Connections, neurons and activation : the organization of representation in artificial neural networks

Connections, Neurons and Activation

The Organization of Representation in Artificial Neural Networks

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Henseler, Johan

Connections, neurons and activation : the organization of representation in artificial neural networks / Johan Henseler. - [S.l. : s.n.]. - Ill.
Proefschrift Maastricht. - Met lit. opg. - Met samenvatting in het Nederlands.
ISBN 90-9006624-1
NUGI 855
Trefw.: neurale netwerken / computers / robotica.

Voor Jolanda, Rutger en Matthijs

Connections, Neurons and Activation

The Organization of Representation in Artificial Neural Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Rijksuniversiteit Limburg te Maastricht,
op gezag van de Rector Magnificus, Prof.dr. H. Philipsen,
volgens het besluit van het College van Dekanen,
in het openbaar te verdedigen
op donderdag 16 december 1993 om 16.00 uur

door

Johan Henseler

Promotor:

Prof.dr. H.J. van den Herik

Co-promotor:

Dr. P.J. Braspenning

Beoordelingscommissie:

Prof.dr.ir.drs. O.J. Vrieze (voorzitter)
Dr. P. Boekhoudt
Prof.dr. P.T.W. Hudson
Prof.dr.ir. B.H. Jansen (University of Houston, Texas, USA)
Prof.dr.ir. J.P.L. Vandewalle (Katholieke Universiteit Leuven)

Preface

This dissertation addresses the internal organization of representations in Artificial Neural Networks (ANNs). The research presented has been carried out at the Department of Computer Science of the University of Limburg, Maastricht, the Netherlands. The work reflects the development of new concepts and ideas from a computer-science perspective. The research is characterized by its interdisciplinary nature, combining work in the fields of ANNs, Neurophysiology, Pattern Recognition, Mathematics and Physics.

ANNs are models of biological neural networks in the brain. The first publications on ANNs date back as far as 1943, i.e. the work of McCulloch and Pitts (1943). However, in the late 1960s and 1970s, researchers lost their interest in ANNs and turned to rapidly growing fields offering similar prospects, e.g. Machine Learning, Pattern Recognition and Artificial Intelligence. This development was mainly due to the performance increase of the digital computers and the expectations raised; compared to them neural networks seemed obsolete. In the 1980s, when the expectations were not fulfilled, interest in ANNs was renewed. Meanwhile, small groups of persistent researchers had independently continued their ANN research, solving some of the earlier problems and developing new and powerful concepts that actually worked. Hence, the revived ANN-research community was catapulted into a rapid of annual international conferences, monthly international journals and governmentally-funded research projects. Admittedly, this dissertation is a small part of the rapid and I gratefully acknowledge the ideas of all those other researchers who have contributed to its realization.

Furthermore, this research would not have been possible without the help of

many other persons. First of all, I would like to thank Jaap van den Herik for the lessons learned and for the opportunities he created for me. I want to thank Peter Braspenning for his enthusiasm and our long talks which were an inspiration for new ideas essentially leading to the present results. My special thanks go to Eric Postma, who proved to be a knowledgeable colleague always willing to discuss new ideas, and to Harm Bakker for our fruitful collaboration. The final version of this dissertation has benefitted from valuable suggestions by Ben Jansen, Patrick Hudson, Piet Boekhoudt, Koos Vrieze and Joos Vandewalle. In addition to the English scrutiny of Jaap van den Herik and Patrick Hudson I received many valuable comments from Ton Broeders, who put the odd finishing touch to the English text. Also, I want to thank both the staff of the Computer-Science Department of the Delft University of Technology as well as my friends at Fygir in Rijswijk for letting me use their computer facilities, and my colleagues at the National Forensic-Science Laboratory in Rijswijk who enabled me to finish this dissertation.

Finally, I want to thank my parents and all other relatives and friends who always showed a stimulating interest in my research. However, their support pales before the support I have received from my wife Jolanda and my two sons Rutger and Matthijs.

Hans Henseler
Delft, October 1993

Contents

Chapter 1

Introduction

Neural Networks (NNs) in the brain represent the world in a very abstract way. This thesis is concerned with how representations are organized in Artificial Neural Networks (ANNs). These may be thought of as elementary models of parts of the brain. In this thesis three distinct approaches are taken to examine how representations of the outside world can be constructed and used. We know that at least one system, the mammalian brain, is very successful and, as such, serves as a source of inspiration for reverse engineering (Stark, 1993).

Neurally inspired systems may represent our best way of tapping the power we need to solve many problems in any reasonable amount of time. From the early days of digital computers there has been a continuous urge to increase their speed of execution. A computer with a single Central Processing Unit (CPU) operates sequentially, i.e. executing a program stored in memory step by step. Hence, the speed of a computer is determined, among other things, by the rate at which the individual steps are executed. During the past decades speed-up has been realized by a process of miniaturization, enabled by the fast progress in electronic component design, e.g. the transistor, very large scale integration (VLSI), wafer-scale integration, etc. Current supercomputers are capable of calculating 20 billion or so operations per second (Pool, 1992). Unfortunately, physical and economical constraints prohibit an unlimited increase of computation performance by miniaturization alone. One of the alternatives which is (believed to be) the-

oretically unlimited is the construction of parallel computers. Underlying the idea of parallel processing is the assumption that the processing speed may be increased by partitioning a program into subprograms that can operate concurrently. However, as it turns out, partitioning a program is a difficult *problem* that may require a radically different approach towards computing. This problem is illustrated in Appendix A which deals with massively parallel computers (MPCs).

When searching for a new computing approach it is essential to understand that modern digital computers still operate by the principles introduced forty years ago. The CPU executes a program stored in memory specifying a sequence of machine-dependent instructions to be performed on input data and on data produced internally. The memory/processor split was introduced in a lecture in 1949 by Von Neumann (Von Neumann, 1966). The idea has been and still is considered as a major breakthrough for the design of computer hardware and software. Due to this choice, however, about 97% of the silicon components in our modern computers sits idle during operation (Hillis, 1985).

Considering again the parallel programming problem, the question arises whether the memory/processor split should be maintained. For instance, the brain may be interpreted as a MPC containing billions of analog (non-Von Neumann) processing elements called neurons, operating in NNs. Apparently, it is possible for 1,000 billion elements to operate in parallel in a consistent and organized way. Moreover, the intelligent behaviour that emerges in the brain is still considered to be far beyond the artificial intelligence capabilities of any computer yet. Inspired by the parallel processing power of NNs in the brain and prompted by the promise of MPCs, we decided to study ANNs. After a period in which we assimilated the fundamentals of ANNs our research diversified. However, central theme to all our pursuits is the organization of representation in ANNs.

In this introduction we provide the reader with background information on both NNs in the brain and ANNs. In Section 1.1 facts and theories about the brain relevant for defining our research aims are described. Section 1.2 presents background information and also introduces a functional architecture for ANNs. By studying ANNs we *aim* at finding out more about the organization of connectionist representations formed by three entities that

can be identified in ANNs. In Section 1.3 the main problem of this dissertation is stated. Subsequently, three problem statements are formulated on which the research will be based. Finally, Section 1.4 gives an overview of the thesis by outlining the research approaches.

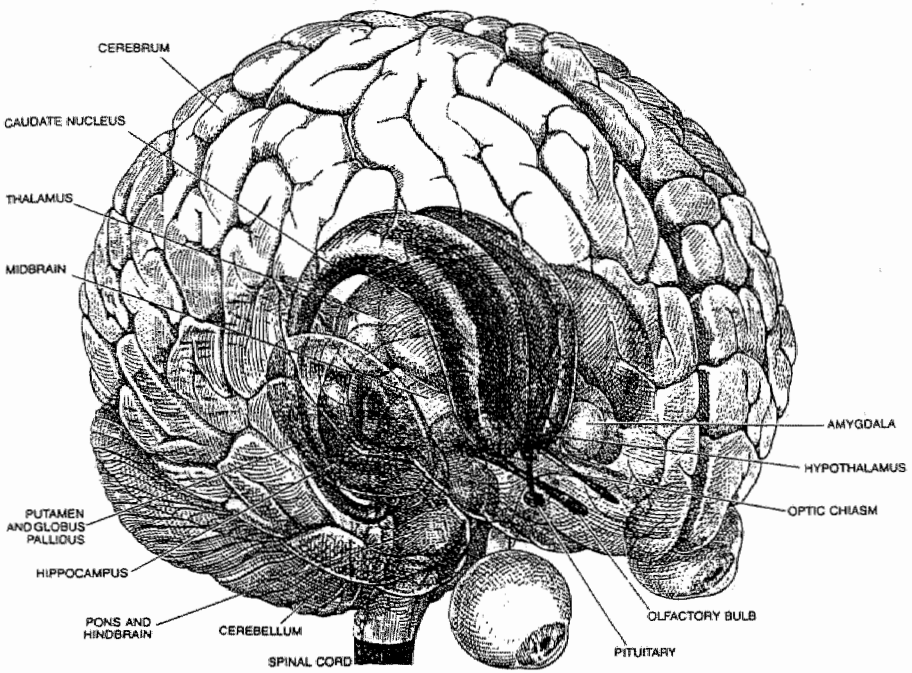


Figure 1.1: *Transparent view of the internal structures of the mammalian brain. The outer surface of the cerebrum is called cortex. (from Nauta and Feirtag, (1979)).*

1.1 Neural Networks in the Brain

The information presented in this section does not define the state of the art in brain studies. It only sets a scene providing the reader with background information on NNs in the brain. More detailed information can be found in for instance Kandel and Schwartz (1985) or Kolb and Whishaw (1990)

and for a comprehensive introduction (in dutch) see for instance Schadé (1983) and Schadé (1984). The brain is a complex nervous tissue in the skull functioning as centre of sensation, control and thought. The general structure of the mammalian brain is illustrated in Figure 1.1 (Nauta and Feirtag, 1979). The nervous tissue is structured as a densely interconnected network of nerve cells called *neurons*. Hence, such a network is called a *neural network*. The largest part of the human brain is the *cerebrum*, consisting of approximately 10^{12} neurons which are interconnected by about 10^{13} to 10^{16} interconnections. In comparison: a worm has 10^3 neurons, a fly has 10^7 neurons, a cockroach has 10^8 neurons and a bee has 10^9 neurons (DARPA, 1988).

Neurons influence each other through their interconnections. Roughly, a neuron consists of a cell body and some finer parts resembling branches of a tree. Two kinds of branches are distinguished. Branches responsible for receiving input from other neurons are called *dendrites*. The dendritic arborization is located near the cell body. Secondly, branches via which the neuron output is transferred are called *axons*. The axon length ranges from $100\mu m$ to $3mm$ in the brain and up to $1m$ in the spinal column, before terminating on the dendrites of one or more destination neurons. The meeting point of an axon terminal fibre and a dendrite is called a *synapse* (having the shape of a spine). Figure 1.2 depicts two coupled neurons. Neuron B receives input activation from neuron A, from which axon terminal fibres make synaptical contacts on B's dendrites. In a synapse signals from the presynaptic membrane located on the spine are transmitted via the synaptic gap to the postsynaptic membrane located on the dendrite. When the total input activation to a neuron exceeds a certain threshold the neuron fires, i.e. the membrane of the neuron *depolarizes*. Synaptic contacts may be either *excitatory* or *inhibitory*, meaning that their activity either facilitates or inhibits neuron depolarization. Whenever a neuron fires, it transmits activation through its axon to any subsequent neuron, possibly leading to new depolarizations.

The minimum time required for depolarization is about 2 ms. Hence, the output signal rate of a neuron is limited to 500 depolarizations per second (500 Hz). Since the amplitude of a depolarization is constant, the signal strength is encoded in the depolarization rate of a neuron. A high output rate implies a strong output signal (Adrian, 1946).

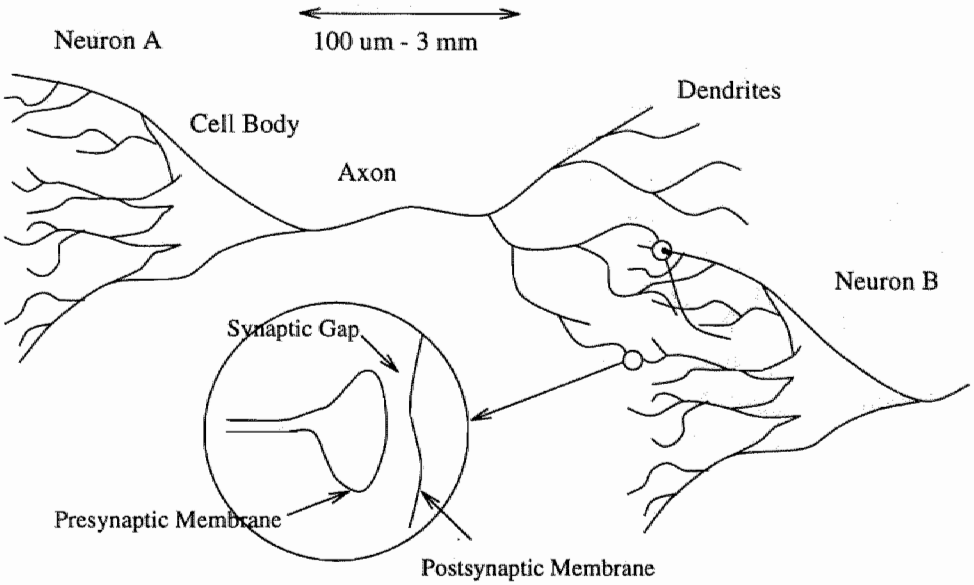


Figure 1.2: *Two biological neurons with dendrites, axon, synapse and cell body.*

1.1.1 Pattern Crunching

When comparing the brain to a parallel computer, the neuron cell bodies and their dendrites can be interpreted as processors and the axons can be interpreted as communication links. Owing to the connectionist nature of neural networks (Feldman, 1981), the behaviour of a single neuron cannot be interpreted without knowing the behaviour of the entire network. Unfortunately, this makes it impossible to compare the internal operation of the brain with the internal operation of a parallel computer or any other neural network unless they have the same structure. However, it is possible to compare their external operation, i.e. to compare the kind of input-output (I/O) representation used in the brain with that used in a parallel computer.

In computer programs, I/O is represented either numerically or symbolically. Special hardware and programming languages have been developed

suited for so-called *number crunching* (e.g. Fortran) or *symbol crunching* (e.g. Lisp). We can say that the brain is suited for *pattern crunching*, i.e. the brain processes patterns. A pattern is a structured data vector, for instance, an image projected on the retina. Unlike components of a numerical or symbolical representation the components of a pattern are not abstract. The pattern structure relates the individual components in such a way that a meaning emerges rather than that it is being referred to. For pattern-crunching tasks the performance of the brain lies far beyond the power of current supercomputers. For instance, we can recognize a face in a split second while this may take several minutes on a digital computer. For such applications, this would seem to contradict the observation that digital computers transmit signals through their circuits about a million times faster than the signal transmission in the neuron circuitry in the brain. The response time of a neuron is a few milliseconds ($10^{-3}s$) while modern computers operate in the nanosecond ($10^{-9}s$) range. Moreover, considering the fact that brain-response times take some hundreds of milliseconds (Posner, 1978) this means that *pattern-recognition tasks are carried out in less than a hundred time steps* (Feldman and Ballard, 1982). Typically, such tasks require millions of time steps when programmed on a sequential computer. Since it is not likely that the number of steps can be drastically decreased, it is believed that the brain processes information in parallel.

Apparently, it is difficult to design a parallel symbol or number cruncher that crunches patterns analogously to the brain. This difficulty may be explained by the fact that symbols and numbers are knowledge *atoms* (abstractions) which are indivisible and hence are hard to distribute. In contrast, a pattern can be partitioned in a natural way. Thus, the visual system of the brain employs *spatial parallelism* for processing patterns (Ullman, 1986). Not only visual information can be represented as spatial patterns. It is believed that cortical areas in particular form two-dimensional representations of a large variety of input signals (Knudsen *et al.*, 1987; Durbin and Mitchison, 1990). The spatial parallelism employed by the brain suggests an efficient mapping between the pattern structure and the parallel computing structure. Hence, by examining the organization of representation in the cortex we expect to find useful methods for parallel processing that are based on spatial parallelism.

1.1.2 The Neocortex

In this section we will describe the neocortex, which plays an important role in Chapter 3. The cortex is made up of the outer layers of the cerebrum, i.e. the encapsulating structure depicted in Figure 1.1. Frequently the cortex is also called neocortex indicating that it is the youngest part of the human brain in the course of evolution. Its extensive development is characteristic of the mammalian brain, especially in the behaviourally more interesting primates (Goldman-Rakic, 1988). The cortex is responsible for the sensation and perception of tactile, visual, auditory and olfactory stimuli as well as the integration of these experiences with motor control (Kolb and Whishaw, 1990; Luria, 1973; Penfield and Jasper, 1954).

The human neocortex is a wrinkled sheet of neuronal matter with an area of up to 2500 cm^2 and a thickness of 1.5 to 3 mm (Kolb and Whishaw, 1990). The cortex is divided into a right and left hemisphere. This bisection plays an important role in the functional behaviour of animals and humans (see, e.g. Gazzaniga (1985)). However, we are only concerned with the primary cortical zones in which sensor and motor signals are coded. The primary cortical zones in the left and right hemispheres are nearly identical (Luria, 1973) and we have restricted ourselves to the primary cortical zones in a single hemisphere.

The neuronal matter in the cortex can be divided into *grey matter* and *white matter*. The grey matter contains the actual cell bodies of neurons, while the white matter contains a large number of axons. The connections within the grey matter are called *intra-cortical* connections, as they are localized inside the cortex. The axons in the white matter belong to one of four categories: (1) input fibres, e.g. in the sensory cortex, (2) output fibres, e.g. in the motor cortex, (3) ipsilateral or cortico-cortical fibres connecting cortical neurons in the same hemisphere, and (4) contralateral connections connecting the left and right hemispheres. The cortico-cortical fibres may be differentiated into relatively short connections between the ridges of the wrinkles (cf. Figure 1.1) and the longer ones connecting the front, back, upper and lateral parts of the cortex. Naturally, the intra-cortical connections also connect different parts of the cortex but, being inside the grey matter, they are not longer than $3mm$ (Scheibel and Scheibel, 1970).

The neocortex has two interesting organizational features: (1) the *laminar* structure and (2) the topographical organization of *brain maps* in the primary cortical zone.

Laminar Structure

The laminar structure of the cortex can be viewed as an organization of different modules, i.e. layers, each with a specific task of its own, as will be explained below. It is interesting to observe that input to the cortex is projected on a specific layer. The activity in this layer is projected onto other layers maintaining contact with other cortical regions. In this way each layer fulfils a particular task in distributing the localized input.

