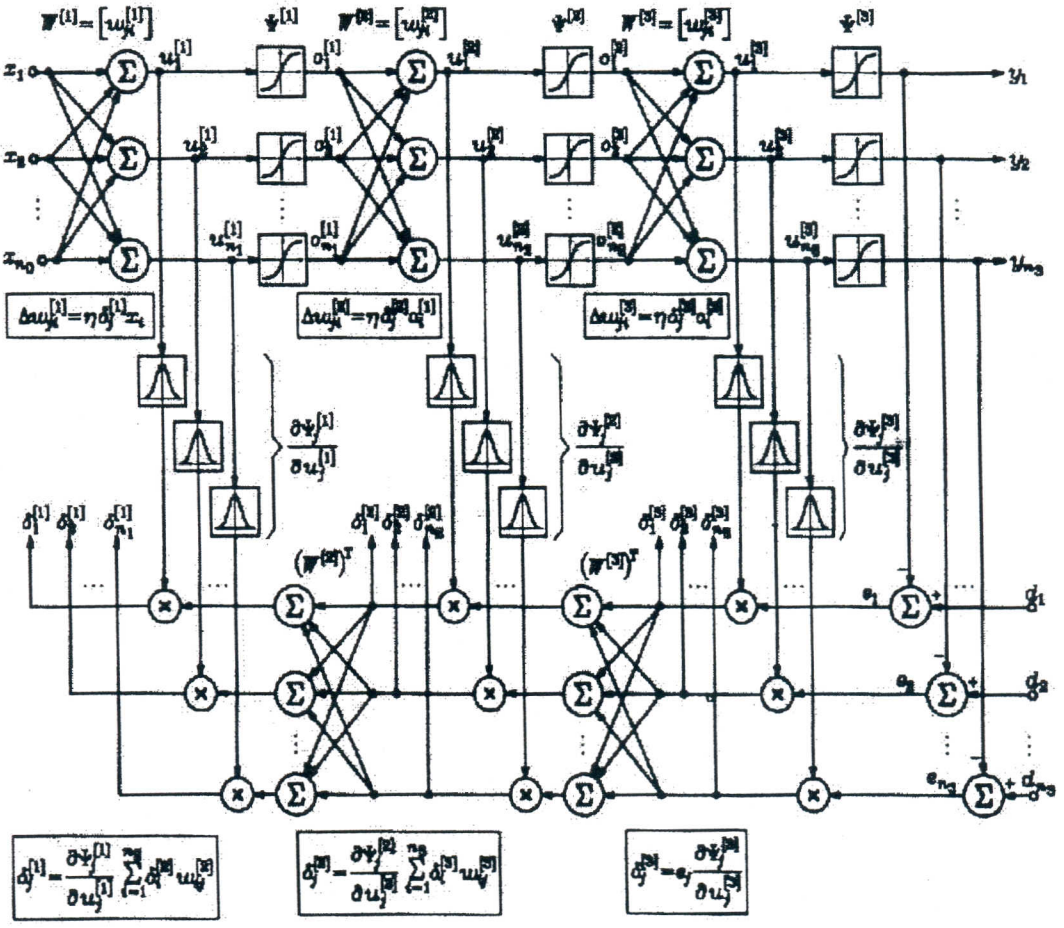


Figure 5.3 Network architecture for standard backpropagation algorithm of a three-layer perceptron

Generally, the local error of the hidden layer is determined on the basis of the local errors at an upper layer. Starting with the highest output layer we compute $\delta_j^{[3]}$ using formula (5.27), next we can propagate the errors $\delta_j^{[3]}$ backward to the lower layers. Figure 5.3 illustrates the functional scheme of the back-propagation

algorithm. The major deference of the learning rule for the output layer and the hidden layers is the evaluation of the local error $\delta_j^{[s]}$ (s = 1, 2, 3). In the output layer the error is a function of the desired and the actual output and the derivative of the sigmoid activation function. For the hidden layers the local errors are evaluated on the basis of the local errors in the upper layer.

The basic back-propagation algorithm can be performed by realizing the following steps:

Step 1: Initialize all synaptic weights wij to small random values.

(The initial values of the weights should be chosen rather small, since, if they are too large, the activation functions may saturate from the beginning, and the network will become stuck in a very flat plateau or a local minimum near the starting point. Usually, the initial values of weights are chosen as random values uniformly distributed between (- 0.5/fan-in) and (+ 0.5/fan-in), where fan-in of a unit is the number of units which are fed forward to this unit.)

Step 2: Present an input from the class of learning examples (input/output pattern) and calculate the actual outputs of all neurons using the present values of $w_{ji}^{[s]}$ and the pattern.

Step 3: Specify the desired output and evaluate the local errors $\delta_j^{[s]}$ for all layers

Step 4: Adjust the synaptic weights according to the iterative formula

$$\Delta w_{ji}^{[s]} = \eta \delta_j^{[s]} x_i^{[s]}, (s = 3,2,1) \qquad (5.37)$$

Step 5: Present another input pattern corresponding to the next learning example and go back to Step 2.

All the training examples are presented cyclically until the synaptic weights are stabilized, that is, until the error for the entire set is acceptably low and the network converges. Sometimes, in order to increase the convergence speed, it is useful to restrict the training set to those patterns for which the network fails to predict correctly. The learning process is extended to the entire training set as the performance of the network improves. After training a multilayer perceptron usually has the feature of a generalization, i.e. it has the ability for proper response to input patterns not

presented during the learning process. Such a generalization is an important feature of multilayer perceptrons.

This somewhat mysterious generalizing ability of the multilayer perceptron can be interpreted as follows. The multilayer perceptron performs a nonlinear mapping between an input and an output space. The learning of the perceptron can be regarded as synthesizing an approximation of a multidimensional function which is performing a simple fitting operation or a hypersurface reconstruction in a multidimensional space to a finite set of the training examples. From this point of view the generalization is nothing more than interpolating the test set on the fitting hypersurface[Cichocki and Unbehauen , 1993].

5.1.3 Back-Propagation Algorithm with Momentum Updating

The learning algorithm described in the previous section has some important drawbacks. First of all, the learning parameter η should be chosen small to provide minimization of the total error function E . However, for a small η the learning process becomes very slow. On the other hand, large values of η correspond to rapid learning, but lead to parasitic oscillations which prevent the algorithm from converging to the desired solution. Moreover, if the error function contains many local minima, the network might get trapped in some local minimum, or get stuck on a very flat plateau.

One simple way to improve the standard back-propagation learning algorithm is to smooth the weight changes by overrelaxation[Rumelhart and Hinton, 1986], that is, by adding the momentum term

$$\Delta w_{ji}^{[s]}(k) = \eta \delta_j^{[s]} o_j^{[s-1]} + \alpha \Delta w_{ji}^{[s]}(k-1). \qquad (5.38)$$

where

$\eta > 0$ $\quad 0 \le \alpha < 1$ (typically α =0.9)

The weights are now updated using the formula

$$w_{ji}^{[s]}(k+1) = w_{ji}^{[s]}(k) + \Delta w_{ji}^{[s]}(k) \qquad (5.39)$$

The second term in equation (5.38) is the momentum term which may improve the convergence rate and the steady state performance of the algorithm . Intuitively, if the previous weight change is large, then adding a fraction of this amount to the current weight update will accelerate the convergence process.

More precisely, if we are moving through a plateau region of the performance surface function then the gradient component $\partial E_p/\partial w_{ji}$ will be the same at each time step and equation (5.38) can be written as

$$\Delta w_{ji}(k) = -\eta \frac{\partial E_p}{\partial w_{ji}^{[s]}} + \alpha \Delta w_{ji}^{[s]}(k-1) \cong -\frac{\eta}{1-\alpha} \frac{\partial E_p}{\partial w_{ji}^{[s]}(k)} \qquad (5.40)$$

This means that the effective learning rate increases to the value $\eta_{eff}=\eta/(1-\alpha)$ without magnifying the parasitic oscillations. A momentum term is useful not only with the on-line learning but also with the batch learning algorithm.

5.1.4 Batch Learning Algorithm

In the on-line algorithm described above a pattern p (a learning example) is presented at the input and then all weights are updated before the next pattern is presented. In the batch learning algorithm the weight changes $\Delta_p w_{ji}$ are accumulated over some number (usually all) of the learning examples before the weights are actually changed. More precisely, using the standard back-propagation procedure described in Section 5.1.2 the weight updates $\Delta_p w_{ji}$ for all the synaptic weights w_{ji} in the MLP are calculated for the particular learning example .

This procedure is repeated for all the learning examples in the training set to yield the resulting update

$$\Delta w_{ji}^{[s]} = \sum_p \Delta_p w_{ji}^{[s]} = -\eta \frac{\partial E}{\partial w_{ji}^{[s]}} = \eta \sum_p \delta_{jp}^{[s]} o_{jp}^{[s-1]}, \qquad (5.41)$$

where the subindex p means that the corresponding variable is computed for the p-th learning example. Updating of all the weights is then made and again all learning

examples in the training set are presented in order to obtain a new update. Since the weight changes must be accumulated over the entire training set, the batch learning requires additional local storage for each connection.

During the training process, each time step is called an epoch and is defined to be a single sweep through all learning examples. At the end of each epoch the weights of the network are updated [Cichocki and Unbehauen , 1993].

In practice the back-propagation batch learning algorithm usually takes a more sophisticated form than that given by . The change in each weight is often calculated from the formula

$$\Delta w_{ji}^{[s]}(k) = \frac{\eta}{n_{s-1}} \sum_{p \in pattern\_set} \delta_{jp}^{[s]} o_{ip}^{[s-1]} + \alpha \Delta w_{ji}^{[s]}(k-1) - \gamma w_{ji}^{[s]}(k) \qquad (5.42)$$

(s= 1, 2, 3),

where η is the learning factor, α is the momentum coefficient, γ is the decay factor and n_{S-1} , is the number of processing units (neurons) in the (s - 1)-th layer.

The decay factor γ (with typical values of 10^{-3} to 10^{-4}) prevents the algorithm from generating very large weights. Very large weights may create such a high barrier in the total error function that a solution cannot be found within reasonable time. Moreover, the decay factor can improve the generalization capability of the neural network [Cichocki and Unbehauen , 1993].

It should be noted that the effective learning rate $\eta_{eff}^{[s]}=[\eta/(1-\alpha)]$ is inversely proportional to fan-in, where fan-in of a neuron is the number of neurons which are inputs to it. In the fully connected multi-layer perceptron the fan-in at each layer is equal to the number of neurons in the lower layer.

As it has been already pointed out, one serious problem of the back-propagation algorithm is that it can be trapped in a local minimum of the error function. A simple and effective technique which often enables us to avoid local minima is to choose examples in random order from the training set. Such a random order generates a "noise" which enables us to get out of local minima. On the other hand, if the training set is cycled regularly in the same sequence of pattern pairs the probability of getting stuck in a local minimum is greater since series weight changes cancel.