The grey matter of the cortex is organized in six layers numbered I through VI (Martin, 1985) (cf. Figure 1.3). Layer I is the top layer; layer VI is the bottom layer, situated next to the white matter. The neurons in the cortex can be categorized into two major cell types (Palm, 1982). Firstly, in layers II, III and V there exist increasingly large, excitatory *pyramidal cells* having a constant shape. Secondly, there are much smaller *stellate cells*, which have either exciting or inhibiting connections with pyramidal cells. The stellate cells are interneurons and can be found in all layers. Their actual shape may vary considerably. The six layers are described below.

Layer I: In layer I there are no cell bodies. It contains apical dendrites and cortico-cortical fibres from other areas. An apical dendrite is the top part of a pyramidal cell body found in layers II, III and V.

Layers II and III: Layers II and III contain small pyramid-shaped and star-shaped cells. They have dendritic trees that receive local synaptic input as well as input from the upper layers. Their axon collateral traverses the cortex radially to layer IV and further. In some cases the axon leaves the grey matter, heading for other cortical areas. The cortico-cortical connections form associative links between cortical areas and hence layers II and III are also referred to as *associative layers*.

Layer IV: The neurons in this layer receive input from the thalamus. Since the thalamus is thought to be a relay station for sensory infor-

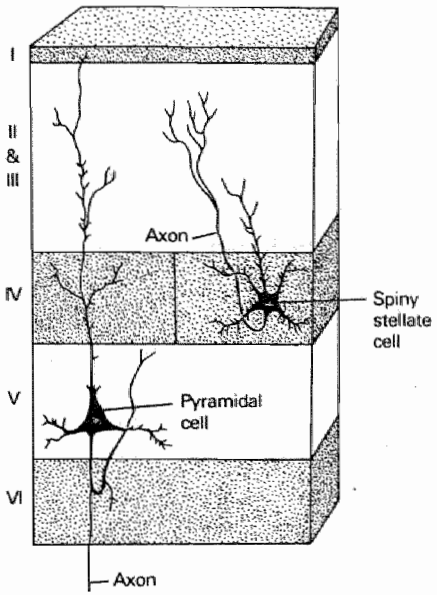


Figure 1.3: *Schematic drawing of the laminar nature of the cerebral cortex. (from Kandel and Schwartz, 1985). Cortical neurons are generally classified as pyramidal or stellate.*

mation, layer IV may be thought of as composing a zone of sensory analysis, i.e. the *input layer* of the cortex. The thalamic fibres are connected to spiny stellate cells, which excite layers above as well as below layer IV. Layer IV is well developed in the primary sensory cortex.

Layer V: Layer V contains large pyramidal cells having dendritic trees that go back to layer I. Their axon collateral has a local arborization allowing feedback and a main axon-collateral that may leave the cortex and project onto other parts of the brain. Hence, layer V can be conceived as composing a zone of output from the cortex, i.e. the *output layer*. Besides the main axon, there are recurrent branches turning back towards higher layers.

Layer VI: Layer VI contains fusi- or multi-form cell bodies that receive

input from cells located in other layers. They have axons that go back to layer I.

The layered differentiation is an organization which is orthogonal to the surface of the cortex. The organization of different sections of the cortex is not always identical. For instance, layer IV in the sensory cortex differs considerably from layer IV in the motor cortex (on the right) (Kolb and Whishaw, 1990) (cf. Figure 1.4). On the one hand, layer IV in the sensory cortex is comparatively large since it receives many (visual) sensory inputs. Layer V, on the other hand, is relatively thin. Just the opposite is true for the motor cortex where layer IV is small and layer V is much larger than in the sensory cortex due to the large number of outputs necessary for motor control.

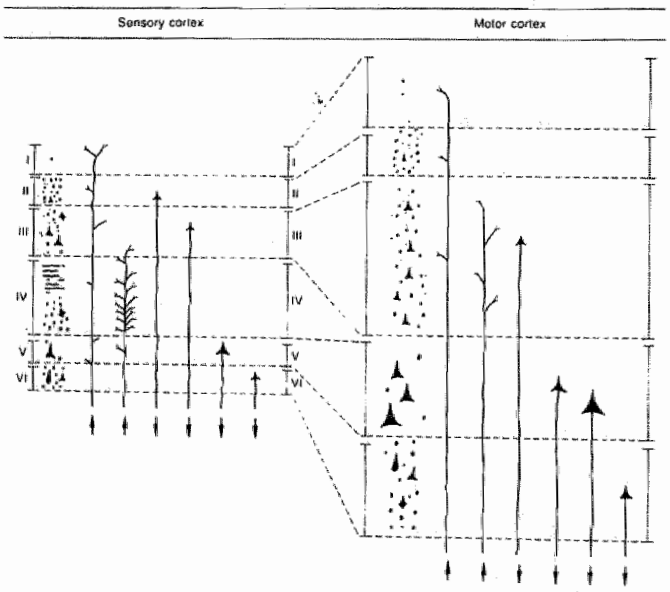


Figure 1.4: *Schematic illustration of the neuronal elements of the sensory and motor cortex (from Kolb and Whishaw (1990)).*

Brain Maps

The laminar structure can be seen as an intricate relay structure for connecting input to and obtaining output from the cortex as well as maintaining intra-cortical and cortico-cortical connections. A large part of the cortex is neither directly involved in processing primary sensory input nor in primary motor control. However, through the numerous cortico-cortical and intra-cortical connections these parts are indirectly involved in integrating the primary sensory experiences and primary motor control. Areas in the cortex are called *brain maps* since they correspond to surface parts. In order to understand how brain maps operate, it is helpful to know how different maps in the cortex may be identified. Globally, there are two methods for labeling parts of the cortex: (1) cytoarchitectonic maps and (2) projection maps.

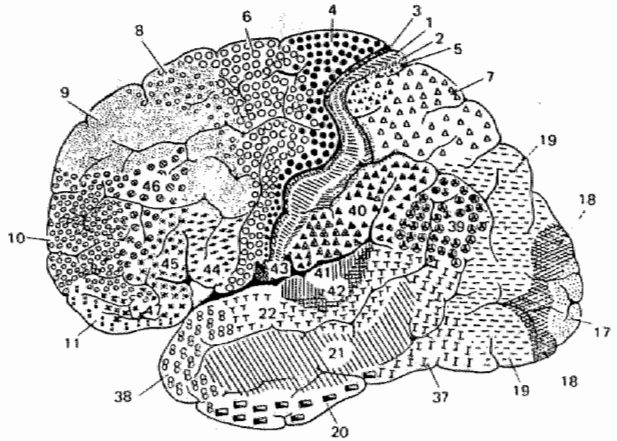


Figure 1.5: *Map of Brodmann areas (Brodmann 1909). Lateral view showing that each area has a different texture.*

Cytoarchitectonic maps: One of the earliest studies on the structure of the neocortex has been performed by Brodmann (1909), who made a chart of the cortex indicating a large number of different areas. This map is called *cytoarchitectonic* because the areas on the map are distinguished by looking at the cell structure of neurons. More recently, new methods have become available enhancing the subdivision of the cortex into functionally different sections (Kaas, 1987).

From the varying thickness of the layers (cf. Figure 1.4) we know that the functionality of cortical areas is to some extent expressed by their architecture (cf. Figure 1.4) and hence the Brodmann areas also represent functionally different areas of the cortex. Figure 1.5 shows the familiar map of Brodmann's areas of the human cerebral cortex.

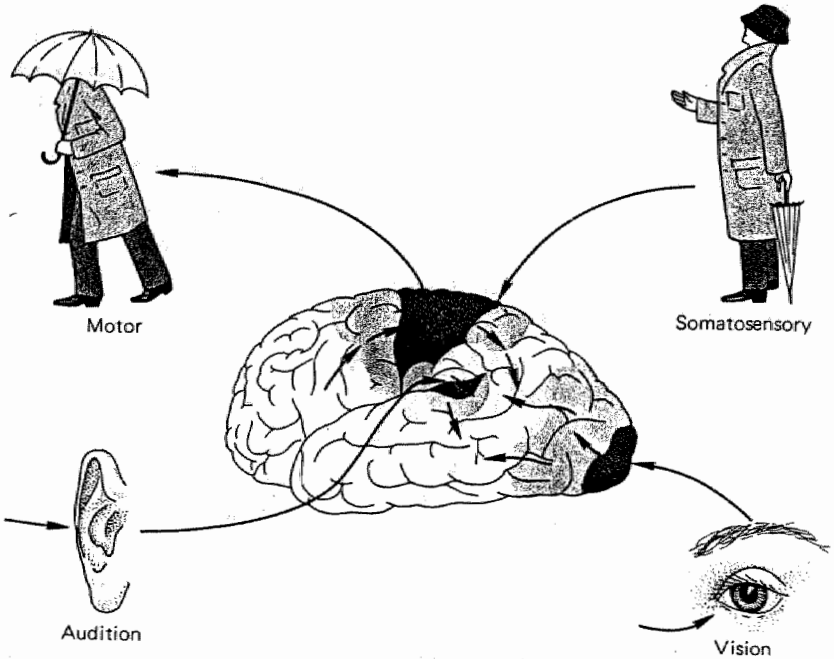


Figure 1.6: *A projection map. The dark areas indicate primary zones, the grey areas are secondary zones and the unshaded areas are tertiary, or association, zones (Kolb and Whishaw (1990)).*

Projection maps: Another way to study the structure of the neocortex is by means of a *projection* map (Kolb and Whishaw, 1990). A projection map is constructed by tracing axons from the sensory systems into the brain, and tracing axons from the neocortex to the motor systems. Such a projection map is depicted in Figure 1.6. It shows that there are three main types of area found in the neocortex: (1) areas in the primary zones of the cortex receiving input from the sensory systems or projecting output to the motor systems, (2) projection

areas in the secondary zones, and (3) projection or association areas in the tertiary zones (cf. Figure 1.6).

In view of Brodmann's map and the projection map of the cortex, cortical areas are believed to represent specific functions. However, it remains unclear how perception and action emerge from the interactions between these areas in the secondary and particularly in the tertiary cortical zone. Mountcastle (1978) believes that areas of the cortex resemble structured sets of neurons projecting their output onto other areas without losing the topological structure. Input and output regions, especially, represent accurate topographical projections of their sensory input and muscle output devices respectively. A topographical projection implies that the spatial proximity of somato-sensory receptors, e.g. a pressure receptor, on our body is reflected by their projection pattern on the cortical surface. Well-known examples of such mappings are the *retinotopic* maps in the visual cortex, *cochleotopic* maps in the audio cortex and *somatotopic* maps that represent input from sensory receptors (Penfield and Jasper, 1954).

Multiple maps may be used for representing different dimensions of the same input signals. For instance, it is known that in the auditory cortex of the mustached bat multiple cochleotopic maps exist representing frequencies, echo delay and doppler effects (Suga, 1990). Because of the large number of cortico-cortical connections, the representations in the secondary and tertiary zones are much more diffuse than the topographical representations in the primary zones. Therefore little is known about the exact way in which information from the primary areas is integrated in the other areas. It is generally believed that the association cortices mediate a variety of cognitive functions (Goldman-Rakic, 1988).

Since the operation of the maps in the secondary and tertiary cortical zones is not clear, we have restricted our research to models of maps in the primary zone. The essential function of the brain maps in this zone is to perform computations. Hence, they are called *computational maps* (Knudsen *et al.*, 1987). In this context "computation" is defined as any transformation in the representation of information. For instance, in the visual cortex a map exists in which neurons, or columns of neurons, respond to a specific orientation in their receptive field (on the retinal image). The neurons are so located that neighbours respond to slightly different orien-

tations. Hence, there is a continuous mapping on the cortex of increasing values of the orientation parameter (Hubel and Wiesel, 1963). Compliant with advances in image processing the transformations in the primary visual cortex may represent visual codes resembling wavelet ensembles (Daugman, 1989; Daugman, 1990).

Computational maps can be described as dimension-reducing mappings from a many-dimensional parameter space to the surface of the cortex. The goal of these mappings is to preserve, as much as possible, neighbourhood relations in the parameter space so that computations requiring localized data in parameter space can be performed locally in the cortex (Durbin and Mitchison, 1990). In a somewhat broader context the role of computational maps may be interpreted as a parametrization scheme. Such schemes are used, for instance, as reduced and comprehensive representations of sensor data (Braspenning, 1986).

1.1.3 Signal Transmission

Signal transmission in most neurons in the brain is based on chemical reactions. In this section we describe the different stages of signal transmission (Kandel and Schwartz, 1985; Kolb and Whishaw, 1990) summarized in Figure 1.7(a). Initially, the output signal of a depolarized neuron is transported as a *spike-shaped* signal (cf. Figure 1.7(b)) through the main axon to the synapses on dendritic trees of other neurons. The spike-shaped signal is a potential arising from a difference in concentration of positive and negative ions on opposite sides of the axon cell membrane. The depolarization of the neuron at its cell body is a potential inversion over the cell membrane. This potential inversion initiates a chain reaction over the cell-membrane extensions in the axon resulting in the spike-shaped output signal. The chain reaction ensures that the spike signal is transported *actively* through the axon similarly to a burning fuse. Hence, the signal strength does not diminish over large distances, enabling non-local connection patterns.

At the end of the axon the spike signal arrives at the *presynaptic* membrane triggering a chemical reaction. This results in a bell-shaped postsynaptic signal transported through the dendrite to the cell membrane. Unlike the active signal transmission in axons, a PostSynaptic Potential (PSP) is a

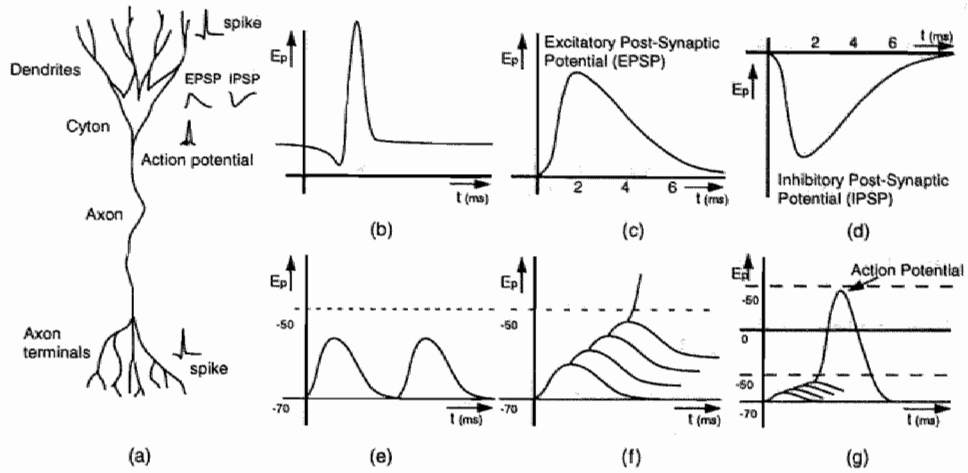


Figure 1.7: *Signal transmission in a neuron. (a) Signal trajectory from synapse to axon terminal passing through various stages. (b) Spike-shaped activation signal. (c) EPSP: exciting postsynaptic potential. (d) IPSP: inhibiting postsynaptic potential. (e) Integrated postsynaptic activity without depolarization. (f) Integrated postsynaptic activity with depolarization. (g) Activity level of a depolarizing neuron.*

graded potential, i.e. it becomes smaller with the distance to be travelled through a dendritic tree (as the size of a wave in water decreases with its distance from the source (Kolb and Whishaw, 1990; Klaassen, 1992)). If a PSP is too weak, it will not contribute to the cell-membrane potential. However, when two PSPs from neighbouring synapses arrive simultaneously they can both contribute to the total activity because their combined potential is large enough. This suggests that *timing* plays an important role in signal transmission between neurons. If the contact is of the excitatory type then the bell shape is positive and is called *Excitatory PostSynaptic Potential* (EPSP) (Figure 1.7(c)). If the contact is of the inhibitory type then the signal is negative and is called *Inhibitory PostSynaptic Potential* (IPSP) (Figure 1.7(d)).

Dendrites perform a spatial summation of input signals. This sum is integrated temporally by the cell membrane acting as a leaky condensator (Sejnowski, 1981; Kandel and Schwartz, 1985). Due to leakage the PSPs only have a temporary effect on the cell membrane. The duration τ of the effect is constant and ranges between 10 and 100 *ms* (Noest, 1988). If, on the one hand, the time interval between two subsequent signals is longer than τ there will be no temporal integration. This situation is depicted in Figure 1.7(e). On the other hand, if the time interval between two subsequent signals is shorter than τ seconds, the potential over the cell membrane will gradually increase (see Figure 1.7(f)).

The *rest potential* of the cell membrane from a neuron that is not activated equals -70 mV. EPSPs are responsible for increasing this potential, i.e. for making it more positive. When the -50 mV level is reached the potential is inverted to +50 mV. This is the actual *depolarization* process and is depicted in Figure 1.7(g). The IPSPs are responsible for decreasing the cell membrane potential, hence inhibiting the neuron. In that case the neuron is said to be in a *hyperpolarized* state. A hyperpolarized neuron is more difficult to depolarize than a neuron that is not activated.

The size (length and width) of the EPSP and IPSP determine at what input rate temporal integration may result in a depolarization. This size is influenced by transmitter substances TTX (tetrodotoxin) and TEA (tetraethylammonium) (Kandel and Schwartz (1985), pp. 121-122). An increased concentration of TTX enlarges the amplitude of the EPSP and IPSP while an increased concentration of TEA increases the duration of the signals. The concentration of transmitter substances increases when there is a large concentration of calcium ions (Ca^{++}). The control of transmitter substances is referred to as *presynaptic inhibition* and *presynaptic facilitation*, the decrease and increase of transmitter concentration respectively. It is known that enduring synaptic activity leads to an intake of Ca^{++}, which increases the TTX and TEA concentrations. The effect of synaptic inhibition and facilitation may continue for several hours or days.

The existence of such a control mechanism is important in that it enables synaptic modification, which is believed to play an important role in *learning* behaviour. The neurophysiologist D.O. Hebb was the first to formulate a hypothesis for adaptive behaviour of connections between neurons (Hebb,

1949). This hypothesis is known as Hebb's rule or Hebbian learning. Hebb's hypothesis reads as follows:

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" (p. 50)

Although this is not a mathematical statement, Hebb's rule is often interpreted as a correlation between neuron activities. This means that the conductivity of a synaptic connection is adapted with respect to the correlation of presynaptic activity and neuron output activity. It is known that transmitter concentrations are influenced by presynaptic activity. However, there is no evidence that the output activity of a neuron is involved as well.