Another promising technique which greatly improves the performance of the neural network is adding noise to the input training set. It appears that a neural network trained with noise-distorted inputs not only often allows an escape from a local minimum, but also has a better ability to recognize noisy patterns, and performs better in recognizing or correctly classifying patterns that have never been presented to the network during the training procedure [Sietsma, 1991]. The last feature is called the generalization ability and it is one of the major strengths of artificial neural networks.

5.1.5 Comparison of the On-Line and the Batch Procedures

(i) The on-line approach has to be used if all training examples are not available before the learning starts and an adaptation to the actual (on-line) stream of training patterns is desired.

(ii) Furthermore, the on-line learning algorithm is usually convenient and more effective than the batch algorithm when the number of the training examples is very large, since the batch procedure requires auxiliary memory to accumulate the local updates.

(iii) The on-line procedure introduces some randomness (noise) that often may help in escaping from local minima. On the contrary the standard batch procedure introduces some inherent averaging filtering due to collecting the total gradient information before deciding the next step. Although the batch learning algorithm provides a better estimate of the gradient components and avoids a mutual interference of the weight changes (caused by different patterns) it does not improve the learning speed sufficiently to compensate for the additional computational cost.

(iv) Usually, the on-line algorithm is faster and more effective than the standard batch procedure, especially for large-scale classification problems, This can be explained by the fact that usually many training examples possess redundant information in the sense that many contributions to the gradient are very similar, and waiting to compute all these contributions before updating the weights is simply wasteful.

(v) However, for many applications, especially, if high precision mapping is required, the batch procedure may be the method of choice.

(vi) Moreover, the batch approach lends itself to straightforward applications of more sophisticated optimization procedures.

Summarizing, the relative effectiveness of the on-line and batch procedures is highly dependent on the problem, but the on-line algorithm seems superior in most cases, especially for large redundant training sets[Cichocki and Unbehauen , 1993].

5.1.6 Back-propagation Algorithm with a Neural Network having Variable Hidden Layer Dimension

The number of neurons to be used in the hidden layers is not known in advance, and usually is estimated by a trial and error approach.

One possible approach is to construct a neural network with an excessive number of hidden units, i.e., a network which is known or suspected to be larger than required is used and then some redundant units are removed during the learning process[Sietsma, 1991]. All neurons that do not contribute to the solution or give information not required at the next layer [Sietsma, 1991] can be considered as redundant hidden units. In order to find which hidden units can be removed, the output of all the hidden units is monitored analyzed accross all the training examples after the network achieves convergence. If the output of a certain hidden units give approximately the same or opposite output for all training examples , only one of these units is needed since both neurons convey the same information[Cichocki and Unbehauen , 1993] . After removing some hidden units which do not contribute to the solution the weights of the reduced network must be appropriately updated or the network must be trained once again to ensure the desired performance. The process of removing the redundant hidden units to produce the smallest neural network capable of performing a desired task is called pruning [Sietsma, 1991].

The technique described above enables neural network designer to reduce the number of hidden units. Sometimes, at the first stage of learning process, it is sometimes convenient to gradually increase the number of hidden units. After achieving

the desired convergence, it is recommended to remove some of them in order to find the minimal size of the network which performs the desired task. [Cichocki and Unbehauen , 1993]

The purpose gradually increasing the number of hidden units is important in two ways:

- The number of hidden units is not known in advance. Gradually increasing the number of units during the training process it is possible to find the optimal number of units.
- The learning process may became trapped in a local minimum or very flat plateau. If one or more extra hidden units are added to the network it may escape from a local minimum since the error function in the weight space changes its shape.

Recently, a heuristic back-propagation algorithm has been propose which varies the number of hidden units dynamically[Cichocki and Unbehauen , 1993]. According to this algorithm a new hidden unit is added when network becomes trapped in a local minimum or a very flat plateau. For this puppies a total error function E is monitored and examined during the learning process. If the error function does not decrease very slowly less than a predefined rate within the next 50 weight update, the network is probably trapped in a local minimum and number of hidden units is insufficient for the convergence. Thus, new hidden units need to be added. The initial values of the weights of this unit can be set randomly in some ways as in the standard back-propagation algorithm.

The network is trained again and if the error function fails to decrease or decrease by more than predefined rate within the next 50 weight update one or more hidden units are added. This procedure is repeated until the network eventually converges.

Since, in some cases, the number of hidden units becomes excessively large in the second stage of the training we can try to reduce the number of hidden units according to technique described above. It should be noticed that too many hidden units usually degregades the network performance to generalise[Cichocki and Unbehauen , 1993].

5.2 Speeding up the Back-propagation Learning Algorithms

Training of the feedforward multilayer neural networks using the standard backpropagation results in slow convergence. There are cases in which the learning speed is limiting factor in practical applications of neural networks. [Cichocki and Unbehauen , 1993]

There are several approaches to increase the convergence speed:

- Estimation of optimal initial conditions. In other words, finding initial weights that are better starting values than pure random values.
- Reducing the size of problem by pre-processing of the data, for example by employing future extraction algorithms or the projections like Principle Component Analysis.
- Applying more sophisticated optimisation algorithms.

Numerous heuristic optimisation algorithms have been proposed to improve the convergence speed of the standard back-propagation algorithm. Unfortunately, some of these algorithms are computationally very expensive.

5.2.1 Back-Propagation Learning Algorithm with an Adaptive Slope of the Activation Functions

In the standard back-propagation learning algorithm, it was assumed that the slope γ of the activation function is fixed (typically 1). Kruschke and Movellan[Amari , 1991] have shown that a modified learning algorithm with adaptive slope significantly increase the learning speed and improve the generalization.

Considering a three layer neural network architecture it can be assumed that all activation functions have variable slopes $\gamma^{[s]}$ which must be computed during the learning process to be able to evaluate output of neurons as follows;

$$o_i^{[s]} = \Psi_i^{[s]}(\gamma_i^{[s]} u_i^{[s]}) = \tanh(\gamma_i^{[s]} u_i^{[s]}) \qquad (5.43)$$

Applying the standard gradient descent approach it can be found out that

$$\Delta w_{ij}^{[s]} = -\eta_w \frac{\partial E_p}{\partial w_{ij}^{[s]}} = \eta_w \delta_i^{[s]} o_j^{[s-1]}, \qquad n_w > 0 \tag{5.44}$$

and

$$\Delta \gamma_i^{[s]} = -\eta_\gamma \frac{\partial E_p}{\partial \gamma_i^{[s]}} = \eta_\gamma \delta_i^{[s]} u_i^{[s]} / \gamma_i^{[s]}, \quad \eta_\gamma > 0 \tag{5.45}$$

where the local errors are calculated as

$$\delta_i^{[s]} = -\frac{\partial E_p}{\partial u_i^{[s]}} = \frac{\partial \Psi_i^{[s]}}{\partial u_i^{[s]}} \sum_{k=1}^{n_{s+1}} \delta_k^{[s+1]} w_{ki}^{[s+1]} \tag{5.46}$$

for s=1,2

and

$$\delta_i^{[3]} = -\frac{\partial E_p}{\partial u_i^{[3]}} = \frac{\partial \Psi_i^{[3]}}{\partial u_i^{[3]}} e_{ip} \tag{5.47}$$

5.2.2 Search-Then Converge Strategy

Darken and Moody[Cichocki and Unbehauen , 1993] proposed a simple approach called " search then converge" strategy. According to this strategy the learning rate is gradually decreasing during the learning process. In the first phase of learning (search phase) the learning rate is almost constant, that is, it decreases very slowly and it is sufficiently large. In the second phase of learning (convergence phase) the learning rate exponentially decreases to zero. Two possible schedules for the learning rate have suggested[Cichocki and Unbehauen , 1993].

$$\eta^{(k)} = \eta_0 \frac{1}{1 + \frac{k}{k_0}} \tag{5.48}$$

and

$$\eta^{(k)} = \eta_0 \frac{1 + \frac{c}{\eta_0} \frac{k}{k_0}}{1 + \frac{c}{\eta_0} \frac{k}{k_0} + k_0 \left(\frac{k}{k_0} \right)^2}, \tag{5.49}$$

where $\eta_0 > 0$, $c > 0$, $k_0 >> 1$ (typically $100 \leq k_0 \leq 500$) are suitably chosen parameters.

Note that for $k << k_0$ the learning rate $\eta^{(k)} \approx \eta_0$, and for $k >> k_0$ the functions decrease proportional to 1/k. It has been demonstarted that for suitabily chosen parameters a considerable improvement in the convergence speed can be achived[Cichocki and Unbehauen , 1993].

5.2.3 Averaging Procedure

The averaging procedure for adaptive filtering developed by Polyak is somewhat similar to the search -then-converge strategy of Darken and Moody. According to this approach the weights can be computed as [Cichocki and Unbehauen , 1993].

$$\tilde{w}_{ij}(k+1) = \tilde{w}_{ij}(k) - \tilde{\eta}^{(k)} \frac{\partial E}{\partial \tilde{w}_{ij}} \qquad (5.50)$$

$$w_{ij}(k+1) = w_{ij}(k) + \eta^{(k)} \left[\tilde{w}_{ij}(k+1) - w_{ij}(k) \right],$$

where

$$\frac{\partial E}{\partial \tilde{w}_{ij}} = \sum_p \frac{\partial E_p}{\partial \tilde{w}_{ij}}, \quad \tilde{\eta}^{(k)} = \eta_0 / k^{\gamma}, \quad \left[\frac{1}{2} < \gamma < 1 \right] \text{ and}$$

$\eta^{(k)} = 1/(1+k)$.

The above learning algorithm combines two processes. The first computing process is a standard stochastic recursive algorithm with its learning rate $\tilde{\eta}^{(k)} \eta_0 / k^{\gamma}$. The second process is an averaging process with learning rate $\eta^{(k)} = 1/(1+k)$. In contrast to the standard approach two sequences of weigths $\{\tilde{w}_{ij}\}$ and $\{w_{ij}\}$ are computed where $\{w_{ij}\}$ is arithmetic average of $\{\tilde{w}_{ij}\}$. The essential future of the algorithm is that it employs two learning rates: $\tilde{\eta}^{(k)} = \eta_0 k^{-\gamma}$ and $\eta^{(k)} = (1+k)^{-1}$. Note that $\gamma=1$ is not included in the algorithm above and that the learning rate $\tilde{\eta}^{(k)}$ decreases more slowly than $\eta^{(k)}$. In practice the averaging process will start not at k=0

but from $k \geq k_0$ for which $\{\tilde{w}_{ij}\}$ enter the neighborhood of the optimal desired values, that is, the learning rates can be changed as

$$\tilde{\eta}^{(k)} = \frac{\eta_0}{k_0} \frac{1}{\left[1 + \frac{k}{k_0}\right]^{\gamma}} \qquad (5.51)$$

and

$$\eta^{(k)} = 1/(k - k_0) \qquad \text{for } k > k_0. \qquad (5.52)$$

It should be noted that that one may use only the set of difference equations but at the expense of slower convergence speed. The use of the second set of equations (averaging process) may improve the convergence speed[Cichocki and Unbehauen , 1993].

5.2.5 Quickprop

Quickprop is a heuristic learning algorithm for a multilayer perceptron, developed by Fahlman, which is based on Newton's method[Cichocki and Unbehauen , 1993].

When processing unit with a standard sigmoid function is near its maximum or minimum, the slope of the activation function is near zero, so the corresponding local error δ_i is near zero and consequently the corresponding weight change Δw_{ij} is near zero. In order to overcome this problem Fahlman uses in his quickprop algorithm as modified activation functions

$$\Psi(u_i) = (1 + e^{-\gamma u_i})^{-1} + 0.1u_i \qquad (5.53)$$

or alternatively

$$\Psi(u_i) = \tanh(\gamma_i u_i) + 0.1u_i \qquad (5.54)$$

The quickprop learning algorithm can be formulated as

$$\Delta w_{ij}(k) = -\eta^{(k)} S_{ij}(k) + \alpha_{ij}^{(k)} \Delta w_{ij}(k-1) \qquad (5.55)$$

$$S_{ij}(k) := \frac{\partial E(w^{(k)})}{\partial w_{ij}} + \gamma w_{ij}(k-1),$$

with learning rate

$$\eta^{(k)} = \begin{cases} \eta_0, & \text{if} \quad \Delta w_{ij}(k-1) = 0.0 \quad \text{or} \quad S_{ij}(k)\Delta w_{ij}(k-1) > 0.0, \\ 0, & \text{otherwise} \end{cases} \qquad (5.56)$$

and the momentum rate

$$\alpha_{ij}^{(k)} = \begin{cases} \alpha_{\max}, & \text{if } \tilde{\alpha}_{ij}^{(k)} > 0 \quad \text{or } S_{ij}(k)\Delta w_{ij}(k-1)\tilde{\alpha}_{ij}^{(k)} < 0.0 \\ \tilde{\alpha}_{ij}^{(k)} = \dfrac{S_{ij}(k)}{S_{ij}(k-1) - S_{ij}(k)}, & \text{otherwise} \end{cases} \qquad (5.57)$$

Note that in this algorithm the learning rate $\eta^{(k)} \neq 0$ is only necessary to start the training or restart it, if $\Delta w_{ij}=0$. A small decay factor γ is used to prevent weights from growing to very large values. Typical values of parameters are: $0.01 \leq \eta_0 \leq 1.75$, $\gamma=10^{-4}$. Usually, the quickprop algorithm is simplified as [Cichocki and Unbehauen , 1993]

$$\Delta w_{ij}(k) = \begin{cases} \alpha_{ij}^{(k)} \Delta w_{ij}(k-1), & \text{if } \Delta w_{ij}(k-1) \neq 0, \\ \eta_0 \dfrac{\partial E}{\partial w_{ij}}, & \text{if } \Delta w_{ij}(k-1) = 0, \end{cases} \qquad (5.58)$$

where

$$\alpha_{ij}^{(k)} = \min\left\{ \frac{\dfrac{\partial E(W^{(k)})}{\partial w_{ij}}}{\dfrac{\partial E(W^{(k-1)})}{\partial w_{ij}} - \dfrac{\partial E(W^{(k)})}{\partial w_{ij}}}, \alpha_{\max} \right\}$$

Chapter 6

MOTION MEASUREMENT, TRAJECTORY PREDICTION AND PARAMETER ESTIMATION

6.1 Introduction

"Parameter" and "estimation" are the most frequently encountered pair of words in the science literature. This is because science aims to convert mysterious events into comprehensible phenomena.

In this chapter, a literature survey of parameter estimation in relation with motion parameter estimation subject is presented. In addition, motion measurement systems currently being used are introduced.

6.2 A Brief Bibliography of Motion Measurement Techniques and Tools

In order to design a real-time system performing motion estimation, a reliable motion measurement need to be done. Motion measurement is carried out with currently available special devices. In this section some of the methods exploited to determine motion parameters and their implemented forms are explained briefly.

Macreflex Motion Measurement System is a professionally produced, user-friendly, and flexible 2D/3D movement analysis system[Bateman, 1995]. Specially designed video camera(s) emit and receive infrared light. Reflective markers are used to identify the moving target. If at least two cameras are used, the relative 3D coordinates of the target are displayed qualitatively on screen. The exact coordinates can be exported to a spreadsheet for further analysis. The package includes the WingZ (Informix) spreadsheet with an additional pull-down menu of MacReflex scripts that will graph the data (position, position-time, velocity, acceleration, angle, angular velocity, angular acceleration). It is possible to write further scripts, for example, to operate on batches of

files. The raw data is available for further statistical manipulation, within WingZ, or following export to an analysis package.

Peak Performance Technologies, Inc. has been supplying 2D and 3D motion measurement systems to the medical, biological, ergonomics, sports and animation markets since 1984. Peak software captures and calculates the 2D and 3D coordinates of specified points on a moving object using video or real-time hardware. These coordinates are then used to calculate kinematics data such as linear and angular velocity and centers of mass. Coordinate data can also be integrated with analog data from force platforms and EMG devices to calculate kinetics such as the joint moments and joint forces. Report generation includes synchronized video images, stick figures and graphics on Cartesian axes, along with exporting the data to animation software or statistical packages.

Motion and Position Measurement (MPM) system data is often used in conjunction with design verification, fatigue calculations and life time estimates, but also for the purpose of on-line positioning control.

6.3 A Brief Bibliography of Motion Prediction

Motion parameter estimation is a special field of interest in which non-linear estimation theory is used extensively. This section is devoted to different approaches that has been used in motion parameter estimation. The original study inspired many concepts from trajectory prediction literature and literature on prediction of non-linear system behavior.

Goerke and Eckmiller [Goerke and Eckmiller, 1993] used a neural network model approximating a function which maps the one-dimensional input space (time) onto an m-dimensional output space (workspace of the trajectory). Approximating a trajectory $\hat{S}$ with the function S generated by the network is done by approximating the second derivative $\hat{S}''$ of the desired trajectory $\hat{S}$ and integrating it twice to form a smooth trajectory S. The acceleration $\hat{S}''$ is considered as a sequence of forces driving a virtual fingertip with unity mass through m-dimensional workspace.

Within the network the locally constant acceleration values are stored as synaptic weight vectors $W^{(x)}$, $W^{(y)}$,... which are activated by the locally active P-layer

neurons and subsequently integrated by the Neural Integrators to the final trajectory in the study made by Goerke and Eckmiller [Goerke and Eckmiller, 1993].

Goerke and Eckmiller [Goerke and Eckmiller, 1993] divided all trajectory in to N intervals with duration D_i which are called parcels; within one parcel i the corresponding neuron i in the Parsel-layer is fully active as defined by a pair of values: "switch on-" and "switch off-" thresholds.

For a given parceling situation the weights **W** are changed according to the cumulative δ-rule after each epoch (duration of trajectory).

$$_{new}w_n^{(x)} =_{old} w_{old}^{(x)} + \eta\delta_n^{(x)}; \quad \delta_n^{(x)} = \frac{1}{D_n}\sum_{t=0}^{t=end}(\gamma\Delta x_t'' + \beta\Delta x_t' + \alpha\Delta x_t)out_n(t) \quad (6.1)$$

where $\delta_n^{(x)}$ denotes the linear combination of the cumulative difference between desired and estimated acceleration, $\Delta x_t''$, velocity $\Delta x_t'$ and spatial position Δx_t normalized by the parcel-duration D , for the x-component with respect to parcel n..

Debrunner and Ahuja [Debrunner and Ahuja, 1990] proposed a motion estimation model based on tracking set of points on the moving object. The system predicts trajectories of many point objects and combine them to predict motion of the rigid object as a whole. Since distances between point do not change because the object is assumed to be a rigid one , using many points at a time to check distances between points provides a mechanism to reduce the prediction errors.

In [Debrunner and Ahuja, 1990] the object centered coordinate system is a cylindrical coordinate system whose axis is aligned with the rotation axis $\vec{\Omega}$ and with its origin at $\vec{c}_0(f)$ in frame f. Each point on the object has three coordinates: the radial distance r_p from the rotation axis, the phase or initial angle θ_p between the radial vector to the point and the axis, and the distance d_p from $\vec{c}_o$ along the rotation axis. They formulated instantaneous position of a single point by the equation (6.2) .

$$\vec{P}(f) = \begin{bmatrix}(1-F_x^2) & -F_xF_y \\ -F_xF_y & (1-F_y^2)\end{bmatrix}\begin{bmatrix}\cos wf & -\sin wf \\ \sin wf & -\cos wf\end{bmatrix}\begin{bmatrix}\cos\theta_p \\ \sin\theta_p\end{bmatrix} + d_p\sqrt{2-F_x^2-F_y^2}\,\vec{F} + \vec{c}_0(f), \quad (6.2)$$

where f is the frame sequence number.

In the system n points viewed in m successive frame are used for estimation. Position difference are used to construct a matrix having dimension m x n. This matrix is decomposed into matrices O and C by means of SVD (singular value decomposition) method. The decomposed matrices are used to predict next position of moving object.

Chen and his friends [Chen et al, 1994] performed image motion estimation by using the spatio-temporal approach which has largely relied on the constant velocity assumption. Time-variation is modeled either as a polynomial or as a periodic function of time. Under these models, they showed that time-varying image motion estimation is equivalent to parameter estimation of one-dimensional polynomial phase or phase modulated signals. Spatio-temporal algorithms was developed in [Chen et al, 1994] to handle time-varying motion. Time-variation is modeled either as a polynomial or as a periodic function of time.

Pierre Payeur and his friends [Payeur et al, 1995] conducted a study on the valid observation that objects do not move totally at random. They exploited neural networks' ability to learn regularities in a sequence to predict motion parameters of moving object so that a manipulator could use this information to catch and grasp the moving object. If prediction of the motion variables could be achieved, manipulator and object could meet under conditions such that no collision and sliding would occur.

In their study, the positions and orientation of the object (x, y, z, θ, ϕ, φ) were provided at .regular intervals by the vision system which processes the scanned images of the scene. They assumed that the object velocity and acceleration are within acceptable range compared to manipulator joint limitations.