1.1.4 Memory

According to Lashley (1950), as long as 300 years ago Descartes believed that information in the brain was somehow stored in specific locations. In the nineteenth and early twentieth century many neurophysiologists believed that memory was localized in the association areas next to the primary sensory areas in the cortex. The term *memory trace* or *engram* was used to refer to traces of memory in the brain. In a paper aptly entitled "In Search of the Engram", Lashley (1950) describes a number of experiments which have led him to believe that such a localization of memory is not possible. In one of his experiments Lashley trained rats to find their way through a maze. By subsequently removing sections of the cortex he was able to study their ability to remember the way through the maze. This experiment showed that destroying small parts of the cortex did not influence the learned habits as long as the primary sensory cortices are not disturbed. However, destroying a large amount of cortex, e.g. an entire lobe, produced a serious loss of habit. This led Lashley to formulate a theory of mass action or mass facilitation stating that the neuronal reaction mechanism, e.g. memory recall, is a definite pattern of integrated neurons. The activation of these patterns may come from a variety of sources. Lashley concluded

that: "Recall involves the synergic action or some sort of resonance among a very large number of neurons. The learning process must consist of the attunement of the elements of a complex system in such a way that a particular combination or pattern of cells responds more rapidly than before the experience.". Hence, memory cannot be localized but has a *distributed representation* according to Lashley.

The experiments summarized in Lashley (1950) are based on the observation of behaviour trying to trace conditioned reflex paths through the brain or to find the locus of specific memory traces, i.e. localized representations in the brain. A radically different approach is to stimulate localized parts of the human brain and let the test person tell what he or she experiences. Many of such experiments were performed by Penfield and his colleagues (Penfield and Rasmussen, 1950; Penfield and Jasper, 1954) who operated on epileptic patients. Penfield discovered that activation of certain neurons in the brain of one of his patients resulted consistently in the experience of perceptual sensation and even in re-activation of certain memory traces. When other neurons were activated different experiences were recollected. Hence, the findings of Penfield suggest the existence of local memory traces not only in the primary, but also in the secondary and tertiary zones.

Encouraged by Penfield's results researchers took a renewed interest in the idea of localization in the brain. When new techniques became available, the existence of localized perception in the cerebral cortex was also confirmed. Research on the visual cortex has revealed a great deal of specificity (Hubel and Wiesel, 1959; Hubel and Wiesel, 1963). A hierarchy of neurons was found in which *complex* and *hypercomplex* neurons exist that respond to very specific shapes in the visual field. Additionally, Gross *et al.* (1972) discovered the existence of cells activated by specific input patterns, such as the shape of a hand. This discovery led to the so-called *neuron doctrine* (Barlow, 1972; Churchland, 1986). This doctrine suggests that cognition is based on *grandmother cells* which only fire when highly specific patterns are presented, e.g. the face of one's grandmother.

Thus it seems as if Lashley was wrong: that there is no such thing as mass action of neurons. However, Penfield experimented with experience whereas Lashley experimented with behaviour. Penfield's ability to evoke previous experiences by localized stimulation of the brain does not necessarily mean

that the brain operates on a localized representation. Moreover, in the sense of a computational map the observations of Penfield and Lashley match perfectly. For instance, consider a computational map in the cortex. On the one hand, Lashley may destroy parts of this map and observe no remarkable degeneration of behaviour as long as there is not too much damage. On the other hand, Penfield may stimulate a particular neuron and observe a detailed experience. Both observations are correct and need not conflict. Hence, the issue at stake is not whether information is represented either locally or in a distributed form. Instead, we should study how local and distributed representations interact. We know that some interdependence is produced by the large number of cortico-cortical connections that go forward and backward resulting in diffuse representations in the secondary en tertiary areas. By examining ANNs we aim to gain more insight into the interaction between localization and distribution which seems to play an important role in the parallel processing in the brain.

1.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are models of biological neural networks. ANNs assume that all information processing in the brain is performed by neurons and synaptic contacts. The signal transmission within a synapse is modelled as a weight coefficient and the dendritic summation as taking the weighted sum of the inputs in a neuron which is transformed into an output by a transfer function f. Moreover, in some models the temporal integration is taken into account.

In Section 1.2.1 we introduce ANNs by describing the Perceptron, one of the earliest and simplest neural-network models. In Section 1.2.2 the functional architecture of ANNs is explained and related to existing ANN models other than the Perceptron. Finally, in Section 1.2.3 attention is paid to three basic ANN entities, i.e. connections, neurons and activation.

1.2.1 The Perceptron

In Section 1.1 the chemical processes responsible for signal transmission in neurons have been described. In this section we present a simple model

for a biological neural network called the *Perceptron* (Rosenblatt, 1959; Rosenblatt, 1962; Block, 1962). The Perceptron consists of *linear threshold units*; a neuron is modelled as a unit that calculates a linear combination of its inputs and fires if this combination exceeds a certain threshold. The linear combination is a weighted sum, i.e. each input is multiplied by a coefficient and then added to the total input activation.

We first describe the Perceptron informally. The neurons are organized in a layer, each receiving input from, e.g. a retina (cf. Figure 1.8). A retina is an array of light receptors representing a black-and-white image such as the X depicted in Figure 1.8. The X image contains black and white spots (pixels) that are coded as binary numbers, i.e. using 0 for white and 1 for black.

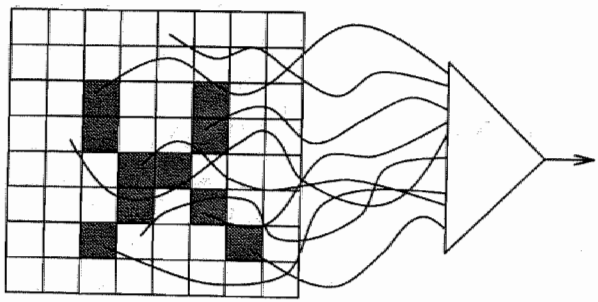


Figure 1.8: *Perceptron connected to a retina.*

The neuron depicted in Figure 1.8 receives activation from a number of receptors. As indicated above, each input in the neuron is multiplied with a coefficient. If the coefficient is large it means that the corresponding input has a considerable influence on the total sum of input activation. If, however, the coefficient is small, the input is apparently not of much importance to the neuron. Through this process of multiplication and summation a weighted sum of the input is calculated in the neuron. The coefficients are usually called *weights*. If the weighted sum exceeds a certain threshold, the neuron fires and its output is set to one. As long as the neuron does not fire, its output remains zero. A weight is a linear model for the transformation of a presynaptic signal into a postsynaptic signal. We note that

this model does not account for the temporal aspects of signal transmission in biological neurons. In Klaassen (1992) a temporal model of signal transmission through synapses is presented. In the Perceptron an input through a positive weight will increase the weighted sum, i.e. resemble an EPSP. An input through a negative weight will decrease the weighted sum, i.e. resemble an IPSP. Summarizing, we can compare this model to a lock that will open (fire) only with the correct key, i.e. a particular combination of input signals.

Next, we present a formal description of the Perceptron. Let σ denote the weighted input sum in a neuron and let θ denote its threshold. Then, the *processing model* of a neuron with output y, inputs $x_1, \ldots, x_n$, and connections $w_1, \ldots, w_n$ can be described as follows:

$$\sigma = \sum_{i=1}^{n} w_i x_i, \quad y = \begin{cases} 1 & \sigma \geq \theta \\ 0 & \sigma < \theta \end{cases} \tag{1.1}$$

The threshold of a neuron can be modelled as a weight which is connected to a so-called *bias* input. The bias input is a virtual input in the Perceptron that is constantly -1. If the connection weight from the bias neuron to the neuron is equal to θ then Equation 1.1 becomes:

$$\sigma = \sum_{i=1}^{n} w_i x_i - \theta, \quad y = \begin{cases} 1 & \sigma \geq 0 \\ 0 & \sigma < 0 \end{cases} \tag{1.2}$$

In terms of the Perceptron depicted in Figure 1.8 the task of the neurons can be defined as recognizing a pattern of black pixels. A particular pattern of black pixels may be detected by assigning positive values to the weights connected to the receptors representing black pixels, and negative values to the other weights. If a proper threshold is chosen it is not difficult to understand that this neuron will only respond to that particular pattern and to no other pattern on the retina. Such a setting of weights and thresholds is called a Perceptron *configuration*. It is not always easy to determine a proper configuration for a given task. Therefore it is desirable to find an adaptation rule so that the network can be adapted automatically in order to *learn* a certain task.

We recall the Hebbian learning rule introduced in Section 1.1.3. Hebbian learning is interpreted as a change in weight value (Δw) based on the correlation of input and output. In the case of the Perceptron this would result in the following adaptation rule:

$$\Delta w_i = x_i y \tag{1.3}$$

According to this rule the change of the i-th weight is equal to the i-th input times the neuron output that results from *all* current input. Such a correlation learning rule has interesting applications, for instance in an auto-associative ANN memory (Kohonen, 1982). However, when this rule is applied there is no possibility to supervise the learning process. Hence, the Perceptron uses the so-called *delta rule*, resembling the Hebbian learning rule described in Equation 1.3. The difference is that the neuron output y in the Hebb rule is replaced by the neuron error δ, i.e. the desired neuron output Y minus the actual output y. For instance, if the desired output exceeds the actual output, the weight should be larger; there should be a positive weight change if the input is positive and a negative weight change if the input is negative. Moreover, if the actual output equals the desired output the weight change is zero and no learning occurs. Hence, using the delta rule a Perceptron can be adapted in a supervised way by providing the desired outputs. The delta rule can be described as follows:

$$\Delta w_i = (Y - y)x_i = \delta x_i \tag{1.4}$$

When a Perceptron is configured to perform a certain task, it has formed an input-output *mapping* determined by the weights and the thresholds of the neurons. A mapping is a transformation of input values into output values. For instance, in Table 1.1 three mappings are presented for the logical OR function, AND function and the eXclusive-OR (XOR) function, respectively. The XOR function is a logical expression of two logical variables resulting in a true, if one but not both input variables are true. A true value is coded as 1 and a false value as 0. The Perceptron can be adapted for realizing the AND-mapping or the OR-mapping. However, it appears that the Perceptron cannot learn the XOR-mapping (Minsky and Papert, 1969).

Input		Output		
x_1	x_2	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Table 1.1: *Mappings of the OR, AND and XOR function.*

The problem encountered in the XOR case can be explained as follows. A single neuron can only categorize inputs x_1 and x_2 in two classes, viz. class 0 containing inputs for which $w_1x_1 + w_2x_2 < \theta$ and class 1 containing inputs for which $w_1x_1 + w_2x_2 \geq \theta$. For any value of w_1, w_2 and θ the boundary between class 0 and class 1 is line-shaped meaning that the inputs corresponding to classes 0 and 1 are *linearly separable*. So, the AND and OR functions are linearly separable but the XOR function requires a non-linear separation (cf. Figure 1.9).

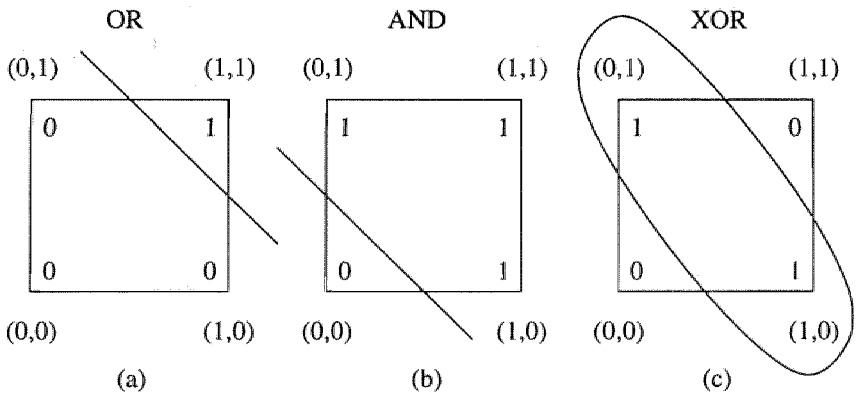


Figure 1.9: *Geometric representation of the OR, AND and XOR function.*

It is theoretically shown by Minsky and Papert (1969) that a Perceptron is not capable of solving non-linearly separable problems, such as the XOR-

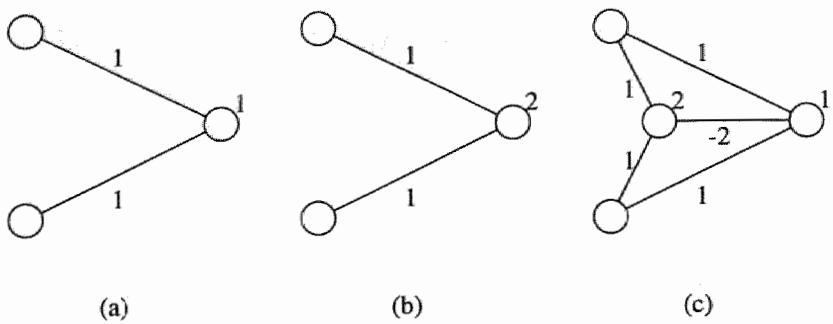


Figure 1.10: *Perceptron configurations for (a) OR function, (b) the AND function, and (c) a network for the XOR function using an intermediate neuron.*

problem. If we were allowed to add another input feature x_3 to the Perceptron, which is the AND function of x_1 and x_2, it would be possible to solve the XOR problem. This input feature could be calculated by a second, intermediate neuron. In Figure 1.10(a) and (b) the Perceptron configurations for the AND and the OR function are drawn. Both networks consist of two links that feed into the same neuron. In Figure 1.10(c) a network that solves the XOR function with an intermediate neuron is depicted.

Unfortunately, an intermediate neuron cannot be configured using the delta rule because the examples do not specify its target output value: there is no value for x_3 specified in Table 1.1. Hence, the Perceptron is restricted to a limited range of applications as long as there is no learning rule for networks with intermediate neurons. In 1985 and 1986 the problem was tackled and a generalization of the delta rule was introduced that could be used for adapting intermediate connections (Parker, 1985; Le Cun, 1985; Rumelhart *et al.*, 1986). The application of the *generalized delta rule* requires two phases. In the first phase input is propagated forward to the output units where the delta error is measured. In the second phase the error is propagated backward through the network and is used for adapting connection weights. Due to the second phase this procedure is also known as *Back Propagation* (Rumelhart *et al.*, 1986). A generalization of the delta rule to networks with intermediate neurons having complex-valued outputs

and weights has been developed by Henseler and Braspenning (1990a) and will be discussed in Chapter 2.

1.2.2 Functional Architecture

The *functional architecture* of an ANN can be decomposed in a *connection architecture* and a *neuron architecture*. The connection architecture describes the structure of interconnections between neurons and their characteristics, e.g. the weight values. The neuron architecture describes the processing model of the neuron, i.e. the way a neuron transforms its input into an output value. For instance, the processing model of the neurons in the Perceptron corresponds to the linear threshold model (cf. Equation 1.1). Moreover, the neurons in the Perceptron only receive input and are not interconnected. Hence, the Perceptron connection architecture only describes the input connections. The connection architecture of the XOR network depicted in Figure 1.10(c) also specifies how the intermediate neuron is connected to the other neurons. Once the functional architecture of a network is known, its operation is unambiguously specified.

Connection Architecture

In general, the connection architecture of an ANN is to be interpreted as a *weighted graph*. The neurons are compared to the *vertices* of a graph while the connections between neurons are compared to the *edges*. A simple weighted graph is represented by its *weight matrix* $\mathbf{W}$, with elements w_{ij} being the weight of the edge from vertex j to i. The weights of non-existent edges are usually set to ∞ or 0, depending on the application. For instance, in a neural network we will use 0, so that no input will be delivered. However, if the weights represent graph distances a missing connection will be represented by ∞, indicating that there is no way through.

The connection structure of an ANN is determined by the connections between neurons. Hence, this structure is described by the shape of the weight matrix $\mathbf{W}$ of the network, i.e. the positions of the non-zero weights w_{ij}. To illustrate the meaning of a connection architecture we discuss four different types : fully-interconnected networks, multi-layer networks,

cellular networks and winner-takes-all networks.

Fully-Interconnected Neural Network (FINN): This type of neural network is fully-interconnected, i.e. a bidirectional connection exists between every pair of neurons. Hence, the weight matrix contains no zero weights. A typical FINN is depicted in Figure 1.11(a).

Multi-Layer Neural Network (MLNN): Multi-layer neural networks can be imagined as a stack of neuron *layers* with connections between subsequent layers only (Rumelhart *et al.*, 1986) (cf. Figure 1.11(b)). MLNNs do *not* have recurrent connections, i.e. there are no loops in the network. A loop is essentially a cycle in the graph structure. Such a cycle results from a sequence of connections leading back to the neuron from which the sequence originally started. On account of the layered structure the weight matrix contains mostly zero weights except for a number of rectangular-shaped areas around the main diagonal. The weights in a rectangular area correspond to the weight matrix of the connections between two subsequent layers.

Cellular Neural Network (CNN): The neurons in a cellular neural network (Chua and Yang, 1988a) are arranged as points on a two-dimensional rectangular grid. The links between neurons are the actual grid lines. Hence, each neuron is connected to only four other neurons resulting in a highly-localized connection architecture shown in Figure 1.11(c). Each neuron in a CNN is identical and hence the sparse weight matrix has a regular structure.

Winner-Take-All Neural Network (WTANN): Finally, we mention Winner-Take-All neural networks (Feldman and Ballard, 1982) which have the property that only the neuron with the highest activity will remain active after some period of time suppressing all other neurons. For instance, the on-center off-surround neural network introduced by Grossberg (1973) is a WTANN. This network is fully interconnected having two different connection types, as neuron inputs from different connection types are processed separately. Firstly, every neuron has a fixed inhibitory connection with every other neuron. Secondly, every neuron has a fixed single excitatory connection with itself, i.e. a feedback connection. We saw that in the neuron-processing model of

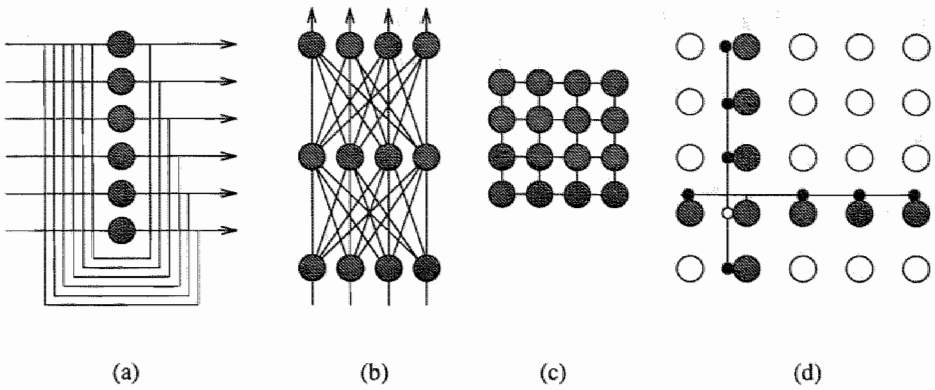


Figure 1.11: *Examples of connection architectures. Circles denote neurons and lines denote connections. (a) FINN. (b) MLNN. (c) CNN. (d) connections for a single neuron in the HTNN (the small black and white circles correspond to inhibitory and excitatory connections respectively).*

the Perceptron (cf. Equation 1.1) it was possible to model inhibitory connections with negative weights and excitatory connections with positive weights. However, in Grossberg's WTANN this is not possible since the neuron-processing model processes both types of input in a different way (see below). A specific case of this WTANN is the connection architecture for the Hopfield-Tank neural network (HTNN) (Hopfield and Tank, 1985). The neurons in the HTNN are arranged in rows and columns, similar to the arrangement of neurons in a CNN. The HTNN is not fully interconnected. Unlike a neuron in the WTANN, a neuron is only inhibited by neurons in the same row and column rather than by all neurons. Hence, instead of one winner there is a winner in each row and column. Moreover, in the HTNN, neurons are not self-connected as the excitatory connection with itself has been replaced by an internal feedback loop. The connection architecture of the HTNN is shown in Figure 1.11(d). The small black and white circles on the neuron denote inhibitory and excitatory connections, respectively.

We mentioned above that a connection architecture can contain more than one connection type. These artificial connection types differ from the synapse types encountered in biological networks. In ANNs the connection type is determined by the neuron architecture, i.e. the neuron-processing model. For instance, neurons in a CNN distinguish between input from the environment and input from neighbouring neurons in their processing model. Neurons in a WTANN process inhibitory and excitatory inputs differently. Some ANNs only have one connection type. For instance, in a MLNN excitatory and inhibitory connections as well as input connections are processed identically. In order to describe the connection structure as a weighted graph, we must introduce a weight matrix for each connection type. If necessary, the architecture of an ANN is represented by an *ensemble of weight matrices.*

Neuron Architecture

The neuron architecture of an ANN specifies the neuron-processing model. From the description of signal processing in Section 1.1.3 it appears that the architecture of neurons can be divided into three parts. The first part resembles the dendrites of a biological neuron. It models the synaptical processing of inputs as well as their spatial summation. The second part resembles the neuron cell membrane. It integrates temporally the result of the first part. The third part is a model for the transformation of the internal neuron state into an output value. These distinctions result in a three-stage neuron-processing model (Henseler and Braspenning, 1991). A block diagram of the three-stage processing model for the neuron architecture is depicted in Figure 1.12.

Consider a neural network with N neurons that receives M inputs coded as an input vector $\xi = \xi_1, \ldots, \xi_m$. Let $\mathbf{y} = y_1, \ldots, y_N$ denote the *activity vector* of the network. The processing model of neuron $i \in 1, \ldots, N$ depicted in Figure 1.12 consists of the following three stages :

Stage I: In the first stage activity y_j from other neurons $j \in 1, \ldots, N$ in the network enters neuron i through connections with weight w_{ij}. Each input is transformed by its corresponding weight resulting in a *weighted input.* This transformation is often a multiplication. The

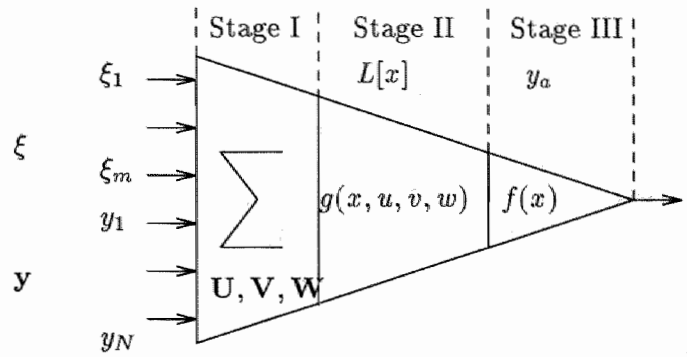


Figure 1.12: *Three-stage neuron-processing model.*

weighted inputs are summed and serve as input for stage II. Various summation forms are possible and multiple connections of different types between neurons may occur. For instance, if three connection types occur, the ensemble of weight matrices can be denoted as **U**, **V** and **W** and their related sums as u, v and w, respectively. Sums u, v and w are calculated by a summation function $\sum()$.

$$(u, v, w) = \sum(\xi, \mathbf{y}, \mathbf{U}, \mathbf{V}, \mathbf{W})$$

Stage II: In the second stage the input sums u, v and w are processed by a differential equation in the neuron internal state x. Let $L[x]$ denote a differential operator.

$$L[x] = g(x, u, v, w)$$

Stage III: The final processing stage transforms the internal neuron state x into a neuron output value by calculating the *neuron-output function* (f) value for x.

$$y_i = f(x)$$

Existing Artificial Neural Networks

The Perceptron is one of the earliest ANN models. During the past two decades a large number of other functional architectures have been developed. This development has resulted in complex models of biological neural networks but also in models with a simple mathematical foundation. The functional architectures of seven such ANNs are presented in Table 1.2.

The first example is the model for Auto-Correlation Matrix Memory (ACMM) (Kohonen, 1977). This network has a fully-interconnected connection architecture and the neuron-processing model is linear. The input to a neuron consists of one external input ξ_i and the weighted sum of all neuron outputs in the network. The neuron output is equal to the total input sum and hence there is no processing in stages II and III. It can be shown that in the case of Hebbian learning, the weight matrix converges to the auto-correlation matrix of the input distribution (Kohonen, 1977). Moreover, if negative Hebbian learning is applied, viz. by introducing a minus sign in Equation 1.3, the system acts as a *Novelty Filter* that will enhance novel components in the input.

The second example is the functional architecture used for Multi-Layer Neural Networks (MLNNs) that are adapted using the Back-Propagation algorithm (Rumelhart *et al.*, 1986). Usually these networks are represented as a layered connection architecture although the underlying principles apply to any connection architecture provided there are no recurrent connections. From the description of the first stage it follows that neurons either receive external input or receive input from other neurons. Analogous to the first example the second processing stage is void. However, in the third stage a non-linear function transforms the result of stage one into the neuron output. This non-linear function is often defined as a sigmoid function. The MLNN will be described in detail in Chapter 2.

The third example is a processing model used in the Brain-State-in-a-Box (BSB) model (Anderson and Mozer, 1981). Analogous to the first example this network has a fully-interconnected connection architecture. In stage I the input from other neurons is weighted. In contrast to the previous examples, input is presented to the network by setting the neuron states to an initial value. This input method is possible since the second stage

Functional Architecture Examples			
Connection-Architecture Type	Neuron Architecture		
	Stage I (u, v, w)	Stage II (x)	Stage III (y_i)
1. FINN	$u = \sum_{j=1}^{N} w_{ij} y_j + \xi_j$	$x = u$	x
2. MLNN	$u = \sum_{j=1}^{N} w_{ij} y_j$, $or u = \sum_{j=1}^{m} w'_{ij} \xi_j$	$x = u$	$\frac{1}{1+e^{-x}}$
3. FINN	$u = \sum_{j=1}^{N} w_{ij} y_j$, $\mathbf{y}(0) = \xi$	$\dot{x} = u$	$\begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & otherwise \end{cases}$
4. FINN	$u = \sum_{j=1}^{m} w_{ij} \xi_j, v(t) = \sum_{j=1}^{N} v_{ij} y_j$	$\lim_{\Delta t \downarrow 0} x(t) = u + v(t - \Delta t)$	$\begin{cases} 0 & x(t) < 0 \\ x(t) & 0 \le x(t) \le 1 \\ 1 & x(t) > 1 \end{cases}$
5. CNN	$u = \sum_{j \in \mathcal{N}(i)} w_{ij} y_j$, $v = \sum_{j \in \mathcal{N}(i)} v_{ij} \xi_i$	$\dot{x} = -Ax + Bu + v$	$\frac{\lvert x+1 \rvert - \lvert x-1 \rvert}{2}$
6. WTANN	$u = y_i, v = \sum_{j=1, j \neq i}^{N} y_j$, $w = \xi_i$	$\dot{x} = -Ax + (B - x)u$ $-xv + w$	e.g., $\frac{x^2}{1+x^2}$
7. HTNN	$u = \sum_{j \in \mathcal{N}_{cr}(i)} y_j$, $v = \xi_i$	$\dot{x} = -Ax - u + v$	$\frac{1}{1+e^{-x}}$

Table 1.2: *Seven examples of existing processing models described in three stages. (1) Auto-Correlation Matrix Memory (Kohonen, 1977), (2) Multi-Layer Neural Network (Rumelhart et al., 1986), (3) Brain-State-in-a-Box (Anderson and Mozer, 1981), (4) Self-Organizing Feature Map, (Kohonen, 1982), (5) Cellular Neural Network (Chua and Yang, 1988a), (6) Grossberg Neural Network (Grossberg, 1973), and (7) Hopfield-Tank Neural Network (Hopfield and Tank, 1985).*

contains a first-order differential equation, i.e. the second stage performs a *temporal* integration. The network activity vector always remains inside a "box" because the neuron outputs are limited to the interval $[-1, 1]$ by the output function in the third stage.

The fourth example shows the processing model of neurons in the Self-Organizing Feature Map (SOFM) (Kohonen, 1982). Analogous to the neurons in the first example, the neurons in the SOFM are fully interconnected receiving both external input and input from other neurons. However, in this case there are two connection types. Firstly, the external input ξ is weighted with $\mathbf{W}$, containing variable weights. Secondly, the inter-neuron connections have fixed weights in such a way that interactions between neighbouring neurons are exciting and long-distance interactions are inhibiting. The distinction between these two connection types is made in the second stage, which contains a non-linear differential equation. The weighted sum of external input is assumed to remain constant while the neuron interaction is slightly delayed, resulting in a relaxation process. In the third stage a hard-limiting output function restricts the neuron output to the interval $[0, 1]$. When the external input-weight vectors of the neurons are organized, the SOFM may be used as a computational map. This model will be described in more detail in Chapter 3.

The fifth example is the Cellular Neural Network (Chua and Yang, 1988a)(CNN). The CNN connection architecture is illustrated in Figure 1.11(c). Essentially, each neuron in the CNN is connected only to neighbouring neurons denoted as a collection $\mathcal{N}$. The second processing stage resembles a leaky integrator and the output function f is essentially the same as in example 3. The CNN model will be described in detail in Chapter 4.

The sixth example is the on-center off-surround Neural Network (Grossberg, 1973) which is essentially a WTANN (Feldman and Ballard, 1982). The connection architecture of this network has been described in the fourth connection architecture example. What is particularly interesting is the introduction of explicit excitatory and inhibitory connections. We note that in the examples 1 to 5 all inter-neuron connections are of the same type. In the WTANN the excitatory and inhibitory connections between neurons are treated separately in a non-linear first-order differential equation in stage

II which can only be solved numerically. Grossberg (1973) analyzes this model and shows, among other things, for what kind of output functions the network becomes a WTA network.

The final example shows the functional architecture of the Hopfield-Tank Neural Network (HTNN) (Hopfield and Tank, 1985). Although the behaviour of this network resembles that of a winner-takes-all network it differs from Grossberg's (1973) WTANN described in the sixth example. A first difference is that neurons only receive inhibition from neurons in the same vertical column or horizontal row in the network (cf. Figure 1.11(d)). This neighbourhood is denoted $\mathcal{N}_{cr}$. A second difference is that there is no self-excitation, i.e. the activation which a neuron receives from itself. We note that in Grossberg's WTANN self-excitation can be realized by letting $B > 0$ so that By_i is added to the integration in the second stage.

1.2.3 Connections, Neurons and Activation

In Section 1.1 the representation of memory in the brain was discussed. The issue of representation in the brain is directly related to the interpretation of brain processing. Analogously, we are interested in the representation of information in ANNs. In contrast to neural networks in the brain, ANNs have a restricted functional architecture so that it is possible to isolate three entities on which the representation of information in ANNs is based. Firstly, the behaviour of an ANN, e.g. a Perceptron, is partially encoded in connection weights and thresholds. We call this the *connections* entity. Secondly, ANNs operate on patterns and hence the arrangement of neurons is essential for the structure of these patterns, i.e. the topology of pattern elements. For instance, an image pattern can be represented by a two-dimensional grid of pixels. We call this the *neurons* entity. Thirdly, the activity of an ANN represents a value for each pattern element. For instance, a pixel value may be black or white. We call this the *activation* entity. Below we present three examples that illustrate the possible role of connections, neurons and activation in the representation of information in ANNs.

Connections in the ACMM: Eigenvectors

The ACMM (Kohonen, 1977) model (cf. the first example in Table 1.2) has a fully-interconnected connection architecture and a linear neuron architecture. Its operation can be described as a multiplication of an input vector with the weight matrix, resulting in an output vector. It is well-known that such a multiplication will modify the components in the input coinciding with principal components of the weight matrix, i.e. the eigenvectors. This can be explained as *association* in which one vector is associated with another by matrix multiplication (Knapp and Anderson, 1984). We note that the meaning of a single weight cannot be isolated. The representation in the ACMM is therefore called *distributed.* The representation of the connection weights in such a network can be analyzed only by calculating the eigenvalues and corresponding eigenvectors of the weight matrix.

Neurons in the SOFM: Topology

Neurons in a two-dimensional SOFM (Kohonen, 1982) are arranged in a two-dimensional regular grid providing a level of representation that is essential to the SOFM. Analogous to a neuron in a computational map, a neuron in a SOFM is tuned to a specific input in such a way that neighbouring neurons respond to similar inputs. Hence, similarity is visualized as a distance on a map of features since neurons are placed on a two-dimensional grid. The map organization represents the topology of the input distribution.

Activation in the MLNN: Features

The neurons in the input and output layer of a MLNN correspond to features of the environment. Hence, the representation of the activity pattern that emerges in these layers forms an unambiguous representation. In contrast, the activity pattern of intermediate neurons is often left without interpretation as if the network is a black box. Still, this internal activity pattern may be analyzed to reveal its representation, e.g. by using cluster analysis (Rosenberg, 1987). However, individual units may remain unexplained similar to the distributed representation of information by ACMM connec-

tion weights. Furthermore, if the temporal extent of the activity pattern is involved (see, e.g. Pribram (1991) or the ANN model presented in Chapter 4), the representation may also have a distributed nature. The ANN should be considered a dynamic system with periodic attractors and strange attractors as *temporal representations* (Hopfield, 1982; Amit, 1989; Hirsch, 1989).

1.3 Problem Statements

In order to obtain a better understanding of the massively parallel operation of ANNs, we propose to study the organization of representation. The ANN-representation medium consists of three entities, viz. connections, neurons and activation. Hence, our main interest lies in the organization of representation in connections, neurons and activation. We address three problems, each reflecting a different research approach. Briefly stated, the problems are related to the organization of representation in (1) MLNNs with complex-valued weights, (2) a modular system containing multiple SOFMs, and (3) a network of harmonical oscillators. The ANN models that are studied each show typical behaviour by which correlations between inputs from the outside world are recorded and represented internally. We note that the models do not compete with each other and will not be compared. The scope of each model is limited to a few functions and all three should be considered very poor brain models by themselves. In a sense, they are complementary allowing us to study the internal organization from different viewpoints. We emphasize that each approach is taken from a designer's perspective, i.e. *exploring* different models and testing their applicability. A summary evaluation will be carried out, providing preliminary evidence for the suitability of the approaches. Below, a research question is formulated for each problem. Furthermore, we outline the research approach that will be followed for each problem.

1.3.1 Complex Values

The research question for the first problem statement is the following: do complex-valued weights in the connection architecture of MLNNs lead to

efficient input-output mappings analogously to the complex transformations known from mathematics and physics? It is known from physics and other disciplines that under certain conditions complex transformations can simplify a particular computation considerably. Examples can be found in signal processing where the Fourier Transform is a complex transformation from the time domain to the frequency domain (Oppenheim *et al.*, 1983).

MLNNs are a generalization of the Perceptron described in Section 1.2. To adapt the connection weights in a MLNN a gradient-descent technique is used minimizing the error of the network by back propagation of the delta error measured at the output neurons (Rumelhart *et al.*, 1986). The flexibility of such a neural network to learn a particular input-output mapping is determined by the connection architecture and a number of learning parameters. We introduce a complex-valued MLNN, i.e. a multi-layer connection architecture in which connection weights and neuron outputs are complex valued. The complex-valued MLNN will be evaluated by deriving a complex-valued generalized delta rule (Henseler and Braspenning, 1990a)[1] and subsequently comparing the complex-valued MLNN with the standard real-valued MLNN.

1.3.2 Computational Maps

The research question for the second problem is the following: how can multiple Self-Organizing Feature Maps (Kohonen, 1982; Kohonen, 1984) be combined? In principle, a single SOFM is capable of integrating information into one topological representation. However, when the dimension of the input becomes large, the SOFM mechanism may lead to poor representations. Hence, by partitioning the input in several independent SOFMs this problem may be avoided.

In Section 1.1.2 the concept underlying the computational maps in the brain has been described. Moreover, it has been pointed out that the structure of the cortex consists of three zones, i.e. the primary, secondary and tertiary zones. In Luria (1973) this system is called *the second functional unit of the*

[1] Others have published similar derivations (Clarke, 1990; Kim and Guest, 1990; Leung and Haykin, 1991; Benvenuto *et al.*, 1991) but the research presented in Henseler and Braspenning (1990a) was independently performed in the first half of 1989.

brain. Although Luria's theory of functional units is still under discussion (Kolb and Whishaw, 1990), it provides a useful basis for the interpretation of the integration of sensory experience by the primary, secondary and tertiary zones in the cortex. It is certain that the primary cortical zone is divided into a number of different areas. A single area may contain multiple computational maps. Knudsen *et al.* (1987) present a detailed investigation of the computational maps in the auditory cortex of the barn owl demonstrating the existence of a representation of auditory space. The aim of this system is to provide the barn owl with a representation of spatial relations in its environment based on auditory experiences. Knudsen reports that two primary maps are formed in which interaural delay and interaural intensity difference are represented respectively. Subsequently, the activity patterns of the primary maps are integrated in a secondary map in such a way that a spatial representation is formed. The resulting system is capable of locating a sound source in space.