They formed a generic motion model which is described by cubic equation given below.

$$X(t) = \frac{1}{6}\alpha_0 t^3 + \frac{1}{2}\beta_0 t^2 + \gamma_0 t + X_0 \qquad (6.3)$$

where X is represents position, orientation, velocity or acceleration depending on the considered variable, X_o is the initial value , t is the time.

The cubic model was chosen because it can represent rather complex system nonlinear trajectories with low risk of overmodeling. Besides, cubic model allows variable acceleration.

A cubic polynomial with assigned coefficients is used to generate training data. Generated data was used as baseline in predictions. Predictions performed by cubic model updating technique and neural network approach. In their study Payeur and his friends pointed out superiority of neural network prediction models over cubic model updating technique.

6.4 A Brief Bibliography of Problem of Non-linear Systems' Behaviour Prediction

As stated in Chapter 3 as well , the estimation of motion parameters is nothing but a problem of time series prediction. The complexity of the prediction task depends on the system that produces time series throughout its evolution. If the system is linear it is not difficult to carry out the estimation task. However, if system is a non-linear or a probabilistic one it is much harder to attain a precision beyond a certain limit.

In this section some studies conducted to find solution to complex time series prediction are summarized. People that made these studies were not intending to develop a motion prediction model, but the methods they used in prediction of some typical time series may be suited for application in motion parameter estimation field. For this reason it is worthwhile to mention these studies here.

Principe and his friends [Principe et. al., 1991] used time delay neural networks to predict Mackey-Glass(D=30) time series. The training set was a different set of 3,500 points of the time series. They obtained prediction results over a test set of 500 points of the Mackey-Glass system characterization

In [Principe et. al., 1991] long term predictions are utilized in rating of the predictors. TDNN used was able to follow with a very small error the Mackey-Glass time series up to 45 steps.

Bengio and his friends [Bengio et. al., 1995] performed a study to optimize the medium term prediction of sunspots using multilayer perceptron (MLP) neural networks with different configuration and different input data set formats.

Output schemes used in [Bengio et. al., 1995] are as follows

a MLP with one output

a MLP with h+1 output units $x_t, x_{t+1}, \ldots, x_{t+h}$

a MLP with one output unit x_t. In that case is predicted using as input previously predicted values $\hat{x}_t, \cdots, \hat{x}_{t+h-1}$, which are estimates of $x_t, x_{t+1}, \ldots, x_{t+h-1}$.

They used different embedding dimension and time delay combinations. In addition, they increased the number of training sample by using daily measurements instead of monthly and weekly ones. Bengio and his friends [Bengio et. al., 1995] observed that increasing size of training set decreased the overfitting.

Chapter 7

A COMPARATIVE STUDY ON MOTION PARAMETER ESTIMATION BY MEANS OF DIFFERENT NEURAL NETWORK ARCHITECTURES

7.1 Introduction

Task of measuring motion parameters requires use of advanced technological devices. These devices determine object location in the 3D space in regard to time as described in the previous chapter. Assuming that such devices are providing necessary information about motion of an object, a mechanism able to make accurate on-line prediction of motion parameters can be used to solve important automation problems. Because complexity of motion types display significant variations, it is more appropriate to approach the problem with motion models. In this chapter, several motion models will be considered and capabilities of different neural network topologies to predict parameters of these motion types are compared on the basis of error function.

In motion estimation context, prediction can be defined as approximating a function underlying dynamics of the system under discussion. The first step in design of a realistic estimation system is identification of variables that influence the parameter to be estimated. This can be achieved by means of statistical analysis or by some mechanisms which decide to remove or keep a given variable in the prediction process dynamically. A mechanism which does not assume any prior conditions about the influence of a certain variable on the parameter to be predicted can be implemented by including all the suspected variable into the system. While the neural network predictor is being trained, those variables which have no effect on the prediction can be identified by observing the changes in the weights coming out of input nodes to which instantaneous values of these variables are fed. If the changes in this group of weights that connect an input node to other neurons are not significant as compared to changes in weights coming out of other input nodes then it

can be concluded that the variable that fed to this input node can be taken out of prediction process. In the artificial neural network literature this operation is named as "weight pruning". That is, by removing the weights coming out of a input node, that input node is made ineffective.

Nevertheless, the mechanism mentioned above is difficult to implement due to the computational complexity of operations and possible synchronization problems of parameter values caused by data acquisition systems. Although this issue will not be discussed any further, it is an important point for the design of realistic prediction system.

7.2 Point Object Following a Periodic Trajectory

Periodic motion is one of the simplest and the most easily understood motion types. In this section this kind of motion is considered and two neural network topologies are employed in the task of learning the dynamics of this motion without supplying them with any prior knowledge about motion model.

Let formulate object spatial position with respect to time by a set of parametric equations.

$$X(t) = 100 \quad Cos(\frac{\pi}{25}t) \tag{7.1}$$

$$Y(t) = 400 \quad Sin(\frac{\pi}{50}t) \tag{7.2}$$

$$Z(t) = 400Cos(\frac{\pi}{100}t)Sin(\frac{\pi}{25}t) \tag{7.3}$$

This set of equation can be a motion model of a machine part on a shaft such that while the shaft is subject to elliptic motion, a machine part itself is moving periodically up and down. Figure 7.1 presents 3-D plot of object trajectory described above. For the safe manipulation of the object attached to this part manipulator has to predict approximate position, velocity and even acceleration of object with a certain accuracy.

7.2.1 Generation of 3-D Motion Time Series Data

Motion data has been generated by assigning successive integer values to time variable t starting from "0". The size of generated data is limited to 450 successive coordinates in the space. Here $\Delta t=1$ is a hypothetical time unit (let say $\pi/25$ second) indicating that an imaging system can identify 450 successive position of object in 450 equally spaced intervals. If the motion measurement system can perform faster position determination, then higher resolution data can be input to the trajectory learning system which is supposed to improve the accuracy of the prediction.

A sample of 3-D motion data is shown in tabular form in Table 7.1.

Table 7.1 Successive spatial positions of point object that is subject to periodic motion

Time	X(t)	Y(t)	Z(t)
0	100	0	0
1	99.21227	25.10349	50.08331
2	96.86148	50.10801	99.2306
...	...	...	...
...	...	...	...
...	...	...	...
447	91.89188	80.53882	-15.9661
448	96.10925	55.79068	-7.72434
449	98.81246	30.82259	-2.36978

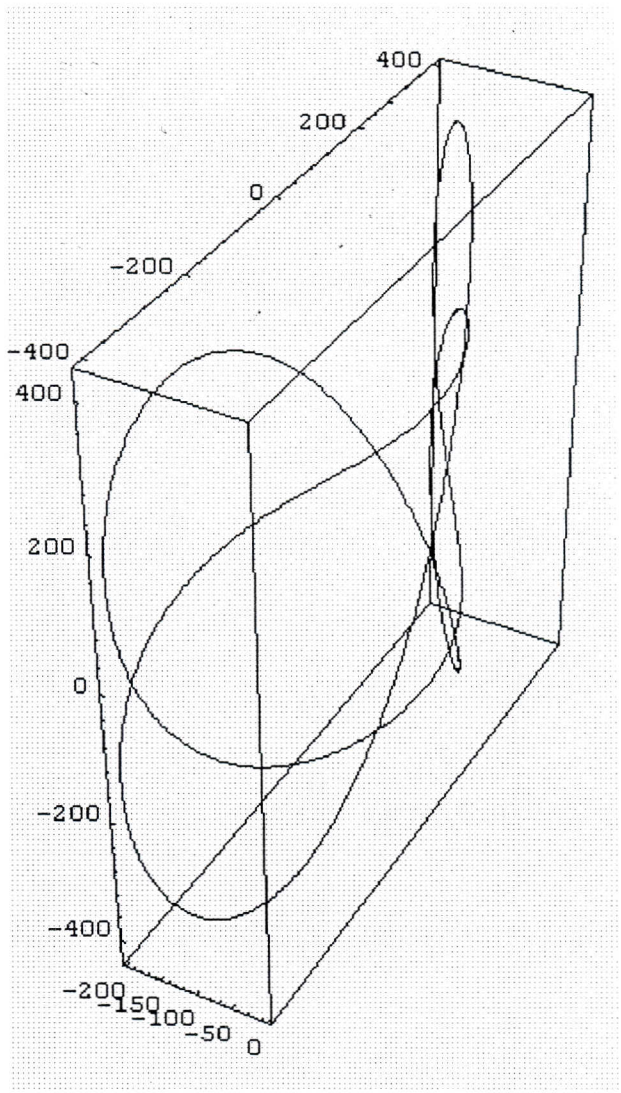


Figure 7.1 Trajectory of the periodic motion described by the set of equations { (7.1), (7.2) and (7.3)} (Graphic output form Mathematica v.2.2)

7.2.2 Preprocessing of Motion Data

In order to feed motion data to neural networks, they have to be pre-processed The reason is that operational range of a neural network is determined by the activation function mapping range. If activation function for neurons in the networks is

$$\Psi(u) = \frac{1}{1 + e^{-\gamma u}}, \qquad (7.4)$$

then operational range of neural network is [0,1]. If selection of activation function for output nodes is made as

$$\Psi(u) = \tanh(u) = \frac{1 - e^{-2\gamma u}}{1 + e^{-2\gamma u}}, \qquad (7.5)$$

then operation range extends from -1 to 1. Because the range is wider for the second case it can be more precise for the application. Besides, negative values can be mapped to interval [-1,0] resulting in more accurate representation of parameters. In the neural network literature, mapping real values of a quantity into representative values lying in operational range of network is called normalization operation. More precisely, if m is a measurement of a quantity M, normalized form of m is

$$m_{norm} = \frac{m}{m_{max}} \qquad (7.6)$$

where m_{max} is the maximum possible value that m can take. If m has negative values, so does m_{norm}. It can easily be deduced from equation (7.6) that $m_{norm} \in [-1,1]$.

Component of position vector of an object can have unbounded magnitude. That is, maximum magnitude of any component cannot be known in advance. Lack of information about absolute maximum value for magnitude of position makes normalization impossible. Under these circumstances, another representative parameter set has to be used instead of absolute magnitude of positions.

Position difference vector can't have magnitude larger than a certain value[Payeur et. al., 1995]. Let X_x be denoting x-component of position vector. If the ΔX_x is the relative difference between two position vectors, it can be stated that $|\Delta X_{x_{max}}| \geq |\Delta X_x|$, where ΔXx_{max} is the maximum possible magnitude of the x-component of position difference vector. The value of ΔXx_{max} depends on the amount of energy that can be converted into kinetic energy form in the time interval ΔX takes place. The fact that this energy is limited is a sound justification for the above inequality.

The first phase of processing data is differencing successive position values. If n positions are present then differencing process yields n-1 values. Beside satisfying the condition mentioned above, differencing operation is useful in the sense that it removes linearity in the signal sequence which yields simpler sequence[Masters, 1994]. Figure 7.2. a and 7.2.b illustrate the operation by presenting before and after phases.