In the previous section we have indicated that the SOFM (Kohonen, 1982; Kohonen, 1984) is a good ANN model of a computational map. However, it does not provide any facilities for combining multiple maps. In view of the primary cortical organization we expect that such a combination will yield similar systems for functional integration of information. Hence, our aim is to find a method for combining multiple SOFMs to construct a modular connectionist approach for the integration of input signals.

1.3.3 Oscillating Neural Network

The research question for the third problem is the following: what is the nature of representation in an ANN based on an overall oscillating activation pattern? In Section 1.1.4 the mass action theory of Lashley (1950) was explained briefly. In his summary Lashley uses a metaphor of waves on the surface of a liquid to emphasize the resonance he envisions between neurons as a basic mechanism for memory recall.

> "... the characteristics of the nervous network are such that, when it is subject to any pattern of excitation, it may develop a pattern of activity, reduplicated throughout an entire functional area by spread of excitations, much as the surface of a liquid

> develops an interference pattern of spreading waves when it is disturbed at several points. "

By comparing the neural network with the dynamic system of interacting water molecules Lashley introduces a particular type of temporal activity representation, namely a wave representation. The capacity of information storage in wave patterns is large as we know from the theory of holography (Young, 1977). This suggests that the existence of some holographic principle of information storage in the brain would explain how it is possible that man is capable of remembering, i.e. storing and recalling, the vast amount of information learned during a life time. Pribram (1991) proposes a unified theory for the brain and perception for which he uses the Schrödinger equation from quantum physics to model neuro-dendritic processes. In this approach, Pribram models neural networks in the brain as a medium in which waves of neural activity are propagated. Kolb and Whishaw (1990) also emphasize postsynaptic potentials spreading through the dendritic trees as a wave travels on a water surface. Inspired by this view and the numerous mathematically defined physical models that are readily available we decided to investigate quantum mechanical models which led in particular to the wave representation in a network of coupled oscillators. The organization of the cortex in cortical columns, and the oscillatory behaviour of such columns, is another justification for this decision. The activation pattern that emerges from such a network of oscillating neurons codes information as a superposition of waves. Detailed analysis must reveal the relation between the functional architecture of the network and the wave representation.

1.4 Outline of the Research

In this chapter we have introduced the basic principles of NNs and ANNs. In view of the high performance of the brain for processing sensory experiences, the connectionist principles underlying its operation are of interest to parallel processing. Hence, by studying the organization of representation in ANNs we expect to learn more about the way computations and data are integrally represented in a connectionist model. The connections, neurons and activation in ANNs each have their particular way of representing

both information and computation required for massively parallel processing. From these representations we have chosen to study the connectionism in (1) layered networks with complex-valued weights, (2) a modular system containing multiple computational maps, and (3) a network of coupled oscillators.

The main results of each study are described in chapters 2, 3 and 4, respectively. Each chapter starts with an introduction to the problem. Subsequently, a functional model is presented of the basic ANN model that will be explored. After its operation is explained, the modifications or additions, due to the exploration of the model, are presented and analysed. Finally, a case-study is presented to illustrate the operation and potential of the new model.

Chapter 2 describes the study of layered networks with complex-valued weights. Firstly, the Back Propagation algorithm (Rumelhart *et al.*, 1986) used for adapting MLNNs is modified for the adaptation of complex-valued weights. Once this modification is realized, simulations are performed in order to compare the complex MLNN with a standard MLNN. We expect that the complex-valued MLNN is more successful than a real-valued MLNN for typical complex-valued input-output mappings (Henseler and Braspenning, 1990b; Clarke, 1990; Kim and Guest, 1990; Leung and Haykin, 1991; Benvenuto *et al.*, 1991).

Chapter 3 introduces a method for combining multiple Self-Organizing Feature Maps (SOFMs) (Kohonen, 1982; Kohonen, 1984) based on the modular structure of the primary cortex in the brain. In order to be able to evaluate the SOFM performance we introduce several techniques to visualize the internal SOFM weight organization. By visually and statistically inspecting this organization it is possible to evaluate the performance of a two-dimensional SOFM.

Chapter 4 introduces a network of linearly-coupled oscillators that are placed on a two-dimensional grid. It is shown that for a particular class of connection architectures the network activity can be described as a superposition of wave patterns. Due to its linearity the system can be solved analytically and it is possible to show that in the case of periodic boundary connections the wave patterns are invariant under translation of the input pattern. The translation invariancy of the chosen representation is tested

by computer simulations using scanned images of various autographs.

Finally, in Chapter 5 the work presented in Chapters 2, 3 and 4 is evaluated and in view of the conclusions drawn we discuss the organization of representation in ANNs and evaluate the research results.

Chapter 2

Back Propagation with Complex-Valued Weights

The Multi-Layer Neural Network (MLNN) is currently the most popular ANN due to its simple training procedure, which is known as the Back-Propagation (BP) algorithm (Rumelhart *et al.*, 1986)[1]. Normally, the network activation and connection weights of a MLNN are real valued. However, it is known from mathematical physics and other disciplines that under certain conditions complex transformations can simplify a particular computation considerably. Is it possible that a complex-valued representation in MLNNs has similar benefits? To answer this question and to understand how this change of internal representation effects the MLNN and BP algorithm, we study a complex-valued MLNN, i.e. a MLNN with *complex-valued* weights and activity patterns. A complex-valued MLNN is a generalization of the real-valued MLNN since a complex-valued number represents two values, i.e. a real and an imaginary value. Consequently, the BP algorithm must be modified in order to adapt a complex-valued MLNN by taking into account the imaginary part as well.

In Section 2.1 the functional architecture of real-valued MLNNs is described. The Back-Propagation algorithm (Rumelhart *et al.*, 1986) is

[1]The Back-Propagation algorithm is similar to an algorithm described much earlier by Werbos (1974).

explained and also a vectorized version of the generalized delta rule is presented. In Section 2.2 the functional architecture of complex-valued MLNNs is presented. We prove that the vectorized version of the generalized delta rule may also be applied to complex-valued MLNNs (Henseler and Braspenning, 1990a). When Henseler and Braspenning (1990a) was published, no other publications on the subject had appeared. We know now that several other researchers in the field of ANNs, most of them later in time, were quite independently also involved in the development of a complex-valued MLNN as well as a complex-valued BP algorithm (Clarke, 1990; Kim and Guest, 1990; Leung and Haykin, 1991; Benvenuto *et al.*, 1991).

Until then we had not paid any attention to the *unboundedness* of the complex-valued sigmoid function. The BP algorithm can still be applied provided that a small learning-rate constant is used. However, such a small learning-rate results in a very slow learning speed, which may not be acceptable. Therefore, in Section 2.3 in the second part of our research we introduce a *bounded* complex-valued neuron-output function. According to Liouville's theorem[2] (Bak and Newman, 1982; Clarke, 1990), all bounded complex-valued functions are not complex differentiable unless they are constant. Hence, the gradient descent in the weight space of such a MLNN is basically different from the complex-valued MLNN that was first introduced (cf. Section 2.2). Actually, we derive a generalized complex delta rule for a complex-valued function that squashes the magnitude of its input, thereby resembling the tanh() function sometimes used in real-valued MLNNs.

Finally, in Section 2.4 an application of the complex-valued MLNN using a bounded complex-valued neuron-output function is presented. In this application the network must learn to position a robot arm with two degrees of freedom using input generated by two direction sensors. The results show that a complex-valued MLNN can outperform a real-valued MLNN. In Section 2.5 the main results of this chapter are summarized.

[2]The mathematical properties of complex numbers can be found in many mathematical books. For instance, we used a recent standard publication of Bak and Newman (1982) entitled "Complex Analysis".

2.1 Back Propagation

As we saw in Chapter 1, a Perceptron is not capable of classifying non-linearly separable classes, as illustrated by the XOR problem (see Figure 1.9(c)). This problem may be solved by introducing so-called *hidden neurons* in the network (Minsky and Papert, 1969). Unfortunately, the connection weights to such neurons cannot be adapted using the *delta rule* described in Equation 1.4 because there is no target value specified for hidden outputs. However, by using neurons with a differentiable output function it is possible to calculate a delta error for the hidden neurons based on the error gradient of the output neurons. Since the threshold function used in the Perceptron is not differentiable it is replaced by a continuous sigmoid function resembling a threshold function. The *extended* delta rule for adapting weights to hidden neurons is called the *generalized delta rule*. This rule represents the heart of the popular BP algorithm for adapting MLNNs (Rumelhart *et al.*, 1986).

2.1.1 Functional Architecture

In Chapter 1 a framework for description of the functional architecture of ANNs was introduced. Here the functional architecture of MLNNs is presented, describing their connection architecture and neuron architecture. Furthermore, we present a short description of the MLNN activation dynamics following directly from the functional architecture.

Connection Architecture

A MLNN is essentially a Perceptron augmented with hidden neurons. The neurons in a MLNN are placed in a number of consecutive layers, viz. an input neuron layer, zero or more hidden layers, and an output layer. If there are no hidden layers a MLNN is, in fact, equivalent to a Perceptron. A neuron in layer i receives input from every neuron in layer $i-1$. However, neurons in the same layer and neurons with one or more other layers in between are *not* interconnected. Neurons in the input layer are virtual neurons which receive no input. Their output represents input features which is presented to the first hidden layer, or, if there are no hidden layers,

directly to the output layer. Neurons in a hidden layer that do not receive inputs from the input layer are connected to the neurons in the previous hidden layer. Hence, the output of a hidden neuron is sent to the next layer, which may either be another hidden layer or the output layer. Finally, the output layer sends its output to the environment. A network consisting of $N + 1$ layers is depicted in Figure 2.1. We have denoted the number of neurons in layer p by m_p. We note that the input layer corresponds to layer 0.

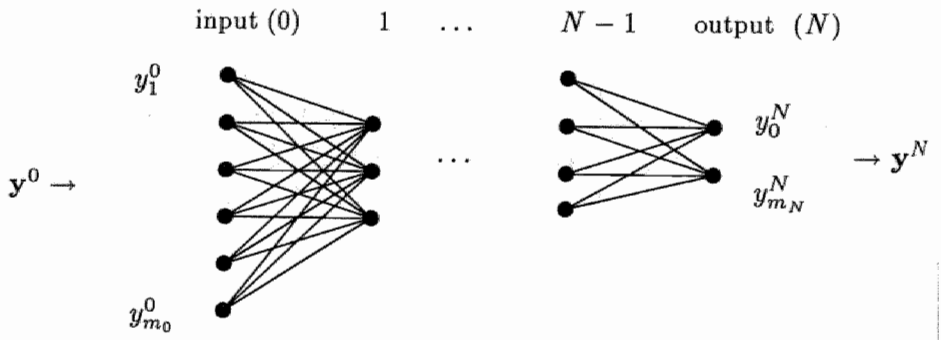


Figure 2.1: *A network consisting of N layers.*

The connections have real-valued *weights* and the input and the output are also real valued. The connection weight *to* the i-th neuron in layer p *from* the j-th neuron in layer $p-1$ is indicated by w_{ij}^p. Two connected layers $p-1$ and p with their connections and corresponding weights are shown in Figure 2.2.

Neuron Architecture

MLNN neurons strongly resemble the neurons in a Perceptron. The principal difference is the discrete function in the Perceptron neuron-processing model, i.e. the threshold function (cf. Equation 1.1), is replaced by a continuous output function. Usually, a *sigmoid* function is used in MLNNs (cf.

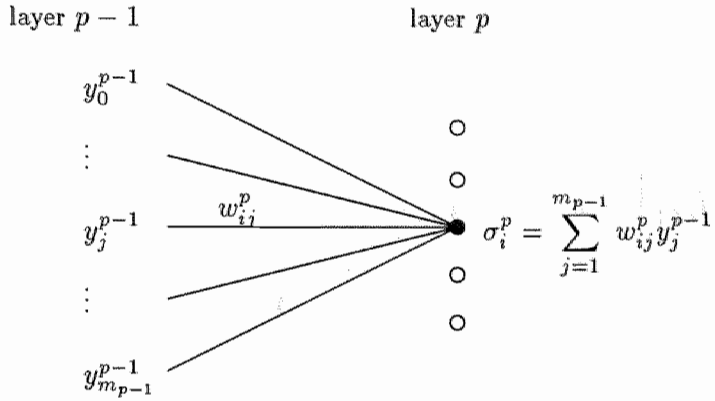


Figure 2.2: *Organization of weights in a multi-layer neural network.*

Figure 2.3(a)) (Rumelhart *et al.*, 1986).

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

Sometimes other functions are used, for instance $\tanh(x)$ (cf. Figure 2.3(b)). The $\tanh(x)$ function is essentially identical to the sigmoid function and is particularly useful when network output should range between -1 and 1.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\frac{1}{1+e^{-2x}} - 1 = 2f(2x) - 1 \tag{2.2}$$

When input is presented to the neurons in layer one, the network output is calculated by propagating the input forward, i.e. from the first layer through the consecutive hidden layers to the output layer. Hence, this procedure is called *forward propagation*. The architecture of neurons in a MLNN may be described using the three-stage processing model (cf. Figure 1.12). In the third entry of Table 1.2 a compact description of the MLNN neuron-processing model is presented. We shall elaborate this

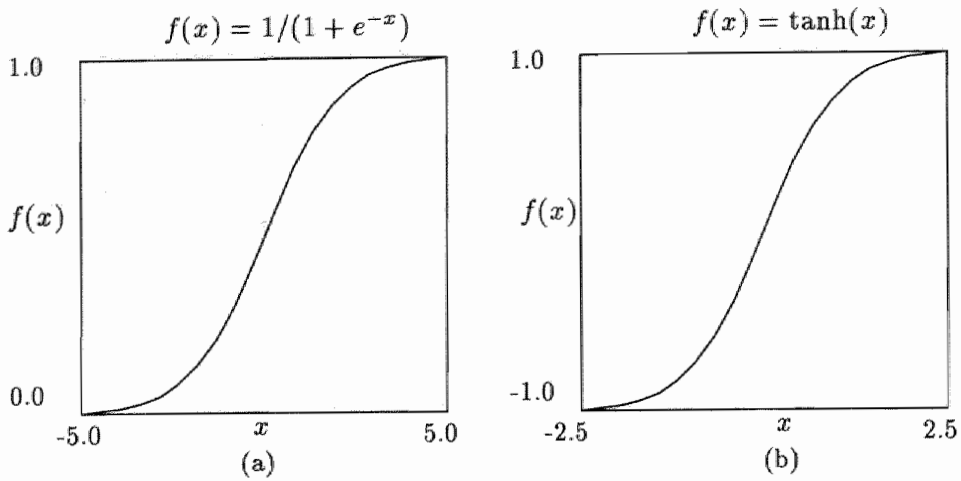


Figure 2.3: *Two typical output functions used in MLNNs: (a) sigmoid function, (b)* $\tanh(x)$ *function.*

model using the notation introduced in Figures 2.1 and 2.2. Then the net input σ_i^p and the neuron output y_i^p resulting from first and third processing stage respectively, are calculated as follows

Stage I: In the first stage a weighted sum of the neuron input is calculated (cf. Figure 2.2)

$$\sigma_i^p = \sum_{j=1}^{m_{p-1}} w_{ij}^p y_j^{p-1} \tag{2.3}$$

We note that the weighted sum of the neurons in layer $p = 1$ is calculated from the network input $\mathbf{y}^0$, i.e. the output of the input layer $p = 0$ containing virtual neurons.

Stage II: The second stage is void, i.e. the weighted sum σ is directly entered into the third stage.

Stage III: In most cases the output function of a MLNN is a sigmoid function (cf. Figure 2.3(a))

$$y_i^p = f(\sigma_i^p) = \frac{1}{1 + e^{-\sigma_i^p}} \tag{2.4}$$

We note that the sigmoid function is a continuous, differentiable function. Moreover, its shape (cf. Figure 2.3) resembles the step function used in a Perceptron. In the Perceptron, however, the step function was additionally characterized by a threshold value. In the sigmoid function presented in Equation 2.4 a threshold may be added as well. A *threshold* parameter θ is introduced for each neuron by using $g(\sigma, \theta) = f(\sigma - \theta)$ instead of just $f(\sigma)$ in the processing model:

$$g(\sigma, \theta) = f(\sigma - \theta) = \frac{1}{1 + e^{-(\sigma - \theta)}} \tag{2.5}$$

In many implementations of the BP algorithm, neurons have no explicit thresholds. Instead, a so-called *bias* neuron is added to the network. The bias neuron receives no input and has constant output $y_{bias} = -1$. Neuron i in layer p has a connection $w_{i,bias}^p$ to the bias neuron. Hence, the weighed-input sum σ_i^p is calculated as

$$\sigma_i^p = \sum_{j=1}^{m_{p-1}} w_{ij}^p y_j^{p-1} + w_{i,bias}^p \underbrace{y_{bias}}_{=-1} = \sum_{j=1}^{m_{p-1}} w_{ij}^p y_j^{p-1} - w_{i,bias}^p \tag{2.6}$$

We note that the connection weight $w_{i,bias}^p$ between neuron y_i^p and the bias neuron equals the threshold θ_i^p.

Activation Dynamics

The activation dynamics of MLNNs are rather straightforward since the connection architecture contains no recurrent connections and the neuron architecture contains no differential equation. Hence, the network is a zeroth-order system with an activity pattern that is static for any activity

pattern presented to the input neurons of the network. It is worth noting that the network outputs are restricted to the interval $[0, 1]$ if a sigmoid output function is used, and to the interval $[-1, 1]$ if the $\tanh(x)$ function is used.

2.1.2 Gradient Descent

The BP algorithm may be used to adapt the connection weights in a MLNN in order to realize a particular input-output behaviour. Similar to the Perceptron learning procedure, Back Propagation is based on a delta rule, i.e. a rule which calculates a weight change based on a delta error between the target output and the actual output. However, in case of a MLNN no target is known for the hidden neurons. It is shown by Rumelhart *et al.* (1986) that the Perceptron delta rule may be generalized so that connection weights to hidden neurons may also be adapted.

We introduce a *vector* notation which enables us to exploit the regular structure of the network finally resulting in a simpler generalized delta rule. Moreover, introducing a vector notation facilitates the application of the generalized delta rule to complex-valued MLNNs (Henseler and Braspenning, 1990a). Thus, the net inputs to and the outputs from the neurons in layer p are denoted by vectors σ^p and $\mathbf{y}^p$, respectively.

$$\sigma^p = \begin{pmatrix} \sigma_1^p \\ \vdots \\ \sigma_{m_p}^p \end{pmatrix}, \mathbf{y}^p = \begin{pmatrix} y_1^p \\ \vdots \\ y_{m_p}^p \end{pmatrix} \tag{2.7}$$

The lower indices in Equation 2.7 run over the neurons in layer p. Appropriately, the weights between layers $p-1$ and p in a MLNN can be written in a *weight matrix* $\mathbf{W}^p$:

$$\mathbf{W}^p = \begin{pmatrix} w_{11}^p & \cdots & w_{1m_{p-1}}^p \\ \vdots & \ddots & \vdots \\ w_{m_p1}^p & \cdots & w_{m_pm_{p-1}}^p \end{pmatrix} \tag{2.8}$$

Let vector $\mathbf{y}^0(t)$ be the input to the network at time t. Generally, we shall leave out the (t) index. The i-th component of $\mathbf{y}^0$ denotes the activity of input neuron i. With $\mathbf{y}^0$ the input in the network, the output $\mathbf{y}^N$ may be obtained by iteratively calculating $\mathbf{y}^1, \ldots, \mathbf{y}^N$, i.e. the output of hidden layers $1, \ldots, N$, respectively. This process is called *forward propagation* of the network input. Equations 2.3 and 2.4 may be rewritten as

$$\sigma^p = \mathbf{W}^p \mathbf{y}^{p-1} \tag{2.9}$$

$$\mathbf{y}^p = f(\sigma^p) \tag{2.10}$$

The generalized delta rule is based on a *gradient-descent* method which minimizes a total output error E by adapting weights in the opposite direction of the gradient of the error surface in weight space, i.e. descending to a lower error value. The error is measured as the sum of the squared errors of the actual responses $y_1^N, \ldots, y_{m_N}^N$ and the desired (target) responses $Y_1, \ldots, Y_{m_N}$ of the neurons in the output layer. In vector notation, the error function is equal to the squared Euclidean length of the error vector, i.e. $\mathbf{Y} - \mathbf{y}^N$.

$$E = \sum_{i=1}^{m_N} (Y_i - y_i^N)^2 = \|\mathbf{Y} - \mathbf{y}^N\|^2 \tag{2.11}$$

Since the network output $\mathbf{y}^N$ is determined by the actual connection weights, the error sum is a function of the connection weights and will therefore be represented by $E(\mathbf{W}^1, \ldots, \mathbf{W}^N)$ briefly denoted as $E(w)$. The objective is to find a network weight configuration $\mathbf{W}^1, \ldots, \mathbf{W}^N$ such that the error $E(w)$ is minimal. A simple method that may be used to approximate this minimum is to adapt $\mathbf{W}^1, \ldots, \mathbf{W}^N$ in the direction of the negative gradient to a lower error value. This method is called *gradient descent*. The error gradient contains components for each weight w_{ij}^p in the network. In a gradient descent, each weight w_{ij}^p is adapted proportionally to the negative partial derivative of $E(w)$ with respect to w_{ij}^p.

$$\Delta w_{ij}^p \propto -\frac{\partial E(w)}{\partial w_{ij}^p} \tag{2.12}$$

We note, however, that gradient descent does not guarantee a global minimum to be found. It is only guaranteed that a local minimum will be found.

2.1.3 The Generalized Delta Rule

The generalized delta rule may be derived by rewriting Equation 2.12. The chain rule is used to rewrite the partial derivative on the right-hand side:

$$\frac{\partial E(w)}{\partial w_{ij}^p} = \underbrace{\frac{\partial E(w)}{\partial \sigma_i^p}}_{\delta_i^p} \frac{\partial \sigma_i^p}{\partial w_{ij}^p} = \delta_i^p \underbrace{\frac{\partial}{\partial w_{ij}^p} \sum_{k=1}^{m_{p-1}} w_{ik}^p y_k^{p-1}}_{=0 \; k \neq j} = \delta_i^p y_j^{p-1} \tag{2.13}$$

The factor δ_i^p introduced in Equation 2.13 is called the *delta error* of the i-th neuron in layer p. The change for an entire weight matrix $\mathbf{W}^p$ is obtained by removing the indices i and j from Equation 2.13. This means that the weight change equals the product of δ in layer p and the output $\mathbf{y}^{\mathrm{T}}$ of the previous layer $p-1$. Superscript $^{\mathrm{T}}$ denotes the transpose of a vector, i.e. $\mathbf{y}^{\mathrm{T}}$ is a row vector and the product of δ and $\mathbf{y}^{\mathrm{T}}$ is a matrix.

$$\Delta \mathbf{W}^p = -\eta \delta^p \mathbf{y}^{p-1^{\mathrm{T}}} \tag{2.14}$$

The constant η is called the *learning rate*. Since the adaptation should be in the opposite direction of the gradient, η is a positive real number. Increasing the learning rate on the one hand speeds up the adaptation process but on the other hand may introduce oscillations in the value of $E(w)$ meaning that the descent along the error surface is not effective. One method that is often used to speed up the adaptation process without introducing oscillations is to modify Equation 2.14 by including a second-order term called a *momentum* (Rumelhart *et al.*, 1986). The momentum represents a fraction of the previous weight adaptation. Let $\Delta \mathbf{W}^p(n)$ denote the weight adaptation at the n-th iteration, then Equation 2.14 is modified as follows taking iteration history into account.

$$\Delta \mathbf{W}^p(n+1) = -\eta \delta^p \mathbf{y}^{p-1^{\mathrm{T}}} + \alpha \Delta \mathbf{W}^p(n) \tag{2.15}$$

The momentum coefficient $\alpha \in [0, 1)$ is a constant which determines the effect of past weight changes on the current direction of the gradient descent. The momentum term filters high-frequency oscillations in the error value, i.e. it suppresses oscillations allowing the learning rate to be larger.

The derivative of the error $E(w)$ with respect to the net input σ_i^p (i.e. δ_i^p cf. Equation 2.13) in a neuron may be calculated by applying the chain rule again. We calculate the partial derivative of $E(w)$ with respect to y_j^p and multiply this with the partial derivative of y_j^p with respect to the net input σ_i^p of neuron j in layer p.

$$\delta_i^p = \frac{\partial E(w)}{\partial \sigma_i^p} = \sum_{j=1}^{m_p} \frac{\partial E(w)}{\partial y_j^p} \frac{\partial y_j^p}{\partial \sigma_i^p} \tag{2.16}$$

Although the partial derivative of y_j^p to σ_i^p is zero if $i \neq j$ the summation over $j = 1, \ldots, m_p$ in Equation 2.16 is entered to formulate δ_i^p as the inner product of two vectors.

$$\delta_i^p = \left(\frac{\partial E(w)}{\partial y_1^p}, \ldots, \frac{\partial E(w)}{\partial y_{m_p}^p} \right) \begin{pmatrix} \partial y_1^p / \partial \sigma_i^p \\ \vdots \\ \partial y_{m_p}^p / \partial \sigma_i^p \end{pmatrix} \tag{2.17}$$

This equation can be extended to a matrix multiplication resulting in $\delta^{p\mathrm{T}} = (\delta_1^p, \ldots, \delta_{m_p}^p)$:

$$\delta^{p\mathrm{T}} = \left(\frac{\partial E(w)}{\partial y_1^p}, \ldots, \frac{\partial E(w)}{\partial y_{m_p}^p} \right) \underbrace{\begin{pmatrix} \partial y_1^p / \partial \sigma_1^p & \ldots & \partial y_1^p / \partial \sigma_{m_p}^p \\ \vdots & & \vdots \\ \partial y_{m_p}^p / \partial \sigma_1^p & \ldots & \partial y_{m_p}^p / \partial \sigma_{m_p}^p \end{pmatrix}}_{\Psi^p} \tag{2.18}$$

Matrix Ψ^p is called the layer differential matrix and the delta error δ^p in layer p is calculated as follows:

$$\delta^p = \Psi^{p\mathrm{T}} \frac{\partial E(w)}{\partial \mathbf{y}^p} \tag{2.19}$$

In a MLNN neurons in the same layer are not interconnected; hence, matrix Ψ^p is diagonal since the i-th component of $\mathbf{y}^p$ depends on σ_i^p only, i.e. $\partial y_j^p / \partial \sigma_i^p = 0$ if $i \neq j$. It can be shown from the definition of f in Equation 2.4 that if $y = f(\sigma)$ then $y' = f'(\sigma) = y(1-y)$. Hence, Ψ^p can be presented as

$$\Psi^p = \begin{pmatrix} y_1^p(1-y_1^p) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & y_{m_p}^p(1-y_{m_p}^p) \end{pmatrix} \tag{2.20}$$

The derivative of $E(w)$ with respect to y_j^p in Equation 2.16 can only be calculated directly for $p = N$, i.e. for the output layer. This was expected since the error is measured in the output layer for which the target values $\mathbf{Y}$ are specified by the example. According to the definition of $E(w)$ in Equation 2.11, the derivative for the output layer is:

$$\frac{\partial E(w)}{\partial \mathbf{y}^N} = -2(\mathbf{Y} - \mathbf{y}^N) = 2(\mathbf{y}^N - \mathbf{Y}) \tag{2.21}$$

The factor 2 will be left out by including it in the learning rate η (cf. Equation 2.15). The delta error in layer N can be calculated using Equation 2.19. The error δ^p in hidden layer p is calculated in terms of the error made by the subsequent layer $p+1$ (up to N):

$$\begin{aligned} \frac{\partial E(w)}{\partial y_i^p} &= \sum_{j=1}^{m_{p+1}} \underbrace{\frac{\partial E(w)}{\partial \sigma_j^{p+1}}}_{\delta_j^{p+1}} \frac{\partial \sigma_j^{p+1}}{\partial y_i^p} = \sum_{j=1}^{m_{p+1}} \delta_j^{p+1} \underbrace{\frac{\partial}{\partial y_i^p} \sum_{l=1}^{m_p} w_{jl}^{p+1} y_l^p}_{=0 \text{ if } l \neq i} \\ &= \sum_{j=1}^{m_{p+1}} \delta_j^{p+1} w_{ji}^{p+1} \end{aligned} \tag{2.22}$$

This result enables the formulation of the generalized delta rule for δ_i^p in its usual form by substituting Equation 2.22 into Equation 2.16 and denoting

$\partial y_i^p / \partial \sigma_i^p = f'(\sigma_i^p)$:

$$\delta_i^p = f'(\sigma_i^p) \sum_{j=1}^{m_{p+1}} \delta_j^{p+1} w_{ji}^{p+1} \tag{2.23}$$

However, we shall rewrite Equation 2.22 as a matrix multiplication resulting in $(\partial E(w)/\partial \mathbf{y}^p)^{\mathrm{T}} = (\partial E(w)/\partial y_1^p, \ldots, \partial E(w)/\partial y_{m_p}^p)$:

$$\left(\frac{\partial E(w)}{\partial \mathbf{y}^p}\right)^{\mathrm{T}} = (\delta_1^{p+1}, \ldots, \delta_{m_{p+1}}^{p+1}) \begin{pmatrix} w_{11}^{p+1} & \cdots & w_{1m_p}^{p+1} \\ \vdots & & \vdots \\ w_{m_{p+1}1}^{p+1} & \cdots & w_{m_{p+1}m_p}^{p+1} \end{pmatrix} \tag{2.24}$$

which reduces further to

$$\frac{\partial E(w)}{\partial \mathbf{y}^p} = \mathbf{W}^{p+1\mathrm{T}} \delta^{p+1} \tag{2.25}$$

Apparently, error propagation is formally similar to forward propagation (cf. Equation 2.10) except that it is in the opposite direction, which is why it is called *back propagation.*

Combining the results found in Equations 2.19, 2.21 and 2.25 the following equations can be formulated for calculating the delta error vector δ^p:

$$\delta^p = \begin{cases} \Psi^{p\mathrm{T}} \mathbf{W}^{p+1\mathrm{T}} \delta^{p+1} & 1 < p < N \\ \Psi^{N\mathrm{T}} (\mathbf{y}^N - \mathbf{Y}) & p = N \end{cases} \tag{2.26}$$

Equation 2.26 is also referred to as the *generalized delta rule* and is used when calculating the weight change $\Delta \mathcal{W}$ defined in Equation 2.15. This rule is a generalized version of the *delta rule* (cf. Equation 1.4) used for adapting weights in layered networks without hidden layers, as in a Perceptron (Rosenblatt, 1959; Rosenblatt, 1962).

It can be shown that the generalized delta rule also applies to threshold adaptation when a threshold parameter is added to the sigmoidal function

(cf. Equation 2.5), i.e. Equation 2.10 becomes $\mathbf{y}^p = f(\sigma^p - \theta^p)$. If the j-th neuron in layer p has threshold θ_j^p then the threshold change (Δ) to minimize the error $E(w)$ is given by:

$$\Delta\theta_i^p = -\eta\frac{\partial E(w)}{\partial\theta_i^p} = -\eta\frac{\partial E(w)}{\partial y_i^p}\frac{\partial y_i^p}{\partial\theta_i^p} = -\eta\frac{\partial E(w)}{\partial y_i^p}\frac{-\partial y_i^p}{\partial\sigma_i^p} = \eta\delta_i^p \quad (2.27)$$

We note that the final transition is accomplished by substituting Equation 2.16, taking into account that $\partial y_j^p/\partial\sigma_i^p = 0$ if $i \neq j$. Using a vector notation and adding a momentum (cf. Equation 2.15) the following threshold adaptation rule is used:

$$\Delta\theta^p(n+1) = \eta\delta^p + \alpha\Delta\theta^p(n) \quad (2.28)$$

2.2 Complex-Valued MLNNs

The functional architecture of complex-valued MLNNs is identical to the functional architecture of real-valued MLNNs described in Section 2.1. However, the connection weights and the neuron outputs as well as their inputs are complex valued. A complex-valued neuron processes its complex-valued weighted-input sum by a non-linear, complex-valued function, for instance the sigmoid function (Henseler and Braspenning, 1990a; Kim and Guest, 1990; Clarke, 1990; Leung and Haykin, 1991). The complex-valued MLNN is characterized by the complex-valued neuron architecture. Hence, we start analyzing the underlying complex-valued processing model focussing on stages I and III, corresponding to the calculation of the complex-valued weighted-input sum and the complex-valued output function respectively.

Using the BP algorithm for adapting complex-valued MLNNs requires the presence of a generalized complex delta rule (Henseler and Braspenning, 1990a). In this section we shall prove that the vectorized generalized delta rule described in Equation 2.26 is still applicable when a complex-differentiable output function, e.g. the complex-valued sigmoid function, is used. Independent of the work presented in this section others have published similar derivations (see, e.g. Kim and Guest (1990), Clarke (1990), Leung and Haykin (1991)).

2.2.1 Complex-Valued Processing Model

The connection architecture of a complex-valued MLNN contains complex-valued weights. Like real-valued weights, they weigh input by multiplication of two (complex) numbers. However, due to properties of complex numbers this weighting operation is quite different, as we shall demonstrate. Let y_1 and y_2 be the output of two neurons each receiving two inputs x_1 and x_2. Let us first define the real-valued weight model:

$$\begin{cases} y_1 = x_1 w_{11} + x_2 w_{12} \\ y_2 = x_1 w_{21} + x_2 w_{22} \end{cases} \tag{2.29}$$

In the complex-valued MLNN only two weighting coefficients are used instead of four. The weights represent a complex value, i.e. $\mathbf{Re}w$ represents the real weight and $\mathbf{Im}w$ represents the imaginary weight. Also the input and output are coded as complex values, i.e. $(x_1, x_2) = (\mathbf{Re}x, \mathbf{Im}x)$ and $(y_1, y_2) = (\mathbf{Re}y, \mathbf{Im}y)$. By introducing an explicit notation we may write a complex input as $x = \mathbf{Re}x + i\mathbf{Im}x$, a complex weight as $w = \mathbf{Re}w + i\mathbf{Im}w$ and a complex output as $y = \mathbf{Re}y + i\mathbf{Im}y$. According to complex calculation $i^2 = -1$. Hence, the complex-valued weight model reads as follows:

$$\begin{aligned} y &= wx = (\mathbf{Re}w + i\mathbf{Im}w)(\mathbf{Re}x + i\mathbf{Im}x) \\ &= \mathbf{Re}w\mathbf{Re}x + i^2\mathbf{Im}w\mathbf{Im}x + i(\mathbf{Im}w\mathbf{Re}x + \mathbf{Re}w\mathbf{Im}x) \\ &= (\mathbf{Re}w\mathbf{Re}x - \mathbf{Im}w\mathbf{Im}x) + i(\mathbf{Im}w\mathbf{Re}x + \mathbf{Re}w\mathbf{Im}x) \end{aligned} \tag{2.30}$$

The complex-valued weight model may also be described in terms of the real-valued weight model:

$$\begin{cases} y_1 = x_1\mathbf{Re}w - x_2\mathbf{Im}w \\ y_2 = x_1\mathbf{Im}w + x_2\mathbf{Re}w \end{cases} \tag{2.31}$$

A complex number can be written in a complex exponential representation. This representation is useful for analyzing the difference between the real-valued weight model and the complex-valued weight model. Let a complex

number $z = a + ib$ have a complex magnitude r and a phase angle ϕ. The complex magnitude r corresponds to $\sqrt{a^2 + b^2}$, i.e. the magnitude of z in the complex plane. The phase angle is the angle of z and real axis, i.e. $a = r\cos(\phi)$ and $b = r\sin(\phi)$. The exponential representation of z is defined as $re^{i\phi}$ and may be used to calculate the product of two complex numbers. For instance, let $z_1 = r_1 e^{i\phi_1}$ and $z_2 = r_2 e^{i\phi_2}$ then

$$z_1 z_2 = r_1 e^{i\phi} r_2 e^{i\phi_2} = r_1 r_2 e^{i(\phi_1 + \phi_2)} \tag{2.32}$$

Hence, in the complex-valued weight model it is not the *individual* inputs $\mathbf{Re}x$ and $\mathbf{Im}x$ that are weighted but the magnitude and phase of the complex value of $\mathbf{Re}x + i\mathbf{Im}x$. Like the real-valued weight model, the complex-valued weight model is a linear projection from $R^2 \rightarrow R^2$ which transforms vector (x_1, x_2) into vector (y_1, y_2). However, the complex-valued weight model is restricted, i.e. vector (x_1, x_2) can only be scaled and/or rotated. By contrast, the real-valued weight model is unrestricted, for instance vector (x_1, x_2) can be mirrored, e.g. $(y_1, y_2) = (x_2, x_1)$, or x_1 can be scaled differently from x_2, e.g. $(y_1, y_2) = (2x_1, 3x_2)$. A possible advantage of the restriction in the complex-valued weight model is that only *two* weights, i.e. $\mathbf{Re}w$ and $\mathbf{Im}w$, have to be adapted to realize a mapping from input (x_1, x_2) to output (y_1, y_2). In the real-valued weight model *four* weights, i.e. w_{11}, w_{12}, w_{21} and w_{22} must be adapted. It is a generally accepted strategy that the number of weights should be kept as small as possible when adapting MLNNs with the BP algorithm (Rumelhart, 1988; Weigend *et al.*, 1990). Obviously, a smaller number of weights yields a faster adaptation process since fewer computations are required. More importantly, a network with fewer weights must generalize better than a network with more weights since it does not have so many resources to adapt to particular examples.