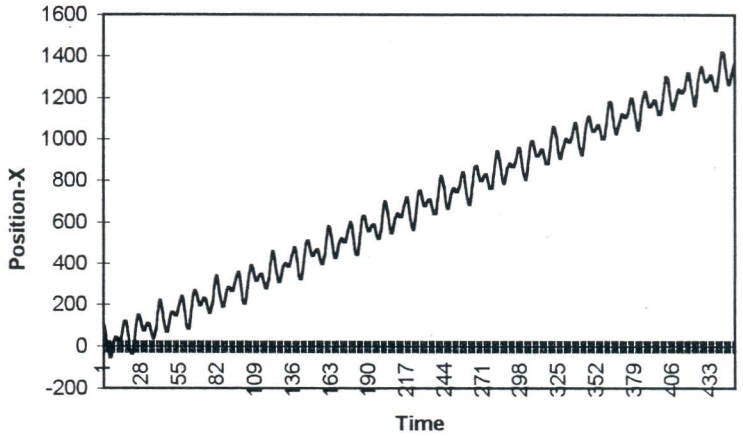


Figure 7.2.a A time series containing linear trend
(Graphic output from MS Excell v7.0. Software)

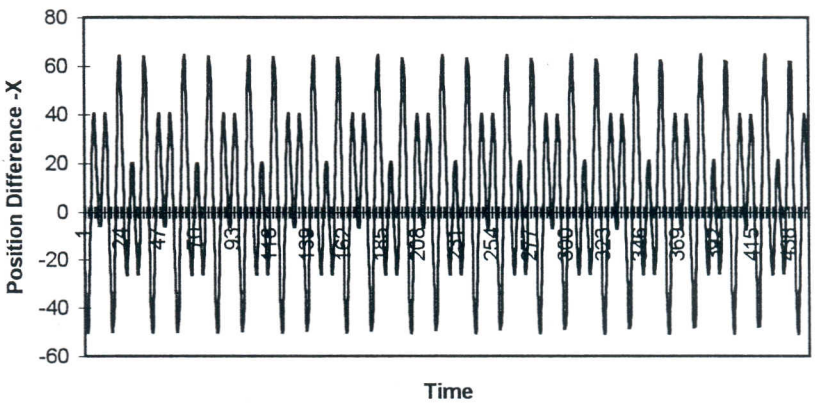


Figure 7.2.b Differencing operation removed linear trends in the sequence(Figure 7.2.a)
(Graphic output from MS Excell v7.0. Software)

Once prediction mechanism estimates difference between any two successive positions the real value of predicted position can be obtained by adding predicted difference value to preceding position value.

Although differencing operation does not remove any linear trend, because no linear trend is existing in any component of object position vector in the motion model described by equations (7.1), (7.2), (7.3), it is still necessary to apply it in order to make normalization process possible. In fact, presuming no linearity in the sequence contradicts with the claim

that prediction mechanism can make short-term estimation without any priori knowledge about motion model.

Table 7.2 contains the difference values between successive positions. Note that the sequence length is decreased by 1.

Table 7.2 Upper and lower bounded sequence obtained by applying differencing operation.

Time Interval No	ΔX(t)	ΔY(t)	ΔZ(t)
0	-0.78773	25.10349	50.08331
1	-2.35079	25.00452	49.14728
2	-3.8768	24.80697	47.29562
...	...	...	...
...	...	...	...
...	...	...	...
446	5.665099	-24.4306	10.85093
447	4.217377	-24.7481	8.241763
448	2.703211	-24.9681	5.354558

In the Table 7.2 $|\Delta X(t)| < 13$ for all t. So normalized values $\Delta X(t)_{norm}$ can be obtained as

$$\Delta X(t)_{norm} = \frac{\Delta X(t)}{13} \qquad (7.7)$$

Although operation range of neural networks that are going to be used in the prediction process is [-1,1], in many applications it is observed that values -1 and 1 are never reached. If normalizing factor is chosen to be the maximum value of sequence it is apparent that the maximum sequence element cannot be reached . Thus , normalizing factor must have a value larger than 13 , say 16.

$$\Delta X(t)_{norm} = \frac{\Delta X(t)}{16} \qquad (7.8)$$

More generally,

$$\Delta X(t)_{norm} = \frac{\Delta X(t)}{(1+a) * \Delta X(t)_{\max}}, \quad where \quad 0 < a < 1. \qquad (7.9)$$

Table 7.3 tabulates normalized position difference in x , y and z coordinates with their respective normalization factors.

Table 7.3 Normalized positions with normalizing factors

Normalization factor	16	31	60
interval no	$\Delta X(t)_{norm}$	$\Delta Y(t)_{norm}$	$\Delta Z(t)_{norm}$
0	-0.04923	0.80979	0.834722
1	-0.14692	0.806597	0.819121
2	-0.2423	0.800225	0.78826
...	...	...	...
...	...	...	...
446	0.354069	-0.78808	0.180849
447	0.263586	-0.79833	0.137363
448	0.168951	-0.80542	0.089243

7.2.3 Preparation of Training and Testing Patterns

Training pattern has been prepared in the form of shifted sequence as in Table 7.3. The last element of each row is the value that has to be predicted in the estimation procedure.

Table 7.4 List Pattern for training and testing with n last value of ΔX_{norm} s

Pattern No(p)	$\Delta X(t-(n)T)_{norm}$	$\Delta X(t-(n-1)T)_{norm}$	$\Delta X(t-(n-2)T)_{norm}$	...	$\Delta X(t)_{norm}$	$\Delta X(t+T)_{norm}$
0	-0.04923	-0.14692	-0.2423		-0.68724	-0.72922
1	-0.14692	-0.2423	-0.33386		-0.72922	-0.75971
. . .	. . .	. . .	. . .	. . .	. . .	. . .
M	0.765002	0.73721	0.697804		0.26358	0.168951

The pattern M, (M-2), (M-4) and (M-6) are used for testing the performances of neural networks in the simulation runs. If the testing pattern used in a simulation run has number K, then training sample set for that run is formed with the patterns located in the range 0 to K-1. Since the size of training patterns vary only very little, the difference in the performances cannot be attributed to the different training sample sizes. This is the basic reason for choosing patterns clustered at the end of whole pattern list.

7.2.4 Training Scheme and Testing Procedure

Two well-known neural network architectures have been employed in the prediction processes. The first one is recurrent neural network and the second one is Jordan/Elman network. Before deciding on these neural networks, their characteristics and performances have been compared to those of some other neural network models namely MLP (multilayer perceptron), PCA(Principle Component Analyzer) neural network etc.

The learning rule for both neural network models has been chosen as backpropagation with momentum update in order to speed-up the learning process..

Recurrent neural network architecture with fully recurrent connection for its first hidden layer has been created . Number of hidden layers in the network is determined to be only one. Testing has been performed during the training.

$$E(s) = \frac{1}{2}\left(d(s) - \Delta X(s)\right)^2 \qquad (7.10)$$

where d(s) is the real value and $\Delta X_i(s)$ is the estimated value for normalized position difference at training step s . The error is measured for both training data and testing (cross validation) data.

Two different embedding dimensions have been tried in the experiments. The embedding dimensions for which the simulations have been carried out are 4 and 9 respectively. Changing embedding dimension has resulted in different performances for both neural network models.

Figure 7.3.a and 7.3.b illustrates effect of changing embedding dimension for the prediction of normalized position difference in X axis. It is clear that increasing the number of input nodes which means increase in embedding dimension results in estimations with higher precision for periodic motion.

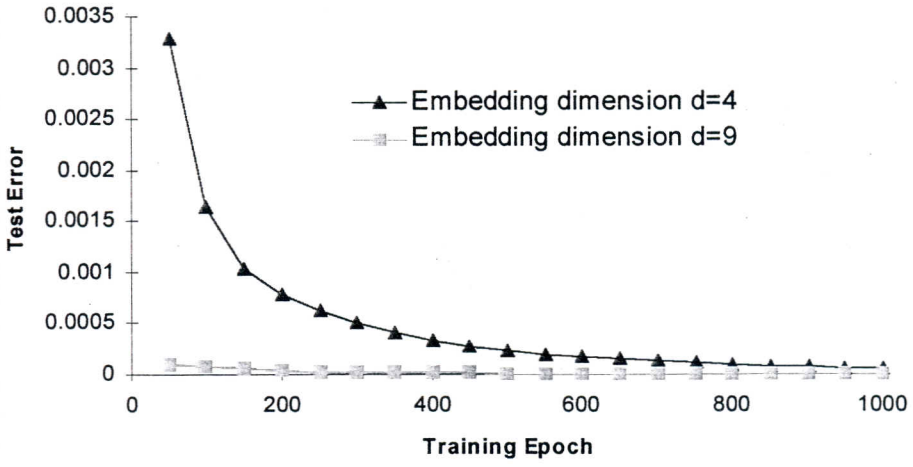


Figure 7.3.a Estimation error functions in regard to training epoch when recurrent neural networks with 4 and 9 input nodes are employed.

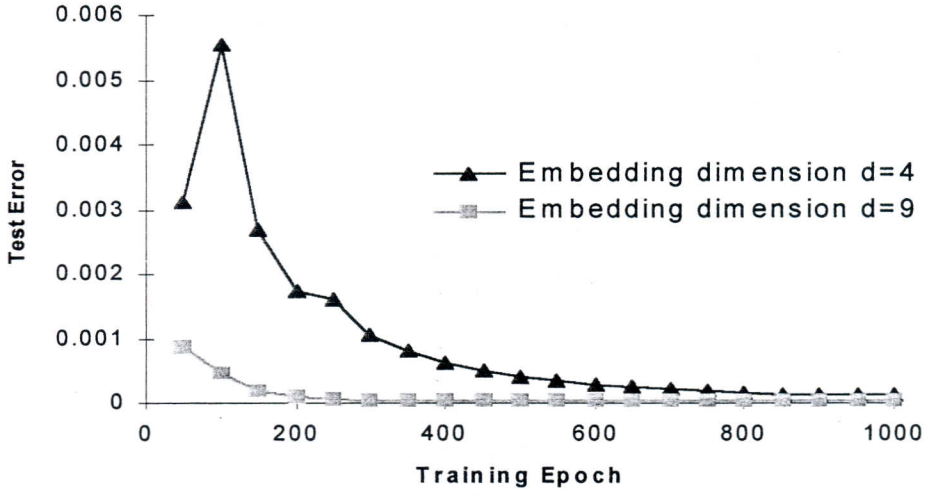


Figure 7.3.b Estimation error functions in regard to training epoch when Jordan/Elman neural networks with 4 and 9 input nodes are employed

The first recurrent neural network used in the experiments has 4 input nodes. Inputs are simple axons which simply perform an identity map between its input and output activity. The delay time τ between inputs is $\pi/25$ second (125.6 ms). The single hidden unit in the network consists of 8 nodes with hyperbolic tangent (tanh) activation function. Firing level of single output node represents normalized form of predicted ΔX value which is used

to calculate next position of the object. Learning rate for weights connecting input layer to single hidden layer in the network has been chosen as 1. For the recurrent weights the learning rate is set to value 0.1. The value for momentum, on the other hand, has been set to be 0.7 for all weights. The second recurrent network has only one hidden layer as the first one. The number of nodes in the input layer 9. Number of nodes in the hidden layer has been increased to 18. On the hand, learning parameters values of the network have been set to those of first recurrent neural network.