The complex-valued MLNN uses the same processing model as the real-valued MLNN. However, the vectors and matrices have a complex-valued structure:

$$\begin{aligned} \sigma^p &= \mathbf{Re}\sigma^p + i\mathbf{Im}\sigma^p \\ \mathbf{y}^p &= \mathbf{Re}\mathbf{y}^p + i\mathbf{Im}\mathbf{y}^p \\ \mathbf{W}^p &= \mathbf{ReW}^p + i\mathbf{ImW}^p \end{aligned} \tag{2.33}$$

In complex-valued MLNNs the output function is a function of a complex variable $z = a + ib$. Since a complex variable can be viewed as a pair of real and imaginary values, a complex function is a function of two real variables a and b. However, in some cases the function may be a "direct" function of $z = a + ib$. For instance, the function $a^2 - b^2 + 2iab$ is a direct function of z because $a^2 - b^2 + 2iab = (a + ib)^2 = z^2$. Moreover, we observe that $a^2 + b^2$ is not expressible as a polynomial in $a + ib$. We recall that the neuron output function in a MLNN must be differentiable for a gradient-descent technique such as Back Propagation to be applicable. Therefore we introduce the following *complex-valued sigmoid* function of $z = a + ib$:

$$f(z) = \frac{1}{1 + e^{-(a+ib)}} \tag{2.34}$$

Since the real sigmoid $f(x)$ is differentiable in x, it follows that the complex sigmoid $f(z)$ is differentiable in z. Hence, the derivative of $f(z)$ with respect to z is identical to the real-valued derivative, i.e. $f'(z) = f(z)(1 - f(z))$. In the remaining part of this section we shall show that the vectorized generalized delta rule is also applicable to the adaptation of complex-valued MLNNs if neurons have a differentiable output function.

We note that the complex-valued sigmoid function is not bounded, unlike the real-valued sigmoid function. Hence, simulation of a complex-valued MLNN may require a small learning rate in order to avoid the singularities of the network. This problem will be discussed in Section 2.3.

2.2.2 Gradient in Complex-Weight Space

We recall that the BP algorithm is based on descent in weight space in the opposite direction of the error gradient. An error gradient in real-weight space for the weight matrix of layer p equals the vector product of the output of layer $p - 1$ and the delta error in layer p (cf. Equation 2.13 in Section 2.1.3):

$$\frac{\partial E(w)}{\partial w_{ij}^p} = \frac{\partial E(w)}{\partial \sigma_i^p} \frac{\partial \sigma_i^p}{\partial w_{ij}^p} = \delta_i^p y_j^{p-1} \tag{2.35}$$

In the case of complex-valued weights, the real and imaginary derivatives should be distinguished. The expressions for the complex-valued weight model imply that both real and imaginary part of the complex-valued weight are involved in the calculation of two different sums, i.e. $\mathbf{Re}\sigma$ and $\mathbf{Im}\sigma$. Consequently, the adaptation of either part of the complex-valued weight is determined by *both parts* of the complex-valued delta error.

$$\begin{aligned}
\frac{\partial E(w)}{\partial \mathbf{Re}w_{jk}^p} &= \frac{\partial E(w)}{\partial \mathbf{Re}\sigma_j^p}\frac{\partial \mathbf{Re}\sigma_j^p}{\partial \mathbf{Re}w_{jk}^p} + \frac{\partial E(w)}{\partial \mathbf{Im}\sigma_j^p}\frac{\partial \mathbf{Im}\sigma_j^p}{\partial \mathbf{Re}w_{jk}^p} \\
&= \mathbf{Re}\delta_j^p \mathbf{Re}y_k^{p-1} + \mathbf{Im}\delta_j^p \mathbf{Im}y_k^{p-1} \\
\frac{\partial E(w)}{\partial \mathbf{Im}w_{jk}^p} &= \frac{\partial E(w)}{\partial \mathbf{Re}\sigma_j^p}\frac{\partial \mathbf{Re}\sigma_j^p}{\partial \mathbf{Im}w_{jk}^p} + \frac{\partial E(w)}{\partial \mathbf{Im}\sigma_j^p}\frac{\partial \mathbf{Im}\sigma_j^p}{\partial \mathbf{Im}w_{jk}^p} \\
&= -\mathbf{Re}\delta_j^p \mathbf{Im}y_k^{p-1} + \mathbf{Im}\delta_j^p \mathbf{Re}y_k^{p-1}
\end{aligned} \tag{2.36}$$

Actually, the derivatives in Equation 2.36 correspond to a real and imaginary error. Let us define the complex delta error as $\delta_j^p = \mathbf{Re}\delta_j^p + i\mathbf{Im}\delta_j^p$ and let y^* denote the complex conjugate of y, i.e. $y_k^{p*} = \mathbf{Re}y_k^p - i\mathbf{Im}y_k^p$. The separate equations in Equation 2.36 can be combined in such a way that the complex gradient is equal to the product of the complex delta error and the complex conjugate of the neuron output. Moreover, when the calculation of the complex-valued weight gradient is vectorized, it is the same as in Equation 2.14. The transpose operator $^{\mathrm{T}}$ not only rearranges the elements of $\mathbf{y}^{p-1}$ it also takes the complex conjugate of each vector component.

$$\frac{\partial E(w)}{\partial \mathbf{W}^p} = \delta^p \mathbf{y}^{p-1\mathrm{T}} \tag{2.37}$$

2.2.3 Generalized Complex Delta Rule

The gradient in complex-weight space can be calculated using the complex delta error δ. The generalized delta rule for calculating δ is derived by

applying the chain rule (cf. Equation 2.16 in Section 2.1.2):

$$\delta_j^p = \frac{\partial E(w)}{\partial \sigma_j^p} = \frac{\partial E(w)}{\partial y_j^p} \frac{\partial y_j^p}{\partial \sigma_j^p} \tag{2.38}$$

We note that the summation on the right-hand side of Equation 2.38 is omitted since y_j^p only depends on σ_j^p. The complex delta error is represented by a real delta error $\mathbf{Re}\delta_j^p$ and an imaginary delta error $\mathbf{Im}\delta_j^p$. Applying the chain rule requires partial differentiation to two variables because $E(w)$ depends on $\mathbf{Re}\sigma_j^p$ via both $\mathbf{Re}y_j^p$ and $\mathbf{Im}y_j^p$. This is caused by the complex output function (cf., e.g. Equations 2.34 and 2.52).

$$\begin{aligned}
\mathbf{Re}\delta_j^p &= \frac{\partial E(w)}{\partial \mathbf{Re}\sigma_j^p} = \frac{\partial E(w)}{\partial \mathbf{Re}y_j^p} \frac{\partial \mathbf{Re}y_j^p}{\partial \mathbf{Re}\sigma_j^p} + \frac{\partial E(w)}{\partial \mathbf{Im}y_j^p} \frac{\partial \mathbf{Im}y_j^p}{\partial \mathbf{Re}\sigma_j^p} \\
\mathbf{Im}\delta_j^p &= \frac{\partial E(w)}{\partial \mathbf{Im}\sigma_j^p} = \frac{\partial E(w)}{\partial \mathbf{Re}y_j^p} \frac{\partial \mathbf{Re}y_j^p}{\partial \mathbf{Im}\sigma_j^p} + \frac{\partial E(w)}{\partial \mathbf{Im}y_j^p} \frac{\partial \mathbf{Im}y_j^p}{\partial \mathbf{Im}\sigma_j^p}
\end{aligned} \tag{2.39}$$

In order to obtain a recurrent formulation for the generalized complex delta rule, we apply the chain rule once more. We recall that the recurrent definition of the generalized delta rule can be obtained by applying the chain rule to the first partial derivative of the right-hand side of Equation 2.38 (cf. Equation 2.22 in Section 2.1.3):

$$\begin{aligned}
\frac{\partial E(w)}{\partial \mathbf{Re}y_j^p} &= \sum_{k=1}^{m_{p+1}} \frac{\partial E(w)}{\partial \mathbf{Re}\sigma_k^{p+1}} \frac{\partial \mathbf{Re}\sigma_k^{p+1}}{\partial \mathbf{Re}y_j^p} + \frac{\partial E(w)}{\partial \mathbf{Im}\sigma_k^{p+1}} \frac{\partial \mathbf{Im}\sigma_k^{p+1}}{\partial \mathbf{Re}y_j^p} \\
&= \sum_{k=1}^{m_{p+1}} \mathbf{Re}\delta_k^{p+1} \mathbf{Re}w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1} \mathbf{Im}w_{kj}^{p+1} \\
\frac{\partial E(w)}{\partial \mathbf{Im}y_j^p} &= \sum_{k=1}^{m_{p+1}} \frac{\partial E(w)}{\partial \mathbf{Re}\sigma_k^{p+1}} \frac{\partial \mathbf{Re}\sigma_k^{p+1}}{\partial \mathbf{Im}y_j^p} + \frac{\partial E(w)}{\partial \mathbf{Im}\sigma_k^{p+1}} \frac{\partial \mathbf{Im}\sigma_k^{p+1}}{\partial \mathbf{Im}y_j^p} \\
&= \sum_{k=1}^{m_{p+1}} -\mathbf{Re}\delta_k^{p+1} \mathbf{Im}w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1} \mathbf{Re}w_{kj}^{p+1}
\end{aligned} \tag{2.40}$$

Substituting Equation 2.40 in Equation 2.39 yields the *generalized complex delta rule.*

$$\begin{aligned}
\mathbf{Re}\delta_j^p &= \frac{\partial \mathbf{Re}y_j^p}{\partial \mathbf{Re}\sigma_j^p} \sum_{k=1}^{m_{p+1}} \mathbf{Re}\delta_k^{p+1}\mathbf{Re}w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1}\mathbf{Im}w_{kj}^{p+1} + \\
&\quad \frac{\partial \mathbf{Im}y_j^p}{\partial \mathbf{Re}\sigma_j^p} \sum_{k=1}^{m_{p+1}} -\mathbf{Re}\delta_k^{p+1}\mathbf{Im}w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1}\mathbf{Re}w_{kj}^{p+1} \\
\mathbf{Im}\delta_j^p &= \frac{\partial \mathbf{Re}y_j^p}{\partial \mathbf{Im}\sigma_j^p} \sum_{k=1}^{m_{p+1}} \mathbf{Re}\delta_k^{p+1}\mathbf{Re}w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1}\mathbf{Im}w_{kj}^{p+1} + \\
&\quad \frac{\partial \mathbf{Im}y_j^p}{\partial \mathbf{Im}\sigma_j^p} \sum_{k=1}^{m_{p+1}} -\mathbf{Re}\delta_k^{p+1}\mathbf{Im}w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1}\mathbf{Re}w_{kj}^{p+1} \qquad (2.41)
\end{aligned}$$

A more precise formulation requires a solution of the partial derivatives of $\mathbf{Im}y_j^p$ and $\mathbf{Re}y_j^p$ with respect to $\mathbf{Im}\sigma_j^p$ and $\mathbf{Re}\sigma_j^p$, respectively, in the right-hand side of Equation 2.41. The derivatives are determined by the non-linear output function. Since this is a complex function it deserves our special attention because complex functions are only complex differentiable when specific conditions are met, i.e. their partial derivatives should satisfy the Cauchy-Riemann equations (Bak and Newman, 1982).

Theorem

Let the output of a complex neuron be defined by $f(\sigma_j^p) = y_j^p$. If $f()$ is a complex differentiable function the generalized complex delta rule (cf. Equation 2.41) reduces to the generalized delta rule (cf. Equation 2.26).

Proof

The complex function $f()$ is complex differentiable. Hence, $y_j^p = f(\sigma_j^p)$ is analytic and must therefore meet the following condition (Bak and Newman, 1982):

$$i\frac{\partial y_j^p}{\partial \mathbf{Re}\sigma_j^p} = \frac{\partial y_j^p}{\partial \mathbf{Im}\sigma_j^p} \qquad (2.42)$$

This condition can be given in terms of the real and imaginary parts of y_j^p resulting in the Cauchy-Riemann equations (Bak and Newman, 1982):

$$\frac{\partial \mathbf{Re} y_j^p}{\partial \mathbf{Re} \sigma_j^p} = \frac{\partial \mathbf{Im} y_j^p}{\partial \mathbf{Im} \sigma_j^p} \; and \; \frac{\partial \mathbf{Re} y_j^p}{\partial \mathbf{Im} \sigma_j^p} = -\frac{\partial \mathbf{Im} y_j^p}{\partial \mathbf{Re} \sigma_j^p} \tag{2.43}$$

The real and imaginary equation described in Equation 2.41 may be combined using the Cauchy-Riemann equations:

$$\begin{aligned} \delta_j^p \;=\; & \left[\frac{\partial \mathbf{Re} y_j^p}{\partial \mathbf{Re} \sigma_j^p} - i\frac{\partial \mathbf{Im} y_j^p}{\partial \mathbf{Re} \sigma_j^p}\right] \sum_{k=1}^{m_{p+1}} \left[\mathbf{Re}\delta_k^{p+1}\mathbf{Re} w_{kj}^{p+1} + \right. \\ & \left. \mathbf{Im}\delta_k^{p+1}\mathbf{Im} w_{kj}^{p+1} + i\left(-\mathbf{Re}\delta_k^{p+1}\mathbf{Im} w_{kj}^{p+1} + \mathbf{Im}\delta_k^{p+1}\mathbf{Re} w_{kj}^{p+1}\right)\right] \end{aligned} \tag{2.44}$$

Furthermore, we note that the partial derivatives that were separately defined in Equation 2.40 are now integrated in a single expression within one summation. This single expression is equal to a complex multiplication of δ_k^{p+1} and the complex conjugate of w_{kj}^{p+1*}:

$$\frac{\partial E(w)}{\partial y_j^p} = \sum_{k=1}^{m_{p+1}} \delta_k^{p+1} w_{kj}^{p+1*} \tag{2.45}$$

The complex function is differentiable and hence

$$\frac{\partial \mathbf{Re} y_j^p}{\partial \mathbf{Re} \sigma_j^p} + i\frac{\partial \mathbf{Im} y_j^p}{\partial \mathbf{Re} \sigma_j^p} = \frac{\partial y_j^p}{\partial \mathbf{Re} \sigma_j^p} = i\frac{\partial y_j^p}{\partial \mathbf{Im} \sigma_j^p} = f'(\sigma_j^p) \tag{2.46}$$

Combining equations 2.44, 2.45 and 2.46 yields the generalized complex-delta rule:

$$\delta_j^p = f'(\sigma_j^p)^* \sum_{k=1}^{m_{p+1}} \delta_k^{p+1} w_{kj}^{p+1*} \tag{2.47}$$

The proof is concluded by noting that Equation 2.47 is the same as Equation 2.23 in Section 2.1.2 except for the complex conjugate operation performed on $f'()$ and w, respectively. However, in the vectorized form of the generalized delta rule (cf. Equation 2.26) the transpose operator is applied. Since the transpose operation also takes the complex conjugate of complex numbers we may conclude that the vectorized form of the generalized delta rule is also suited for adapting complex-valued MLNNs with analytic neuron output functions.

End of Proof

The proof presented above may be illustrated by showing that the complex sigmoid function (cf. Equation 2.34) is complex differentiable. Let y denote $f(\sigma)$ where $\sigma = a + ib$ then $\partial y/\partial a = \sigma(1-\sigma)$ and $\partial y/\partial b = i\sigma(1-\sigma)$, the conditions in Equation 2.43 are satisfied since $\partial\sigma/\partial a = i\partial\sigma/\partial b$. Hence, $f(\sigma)$ is complex differentiable and its derivative has the same form as in the real-valued MLNN:

$$f'(\sigma_j^p) = y_j^p(1-y_j^p) \tag{2.48}$$

2.3 Non-Analytic Complex-Valued MLNNs

One of the reasons for using a sigmoid function in the real-valued MLNN is that it is a *bounded* function. Like the output of biological neurons, the output of neurons in a real-valued MLNN is therefore bounded. However, the sigmoid function used in the previous section is not bounded in the complex plane. The singular points of the complex-valued sigmoid function, i.e. the inputs for which it is unbounded, are studied in Section 2.3.1. Due to the singularities a small learning rate must be used in order to obtain a stable gradient descent. Using a bounded function would enable a considerable increase in the learning rate directly resulting in a better performance of the complex-valued BP algorithm.

It appears that bounded complex-valued functions are not complex differentiable. Liouville's theorem states that only constant complex functions are both complex differentiable and bounded (Bak and Newman, 1982). Since a bias neuron is the only neuron with a constant output, a regular

complex-differentiable neuron-output function will have at least one singularity (Clarke, 1990). In Section 2.3.2 a complex-valued MLNN with a non-analytic bounded neuron-output function is introduced. Consequently, this function is not complex differentiable and the MLNN requires a specialized generalized complex delta rule presented in Section 2.3.3.

2.3.1 Singular Points of the Complex Sigmoid

Complex functions that are everywhere differentiable are called *entire functions* (Bak and Newman, 1982). Many of the properties of differentiability of entire functions are analogous to those of differentiable functions of a real variable. For instance, the complex-valued sigmoid function can be differentiated as a complex function. However, there is one remarkable property of entire functions that is stated by Liouville's theorem (Bak and Newman, 1982): *A bounded entire function is constant.* This property is relevant to complex-valued MLNNs. It indicates that every non-constant entire function must be unbounded. We note that only bias neurons have a constant output function. If we require that complex-valued neurons have a complex differentiable output function they will be unbounded (Clarke, 1990). For instance, the standard sigmoid function used in real-valued MLNNs has singular points in the complex plane (Benvenuto *et al.*, 1991; Leung and Haykin, 1991). The complex-valued sigmoid function has already been described in Equation 2.34 and is repeated below.

$$f(z) = \frac{1}{1 + e^{-(a+ib)}}$$

Let $f(z) = X + iY$, then it can be shown that:

$$X = \frac{1 + e^{-a}\cos b}{\left|1 + e^{-(a+ib)}\right|^2} \tag{2.49}$$

$$Y = \frac{e^{-a}\sin b}{\left|1 + e^{-(a+ib)}\right|^2} \tag{2.50}$$

The complex sigmoid function has many singular points $a + ib$ whenever $1 + e^{-a-bi} = 0$, i.e. when

$$a = 0 \text{ and } b = \pi mod 2\pi \tag{2.51}$$

The existence of singularities need not always be a problem in simulations. The BP algorithm adjusts the weights in the MLNN to the input and output values specified in the examples. However, due to the existence of singularities *it is possible* that small weight changes lead to large changes in the output values. In other words, the learning behaviour of the complex-valued MLNN is not as stable as the learning behaviour of a real-valued MLNN. Moreover, once a singularity has occurred, numerical simulations have to be reset entirely due to an arithmetic overflow in the computer. Hence, the learning rate must be set at a small value, causing the BP algorithm for complex-valued MLNNs to learn considerably slower than for real-valued MLNNs.