The first Jordan/Elman Network used for time series prediction with embedding dimension d=4 and time delay τ=125.6 ms has 4 input nodes. The first and second hidden layer contains 8 and 4 nodes respectively. Backpropagation rule with momentum update has been employed for training and constant for momentum term is defined to be 0.7. The learning rate for weights connecting input nodes to first hidden layer is 0.4. For weights connecting two hidden layers in the network the learning rate is set to 0.3. As for the weights between last hidden layer and single output node, the learning rate is 0.2. It is necessary to mention here that these values have determined throughout experiments. The second Jordan /Elman network configuration contains 9 nodes in input layer. For first and second hidden layers the sizes have been determined to be 18 and 9 nodes respectively. However, the same values for learning parameters of first configuration have been used throughout the training process.

Experiments have shown that recurrent networks outperform the Jordan/Elman neural networks in predicting normalized position difference of an object that is subject to periodic motion. It should be noticed that recurrent network is more compact than the Jordan/Elman network which means less number of weights and less computation to update them. Let us see the performance curves of these neural networks when the embedding dimension is 9. An average performance measurements based on error criterion defined with equation (7.10) are plotted in the Figure 7.4.

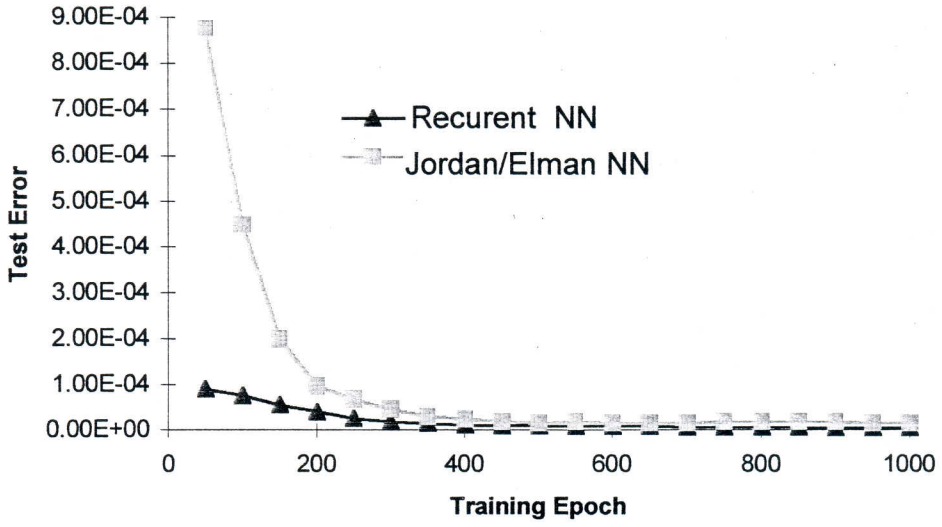


Figure 7.4 Comparison of Recurrent and Jordan/Elman neural networks on their position change predicting capability for a periodic motion. (embedding dimension $d = 9$)

7.3 Nonlinear Attractor Motion Model

In this section prediction for chaotic motion model will be discussed in order to measure the prediction capability of neural networks with more complicated motion time series. Motion types displaying chaotic characteristics have been modeled using the famous chaotic formulations done in the nonlinear dynamics literature. The first formulation is Thinkerbell attractor for which several neural network topologies have been used to perform predictions. The second one is the chaos described by Lorenz equations.

7.3.1 Tinkerbell Atractor Motion Model

Tinkerbell attractor is defined with the mapping function below.

$$F\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x^2 - y^2 + 0.9x - 0.6013y \\ 2xy + 2x + 0.5y \end{pmatrix} \qquad (7.11)$$

The attractor studied by James York is plotted in the Figure 7.5. A planer chaotic motion can be modeled by using the formulation done by James York.

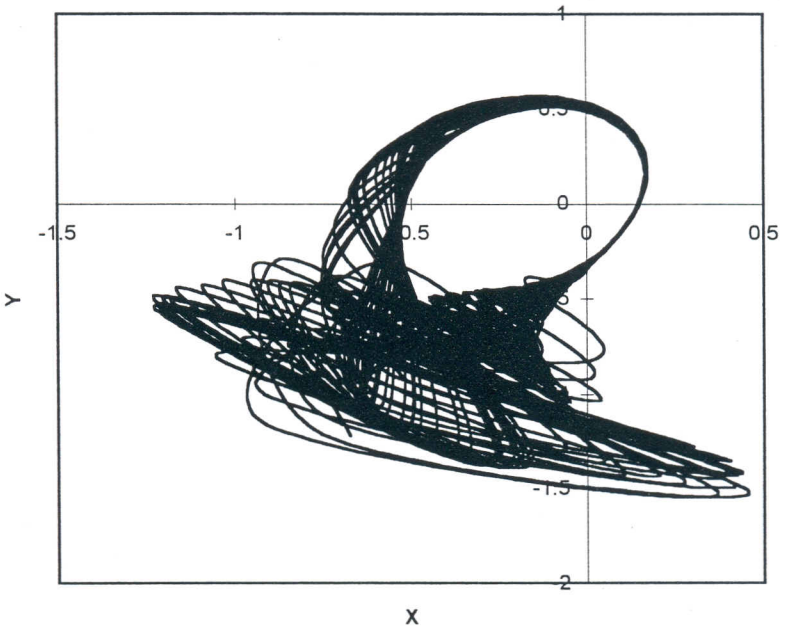


Figure 7.5 Tinkerbell attractor can be a model for a 2-D motion

7.3.1.1 Generation of 2-D Motion Time Series Data

Since the Tinkerbell attractor is used as motion model, two dimensional position of an object can be obtained by applying the recursive nonlinear formulation done with equation (7.11). After trying many values as initial conditions, x and y have been set to values -0.5 and 0.45 respectively. The reason for choosing small values as an initial conditions is that using large values yield very large values after a number of iterations such that computers used in the data generation generate overflow error.

7.3.1.2 Preprocessing of Training and Testing Patterns

As in the section 7.1 differencing operation has been applied on the data produced by iterating equation 7.11. The length of training time series is 400. Embedding dimension in the prediction is 9 for X and Y. That is, it is not used as a sample training by the neural network while it is being trained. The aim is to observe the predicted values of the next position difference while the network is being trained . The graph of position differences in x and y coordinates are plotted in the Figure 7.6.a and Figure 7.6.b respectively.

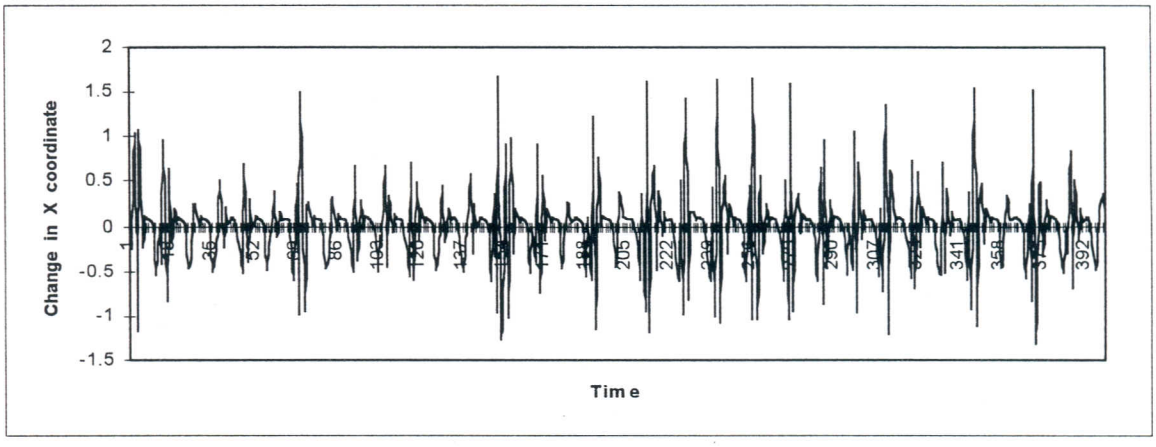


Figure 7.6.a Plot of change in x coordinate of object position

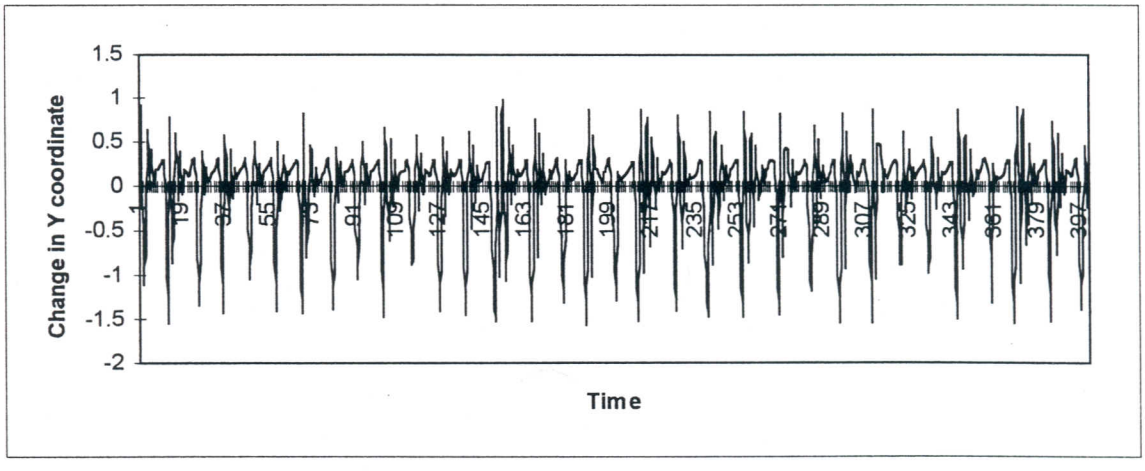


Figure 7.6.b Plot of change in y coordinate of object position with respect to time

7.3.1.3 Training Scheme and Testing Procedure

During the training process, neural networks have been tested against the real value of the next position difference in x (y) axis. The error has been measured for both training data and testing (cross validation) data using the equation 7.10.

The first studied configuration is Jordan/Elman neural network architecture with input nodes fed with the last 9 x component (y component) of position differences. At the output node prediction for the next position difference has been obtained. Two hidden layers have 18 and 9 nodes respectively. The backpropagation with momentum learning rule has been used for the training of the network. The learning rate used in updating the weights connecting input layer and first hidden layer is 0.4. Weights between two hidden layers have been updated with learning rate of 0.3. As for the weights between the last hidden layer and single output node, the learning rate has been chosen to be 0.2. The maximum iteration size for the training has been set to 1000 epochs.

The experiments have been carried out with recurrent neural network for the same testing patterns . The network has 9 input nodes and it is fed with last 9 normalized position difference in x coordinate. The first of two hidden layer has 18 nodes. The second one has 9 nodes. The firing rate of output node is considered as the predicted normalized difference in successive values of the parameter to be estimated . Backpropagation with momentum learning rule has been employed to train the network. Momentum all weights has set to be 0.7. Learning rate for weights between input and first hidden layer is 0.4. Weights between hidden are updated with learning rate of 0.3. As for the weights connecting second hidden layer to single output neuron has been reevaluated with the learning rate of 0.2. Another feedforward connection stream between the input layer and second hidden layer has been created and learning rate for this weight stream is set to 0.4. The only recurrent weight connection stream from first hidden layer to itself has the learning rate equal to 0.1. The recurrent connection stream has been chosen as partial rather than a full connection type because of instability problem associated with fully recurrent connections.

Testing patterns have been determined in the same way in periodic motion experiments. Testing error for every 50 epochs has been measured and plotted for each

simulation runs. The average performance curves for recurrent network and Jordan Elman network are plotted in Figure 7.7.

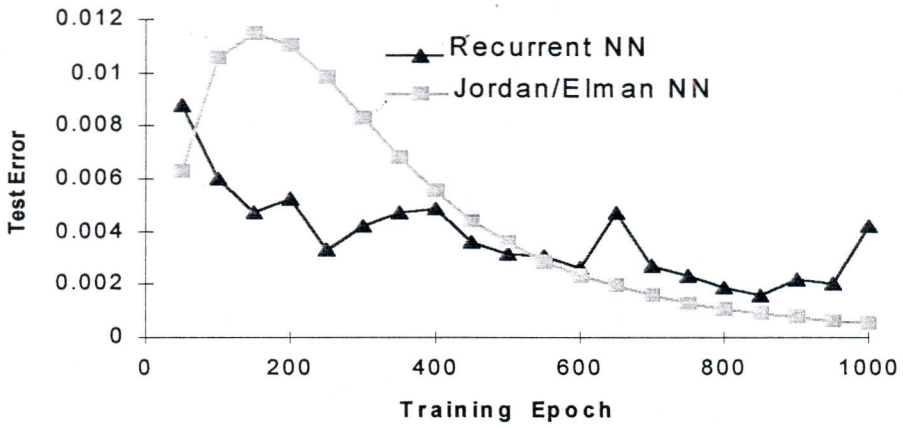


Figure 7.7 Comparison of Recurrent and Jordan/Elman neural networks on their position change predicting capability for Tinkerbell Attractor motion model . (embedding dimension d = 9)

According to the Figure 7.7, it can be said that prediction with Jordan/Elman networks is more reliable than that with recurrent network for Tinkerbell-like chaos case. It should be noticed that after 800-1000 epoch a reasonable accuracy can be attained with Jordan/Elman networks.

7.3.2 Lorenz Attractor Motion Model

Lorenz attractor is a famous model in the nonlinear dynamics literature. It was used to model Far-infrared (FIR) laser beam behavior and fluid convection phenomena. Its natural aspects and 3 dimensional property have made it an interesting motion model to be analyzed.

The characteristic equations describing Lorenz attractor are:

$$\frac{dX}{dt} = \sigma Y - \sigma X \ , \qquad (7.12.a)$$

$$\frac{dY}{dt} = -XZ + rX - Y \ , \qquad (7.12.b)$$

$$\frac{dZ}{dt} = XY - bZ \ . \qquad (7.12.c)$$

The model used in this study has the form

$$\frac{dX}{dt} = 12Y - 12X \qquad (7.13.a)$$

$$\frac{dY}{dt} = -XZ + 60X - Y \qquad (7.13.b)$$

$$\frac{dZ}{dt} = XY - \frac{8}{3}Z \qquad (7.13.c)$$

with initial conditions X(0)=1, Y(0)=1, Z(0)=1.

3-D plot of trajectory described by the equations (7.13.a), (7.13.b) and (7.13.c) is presented in Figure 7.7.

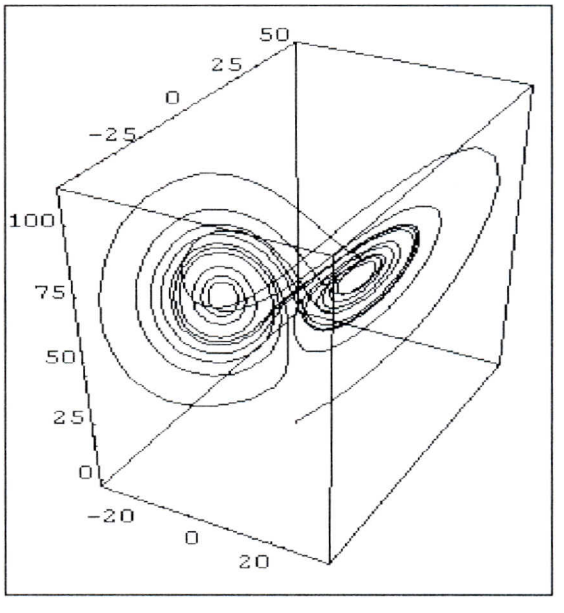


Figure 7.8 A Trajectory of Lorenz Attractor Motion Model

(Graphic output from Mathematica vers.2.2)

7.3.2.1 Generation of 3-D Motion Time Series Data

The equations (7.13.a), (7.13.b) and (7.13.c) with initial conditions are used to generate time series data for X, Y and Z coordinates separately. The starting position of the motion has assumed to be (1, 1, 1) in xyz cartesian coordinate system. The length of generated time series is 400 for each axis.

Selection of testing patterns has been done in the same way done for periodic motion case.

7.3.2.2 Preprocessing of Training and Testing Patterns

As in the section 7.1 differencing operation has been applied on the data produced by solving iterating equations 7.13 with initial conditions supplied. The aim is to observe the predicted values of the next position difference while the network is being trained . The same procedure in the previous motion model has been followed for selection of testing patterns. The graph of normalized position differences in x, y and z coordinates are plotted in the Figures 7.9.a , 7.9.b and 7.9.c respectively.

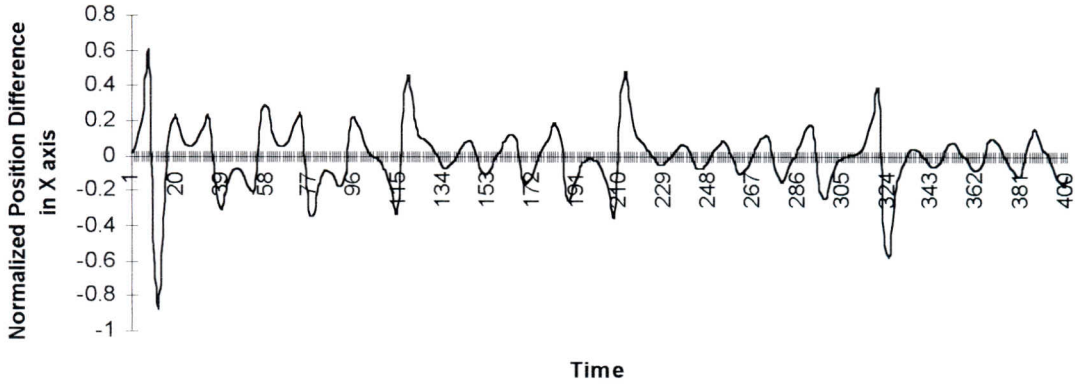


Figure 7.9.a Plot of change in x coordinate of object position

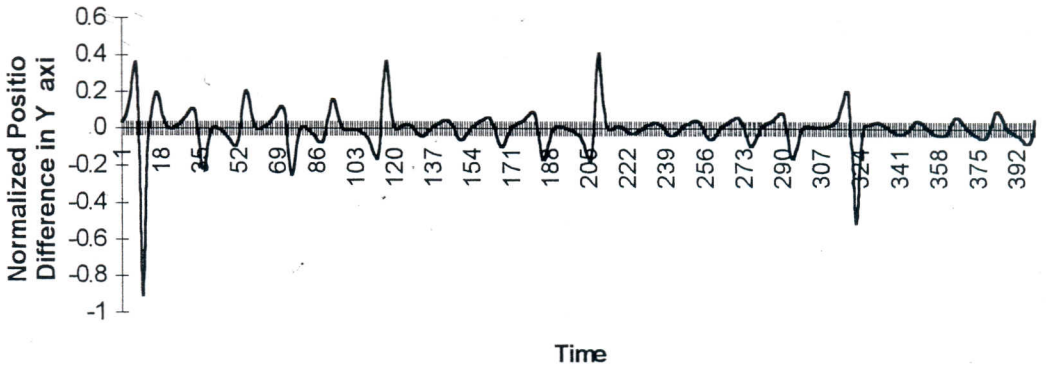


Figure 7.9.b Plot of change in y coordinate of object position

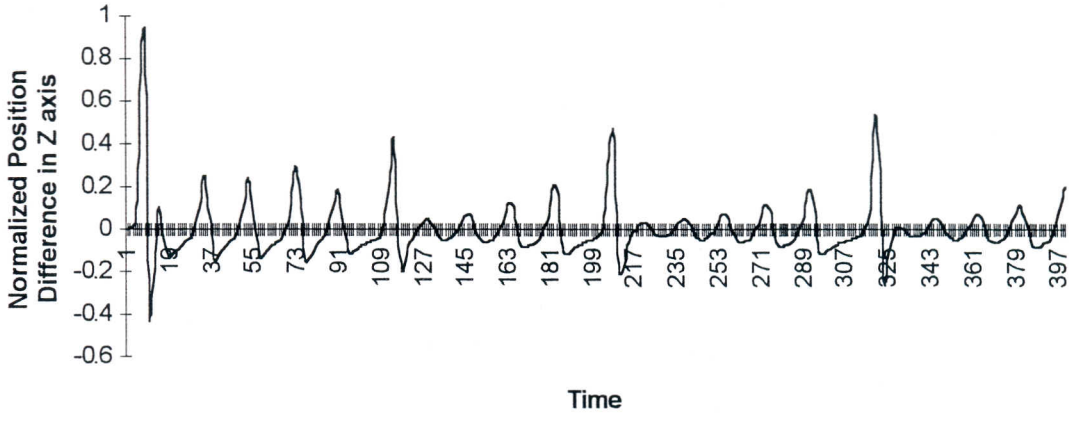


Figure 7.9.c Plot of change in z coordinate of object position

7.3.2.3 Training Scheme and Testing Procedure

The procedure used in prediction of Lorenz motion parameters is no different from the previous ones. As in the previous cases, two neural networks namely recurrent neural networks and Jordan/Elman neural networks have been employed in the predictions. Embedding dimension for the prediction has determined to be 9. The networks used in the predictions are not very large. Only one hidden layer has been used in each network topology. Both network has 9 input nodes, 18 hidden layer nodes and only one output nodes.

Each input node of the recurrent network is simple axon which makes unity map. On the other hand, each input node of the Jordan/Elman network is a tanhintegrator axon which implements normalized feedback mechanism by in the Jordan/Elman architecture by integrating with a time constant the activity received by each PE in the layer.

For training, backpropagation with momentum update rule has been used for both network topology. The momentum value for all weights in both neural network topology has been set to be 0.7.

Weights connecting input nodes to hidden layers have been trained with learning parameter 1 for the recurrent neural network. On the other hand, in order to obtain good results, the counterparts of these weights in Jordan/Elman network has been updated with learning parameter 0.1.

The weights between the hidden layer and output layer of both network have trained with learning parameters set to the value 0.1.

Figure 7.10 illustrates comparison of recurrent and Jordan/Elman neural networks on their position change predicting capability (in x coordinate axis) for Lorenz motion model.

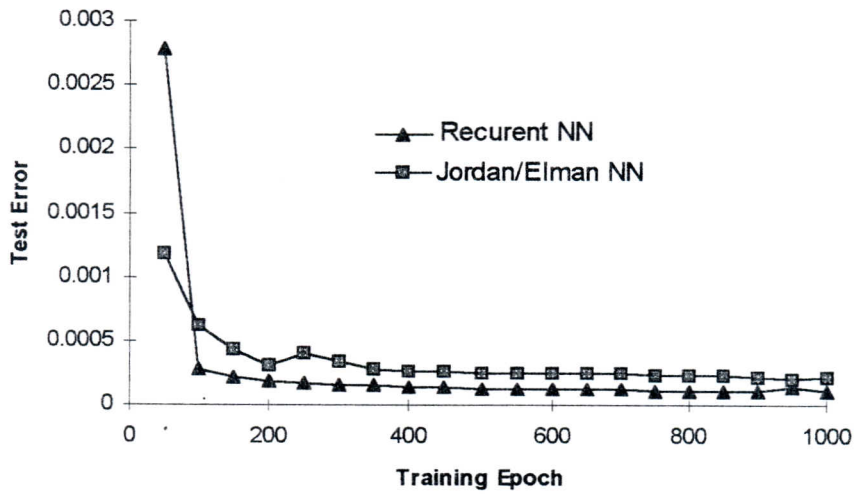


Figure 7.10 Comparison of Recurrent and Jordan/Elman neural networks on their position change(in x coordinate axis) predicting capability for Lorenz attractor motion model . (embedding dimension d = 9)

In the graph, test error for recurrent network is smaller. It should be noticed that testing error is in the order of 10^{-4} after 500 epochs.

Chapter 8

CONCLUSIONS AND FUTURE WORKS

8.1. Conclusions

Problem of prediction of motion parameters has been studied in the context of time series prediction. Throughout the study, characteristics of time series have been assessed. The methods to analyze and model time series have been reviewed. Their advantages and disadvantages have been discussed.

Since the aim of the study is to predict motion parameters, the set of parameters which are to be estimated have been identified and the mathematical relations between these parameters have been stated.

In order to use neural networks in the motion prediction problem, the neural network solution paradigm has been explained by introducing properties of various neural network topologies. The learning rules that are used to train neural networks are examined and the strategies to construct an effective neural network have been assessed by experimental studies.

Mathematical models describing trajectories have been analyzed and those models which are suitable to describe a motion have been selected for the study.

The neural network topologies namely PCA(Principle Component Analysis), RBF(Radial Bases Function), MLP (Multilayer Perceptron), Jordan/Elman and Recurrent Networks have been tested for fast and accurate prediction of motion parameter. The outstanding neural network topologies in these experiments have been selected to be used in the study.

The factors affecting the performance of predictions have been worked out and following results have been obtained:

- Provided with high performance hardware, estimation of complex trajectories via neural networks is possible.

- Recurrent networks can make good predictions for smooth curves after reasonably short training period.
- Although the convergence of Jordan/Elman networks is slower, it has been observed this network shows better performance with time series displaying fast fluctuations.
- Overtraining of neural networks decreases prediction accuracy. However, the number of iteration at which learning has to be stopped varies from one time series type to another.
- The larger the size of neural networks the greater the number of iterations required to obtain predictions with a predefined accuracy.
- The resolution of measurements has significant effect on precision of estimations. For this reason motion measurement systems has to operate to keep up with frequency of measurement requirements (resolution).

8.2. Future Works

The sequences of difference of motion parameters constitute fairly complicated time series. As the complexity increases it becomes more difficult to identify the dynamics of system producing the time series. For most of the time series, equally spaced signal amplitudes does not provide deep insight to a neural network model about the dynamics of the process. It is known that if neural networks are provided with frequency information of a signal, then their identification capability increases. However, presenting only frequency information obtained by Fourier-like transformations are not effective due to the lack of information about frequency localization. Based on these facts a new training scheme should be developed such that both time and frequency information can be presented to the network. Transformations that are providing both type of information are Short-Term Fourier Transformation, Gabor Transformation, Wavelet Transformation, etc. The most feasible transformation should be selected and training should be performed in complex number domain.

Another strategy to improve the accuracy of predictions may be cross validation of predictions. For instance, the prediction value for acceleration can be used to calculate the predicted value of velocity by numerical integration methods. Moreover, predictions for position should be used to calculate the predicted velocity by numerical differentiation. The value obtained from different predictors should be combined to obtain estimations with higher precision.

Another point in the prediction problem is including all the variables that has affect on the evolution of the system. The mechanism briefly explained in section 7.1 should be implemented. The cost of maintaining this mechanism and contribution of the mechanism to the accuracy of motion parameter predictions should be reconciled.

References

1 **[Amari , 1991]** S. Amari, "Mathematical theory of neural learning", New Generation Computing, vol. 8, pp 281-294, 1991.

2 **[Bateman, 1995]** Andrew Bateman, "A useful and flexible tool for kinematics analysis of action", Psychology Software News, Volume 6, Number 1, September, 1995

3 **[Bengio et. al., 1995]** Samy Bengio, Françoise Fessant, Daniel Collobert , "A connectionist system for medium-term horizon time series prediction", France Télécom, Centre National d' Études des Télécommunications, LAB/RIO/TNT, 1995.

4 **[Chichocki, Unbehauen, 1993]**A. Cichocki, R. Unbehauen, Neural Networks for Optimization and Signal Processing, 1993, John Wiley & Sons, pp 448-449.

5 **[Chen et al, 1994]** Wei-ge Chen, Georgios B. Giannakis and N. Nandhakumar, "Spatio-temporal motion estimation", Proc. 1st IEEE Inter. Conf. on Imag. Proc. 1994.

6 **[Çınar, 1994]** Ali Çınar, "Nonlinear time series models for multivariable dynamic processes", Department of Chemical Engineering, Illinois Institute of Technology", 1994.

7 **[Debrunner and Ahuja, 1995]** Christian H. Debrunner and Narendra Ahuja, "A direct data approximation based motion estimation algorithm", Psychology Software News, Volume 6, Number 1, September, 1995.

8 **[Elman, 1990]** Elman J. "Finding structure in time." Cognitive science 14, pp. 179-211, 1990.

9 **[Goerke and Eckmiller, 1993]** Goerke, N.H.-R., Eckmiller, R. "A method of error-driven neuron placement for trajectory learning", Department of Computer Science VI- Neuroinformatik-University of Bonn,1993.

10 **[Gulick,1992]** Gullick, Denny, Encounters with Chaos, McGraw-Hill, pp 1-2, 1992

11 **[Jordan, 1986]** Jordan M. "Attractor dynamics and parallelism in a connectionist sequential machine." Proc. 8th Annual Conf. on Cognition Science Society, pp 531-546, 1986.

12 **[Kohonen, 1988]** Kohonen T. Self Organization and Associative Memory. Springer Verlag, 1988.

13 **[Lim Ni Fu, 1994]** LiMim Fu, ,Neural Networks in Computer Intelligence, McGraw-Hill pp. 169-185, 1994.

14 **[Lorentz, G.G.,1976]** Lorentz, G.G , "The 13th problem of Hilbert. in Mathematical Developments Arising from Hilbert Problems ", American Mathematical Society, Providence,RI, 1976.

15 **[Masters, 1994]** Masters Timothy, " Signal and Image Processing with Neural Networks", John Wiley & Sons, pp. 62-72,1994.

16 **[Makhoul, 1992]** Makhoul J. "Pattern recognition properties of neural networks." Proc. 1991 IEEE Workshop on Neural Networks for Signal Processing, pp 173-187, 1992.

17 **[Packard et. al., 1980]** Packard, N.H., J.P. Crutchfeld, J.D. Farmer, and R.S. Shaw. 1980. " Geometry from a time series", Phys. Rev. Lett. 45(9) 712-719.

18 **[Payeur et. al., 1995]** Pierre Payeur, Hoang Le-Huy, Clement M. Gosselin, "Trajectory prediction for moving objects using artificial neural networks", IEEE Transaction on Industrial Electronics, 1995.

19 [Principe et. al., 1991] Jose C. Principe, Alok Rathie anfd Jyh-Ming Kuo,"Prediction of chaotic time series with neural networks", Electrical Engineering Department, University of Florida, Gainesville, 1992.

20 [Principe et. al., 1996] Jose Principe, Curt Lefebvre, Gary Lynn and Dan Wooten, " Neurosolutions Documentation", Famous Neural Network Toplogies, 1996.

21 [Sears et. al., 1987] F.W. Sears, M. W. Zemansky, H. D.Young, University Physics -Seventh Edition, Addison-Wesley Publishing Comp. , pp 26-41, pp 210-212, 1987.

22 [Sietsma, 1991] J. Sietsma and R.J.F. Dow, "Creating artifical neural network that generalize", Neural Networks, vol 4,1991,pp 67-79.

23 [Weighend, Gershenfeld, 1994] Andreas S. Weighend, Neil A. Gershenfeld, Time Series Prediction, Forecasting the Future and Understanding the Past, Addison-Wesley Publishing Company , pp 2-5, pp 16-17, 1994.