The only way to avoid singularities is to use non-analytic complex functions. Either a non-analytic function may be derived from the analytic function or a totally new function may be used. In the former case all singular points in the analytic function should be eliminated, for instance by using only the *principal* values (Leung and Haykin, 1991). In the latter case, interesting results are obtained when a typical real-valued MLNN output function, e.g. the sigmoid, is applied to the real and imaginary part of the weighted-input sum separately (Benvenuto *et al.*, 1991). In the next section we shall propose another type of non-analytic function for complex-valued MLNNs.

2.3.2 The Complex Squashing Function

Inputs to a real-valued neuron are numbers with two properties, viz. a sign and an absolute value. The input is weighted by multiplication with a weight in such a way that the sign of the weight changes the sign of the input, i.e. $++ = -- = +$ and $+- = -+ = -$, and the absolute value of the weight changes the absolute value of the input. Identifying the relevant properties of a complex-valued neuron, we compare its phase angle ϕ with the sign of a real number and its magnitude r with the absolute real

value. Hence, in a complex-valued MLNN *phase angle* and *magnitude* are elementary properties because sign and absolute value are elementary in a real-valued MLNN.

The output of a non-analytic complex-valued function is a function from $R^2 \rightarrow R^2$.

$$(\mathbf{Re}y_j^p, \mathbf{Im}y_j^p) = f(\mathbf{Re}\sigma_j^p, \mathbf{Im}\sigma_j^p) \tag{2.52}$$

We recall that the sigmoid function in a real-valued neuron is intended as a threshold or step function "squashing" the weighted-input sum. We have seen that the same sigmoid function has a completely different behaviour in the complex plane and bears no resemblance to the step function due to periodical singular points at which the function is unbounded. If we want to maintain the *squashing* characteristic we should determine which aspect of the complex number is to be squashed. Essentially, the $\tanh(x)$ output function for real-valued MLNNs (cf. Equation 2.2) squashes the absolute value of x while its sign remains unchanged; that is, $\tanh(x)$ is an odd function, i.e. $\tanh(-x) = -\tanh(x)$ (cf. Figure 2.3(b)). Hence, we may rewrite $\tanh(x)$ using the absolute value $|x|$ and sign $sign(x)$ of its argument:

$$\tanh(x) = sign(x)\tanh(|x|) \tag{2.53}$$

This formulation presents a useful basis for a generalization to the complex plane since we identified $|x|$ and $sign(x)$ as main characteristics of a real number. We define a complex function of $z = re^{i\phi}$ that operates in a similar way as the $\tanh(x)$ function, substituting $|x|$ by the complex magnitude r and $sign(x)$ by the complex exponential $e^{i\phi}$.

$$f(z) = e^{i\phi}\tanh(r) \tag{2.54}$$

Because of its resemblance to the $\tanh(x)$ function we call this function the *complex squashing function*. We note that it is a non-analytic complex function and hence it is not complex differentiable. In the next section we show how a generalized complex delta rule may be combined with the partial derivatives of a non-analytic complex function.

2.3.3 Gradient of the Complex Squashing Function

The complex squashing function (cf. Equation 2.54) is a non-analytic complex function. Hence, the generalized delta rule is no longer appropriate. Instead, a specialized rule must be derived to calculate the four partial derivatives used in Equation 2.41. Since the derivation is rather long we perform the following substitutions: $a = \mathbf{Re}\sigma_j^p$, $b = \mathbf{Im}\sigma_j^p$, $X = \mathbf{Re}y_j^p$ and $Y = \mathbf{Im}y_j^p$. Let $z = a+ib = re^{i\phi}$ and let $f(z) = X+iY$ denote the complex squashing function of z, i.e. $X + iY = \tanh(r)e^{i\phi} = \tanh(r)z/r$. Hence, $X = a\tanh(r)/r$ and $Y = b\tanh(r)/r$. The gradient of $f(z)$ in $z = a + ib$ is calculated as follows:

$$\frac{\partial X}{\partial a} = \frac{\partial a\frac{\tanh(r)}{r}}{\partial a} = \frac{\tanh(r)}{r} + a\frac{\partial\frac{\tanh(r)}{r}}{\partial r}\frac{\partial r}{\partial a} \tag{2.55}$$

Analogously, the other partial derivatives, i.e. $\partial X/\partial b$, $\partial Y/\partial a$ and $\partial Y/\partial b$ can be calculated. The partial derivative with respect to r is solved separately yielding

$$\frac{\partial\frac{\tanh(r)}{r}}{\partial r} = \frac{r(1-\tanh^2(r)) - \tanh(r)}{r^2} \tag{2.56}$$

Since $r = \sqrt{a^2+b^2}$, the partial derivative of r with respect to a equals $\partial r/\partial a = a/r$. Similarly, $\partial r/\partial b = b/r$. Deriving all four partial derivatives for Equation 2.41 yields:

$$\frac{\partial\mathbf{Re}y_j^p}{\partial\mathbf{Re}\sigma_j^p} = \frac{\tanh(r)}{r} + \frac{\mathbf{Re}\sigma_j^{p2}}{r}\frac{\{r(1-\tanh^2(r)) - \tanh(r)\}}{r^2} \tag{2.57}$$

$$\frac{\partial\mathbf{Re}y_j^p}{\partial\mathbf{Im}\sigma_j^p} = \frac{\mathbf{Re}\sigma_j^p\mathbf{Im}\sigma_j^p}{r}\frac{\{r(1-\tanh^2(r)) - \tanh(r)\}}{r^2} \tag{2.58}$$

$$\frac{\partial\mathbf{Im}y_j^p}{\partial\mathbf{Re}\sigma_j^p} = \frac{\mathbf{Re}\sigma_j^p\mathbf{Im}\sigma_j^p}{r}\frac{\{r(1-\tanh^2(r)) - \tanh(r)\}}{r^2} \tag{2.59}$$

$$\frac{\partial \mathbf{Im} y_j^p}{\partial \mathbf{Im} \sigma_j^p} = \frac{\tanh(r)}{r} + \frac{\mathbf{Im}\sigma_j^{p2}}{r} \frac{\{r(1-\tanh^2(r)) - \tanh(r)\}}{r^2} \tag{2.60}$$

We note that the Cauchy-Riemann equations (cf. Equations 2.43) are not met and hence the complex squashing function is not analytic as stated before. Substituting the partial derivatives in Equation 2.41 provides a useful rule enabling a BP algorithm to adapt complex MLNNs using a complex-squashing function.

In the previous section it was proven that the vectorized generalized delta rule can be applied. For a non-analytic function we can also derive a vectorized generalized complex delta rule. Because the output function is not complex differentiable the layer differential matrix Ψ^p must be extended introducing Ψ_1^p, Ψ_2^p, Ψ_3^p and Ψ_4^p. As in Equations 2.18 and 2.20, these matrices are also diagonal matrices:

$$\begin{aligned}
\Psi_1^p &= \mathrm{diag}\left\{\frac{\partial \mathbf{Re} y_1^p}{\partial \mathbf{Re}\sigma_1^p}, \ldots, \frac{\partial \mathbf{Re} y_{m_p}^p}{\partial \mathbf{Re}\sigma_{m_p}^p}\right\} \\
\Psi_2^p &= \mathrm{diag}\left\{\frac{\partial \mathbf{Re} y_1^p}{\partial \mathbf{Im}\sigma_1^p}, \ldots, \frac{\partial \mathbf{Re} y_{m_p}^p}{\partial \mathbf{Im}\sigma_{m_p}^p}\right\} \\
\Psi_3^p &= \mathrm{diag}\left\{\frac{\partial \mathbf{Im} y_1^p}{\partial \mathbf{Re}\sigma_1^p}, \ldots, \frac{\partial \mathbf{Im} y_{m_p}^p}{\partial \mathbf{Re}\sigma_{m_p}^p}\right\} \\
\Psi_4^p &= \mathrm{diag}\left\{\frac{\partial \mathbf{Im} y_1^p}{\partial \mathbf{Im}\sigma_1^p}, \ldots, \frac{\partial \mathbf{Im} y_{m_p}^p}{\partial \mathbf{Im}\sigma_{m_p}^p}\right\}
\end{aligned} \tag{2.61}$$

We observe that Equation 2.58 is equal to Equation 2.59 and, hence $\Psi_2^p = \Psi_3^p$. Vectorizing Equation 2.41 yields for $p < N$:

$$\begin{aligned}
\mathbf{Re}\delta^p &= \Psi_1^{p\mathrm{T}} \mathrm{Re}\left\{\mathbf{W}^{p+1\mathrm{T}} \delta^{p+1}\right\} + \Psi_2^{p\mathrm{T}} \mathrm{Im}\left\{\mathbf{W}^{p+1\mathrm{T}} \delta^{p+1}\right\} \\
\mathbf{Im}\delta^p &= \Psi_3^{p\mathrm{T}} \mathrm{Re}\left\{\mathbf{W}^{p+1\mathrm{T}} \delta^{p+1}\right\} + \Psi_4^{p\mathrm{T}} \mathrm{Im}\left\{\mathbf{W}^{p+1\mathrm{T}} \delta^{p+1}\right\}
\end{aligned} \tag{2.62}$$

We note that in the previous section this extension was not required since the Cauchy-Riemann equations (cf. Equation 2.43) could be applied and

hence $\Psi_1^p = \Psi_4^p$ and $\Psi_2^p = -\Psi_3^p$. In that case it is sufficient to define $\Psi^p = \mathbf{Re}\Psi^p + i\mathbf{Im}\Psi^p$ and substitute $\Psi_1^p = \mathbf{Re}\Psi^p$, $\Psi_2^p = \mathbf{Im}\Psi^p$, $\Psi_3^p = -\mathbf{Im}\Psi^p$ and $\Psi_4^p = \mathbf{Re}\Psi^p$ in Equation 2.62 transforming it into Equation 2.26.

$$\begin{aligned} \mathbf{Re}\delta^p + i\mathbf{Im}\delta^p &= \left(\mathbf{Re}\Psi^{p\mathrm{T}} + i\mathbf{Im}\Psi^{p\mathrm{T}}\right) \\ &\quad \left(\mathrm{Re}\left\{\mathbf{W}^{p+1\mathrm{T}}\delta^{p+1}\right\} + i\mathrm{Im}\left\{\mathbf{W}^{p+1\mathrm{T}}\delta^{p+1}\right\}\right) \\ \delta^p &= \Psi^{p\mathrm{T}}\mathbf{W}^{p+1\mathrm{T}}\delta^{p+1} \end{aligned} \tag{2.63}$$

We note that $\mathbf{Im}\Psi^p = -\mathbf{Im}\Psi^{p\mathrm{T}}$ since Ψ^p is diagonal and the transpose operator takes the complex conjugate of the elements of Ψ^p.

2.4 Simulations

In the previous section we presented a generalized complex delta rule which may be used to adapt complex-valued MLNNs in which neuron output is defined by a complex squashing function (cf. Equation 2.54). We have implemented this BP algorithm in order to compare its performance and the performance of a complex-valued MLNN to the performance of a real-valued MLNN. For this purpose, we selected a robot control application which is described in Section 2.4.1. We note, however, that results have been reported indicating that complex MLNNs are also suited for signal-processing applications operating in the (complex) frequency domain (Benvenuto *et al.*, 1991).

For the robot-control application a real and a complex encoding are proposed in Section 2.4.2. A test set and a learning set corresponding to the robot model are generated. Counting the degrees of freedom, i.e. the number of weights plus the number of thresholds, the performance of the complex-valued MLNN and the real-valued MLNN are compared as well as the learning processes. However, we have confined the experiments to MLNNs with one hidden layer since emphasis is put on the complex-valued processing model based on the complex squashing function and not on the multi-layer architecture. The results are presented in Section 2.4.3, indi-

cating that for this application at least the complex-valued MLNN and BP algorithm outperform the real-valued MLNN and BP algorithm.

2.4.1 The Robot Arm

The proposed system is applied to a configuration consisting of two direction sensors and an arm, inspired by the crab system introduced by Churchland (1986). The focus of the direction sensors indicates a point that should be reached by the arm. The analytical solutions underlying the transformations of the robot arm are quite simple. They are used to generate examples required by the BP algorithm. We emphasize that the robot arm described here is rather simple compared to industrial arms. However, the control of the arm is essentially the same. If no analytical model is available, it is also possible to use BP in a reinforcement learning method (Van der Smagt and Kröse, 1991). More importantly, the examples derived from our model do not contain noise. Since training a BP MLNN with noisy robot data is a difficult problem, we found that it did not suit our purpose, i.e. comparing real- and complex-valued MLNNs.

The configuration of the robot arm and the two direction sensors is depicted in Figure 2.4, with all parameters shown that are used in the transformations below (cf. Equations 2.64 through 2.70). The sensors can be directed so that they are focussed on a particular point, e.g. the hand of the arm. The arm itself has two degrees of freedom. Firstly, the upper arm can turn 0–360 degrees around the shoulder joint, i.e. the center point. Secondly, the lower arm is connected to the upper arm at the elbow. The elbow joint is restricted to 0–180 degrees.

The transformations are composed of three stages. Equations 2.64 and 2.65 describe the transformation of the direction angles into an (x, y) coordinate representing the focus point. Equations 2.66, 2.67 and 2.68 describe the transformation of the arm dimensions and the visual focus into the angles of the "inner-arm triangle" (shoulder, elbow and hand) depicted in Figure 2.4. Finally, Equations 2.69 and 2.70 represent the transformation of the "inner-arm triangle" into the joint angles of the robot arm.

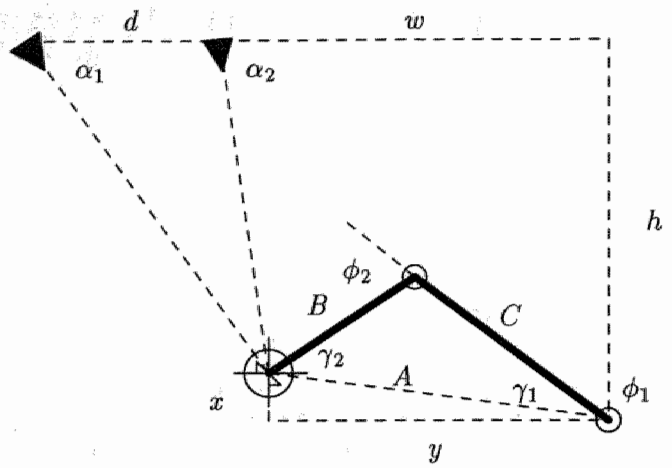


Figure 2.4: *Configuration of the robot arm and direction sensors indicating the parameters used in the mathematical transformations.*

$$x = w - d\frac{\tan(\alpha_1)}{\tan(\alpha_2) - \tan(\alpha_1)} \tag{2.64}$$

$$y = h - d\frac{\tan(\alpha_1)\tan(\alpha_2)}{\tan(\alpha_2) - \tan(\alpha_1)} \tag{2.65}$$

$$A^2 = x^2 + y^2 \tag{2.66}$$

$$\cos(\gamma_1) = \frac{A^2 - B^2 + C^2}{2CA} \tag{2.67}$$

$$\cos(\gamma_2) = \frac{A^2 + B^2 - C^2}{2BA} \tag{2.68}$$

$$\phi_1 = \pi - \arctan(\frac{y}{x}) - \gamma_1 \tag{2.69}$$

$$\phi_2 = \gamma_1 + \gamma_2 \tag{2.70}$$

Using these transformations we may generate examples as vectors representing the states of the direction sensors and the corresponding states for

the arm joints, i.e. $(\alpha_1, \alpha_2, \phi_1, \phi_2)$. In total, 400 vectors were randomly generated. They were divided into a learning set (200 examples) and a test set (200 examples).

2.4.2 Real and Complex Coding

In order to compare the complex-valued MLNN with the real-valued MLNN a real and a complex encoding are used for the examples in the learning and the test sets. Actually, the encodings are physically the same but their interpretation is different. Since physical encoding of the complex-valued and the real-valued representation is identical, it is possible to compare the sum-of-squared-errors performance of both networks.

Real Encoding

Firstly, the real encoding of the angles of the robot arm joints and the direction sensors is represented by the corresponding sine and cosine values.

$$\begin{aligned} x_1 &= \sin(\alpha_1), \; x_2 = \cos(\alpha_1) \\ x_3 &= \sin(\alpha_2), \; x_4 = \cos(\alpha_2) \\ y_1 &= \sin(\phi_1), \; y_2 = \cos(\phi_1) \\ y_3 &= \sin(\phi_2), \; y_4 = \cos(\phi_2) \end{aligned} \tag{2.71}$$

Then, using a real representation, the examples are presented to a real MLNN with four inputs (x_1, x_2, x_3, x_4) and four outputs (y_1, y_2, y_3, y_4). We note that the values of sine and cosine range between $[-1, 1]$. Hence, the neurons in the real MLNN use the $\tanh(x)$ function as their output function (cf. Figure 2.3(b)).

Complex Encoding

Secondly, the examples are encoded in a complex number using the angle as the complex phase, with complex magnitude 1.

$$\begin{aligned} x_1 &= e^{i\alpha_1} \\ x_2 &= e^{i\alpha_2} \\ y_1 &= e^{i\phi_1} \\ y_2 &= e^{i\phi_2} \end{aligned} \tag{2.72}$$

These examples are presented to a complex-valued MLNN with two complex inputs and two complex outputs. Since a complex value $e^{i\phi}$ is represented by $\cos(\phi)+i\sin(\phi)$, the same data as used for training the real-valued MLNN (cf. Equation 2.71) is used for training the complex-valued MLNN.

2.4.3 Results

The purpose of the simulations is to compare the performance of the real-valued MLNN to the performance of the complex-valued MLNN. An acceptable measure for the performance of a MLNN is the error it makes on a set of examples. These examples either may be the same examples that were used during training, i.e. the learning set, or may be different examples, i.e. the test set. We note that the performance of the BP algorithm may be influenced in various ways. For instance, by using a momentum, adaptive learning rate or a conjugate-gradient descent method. To avoid unknown influences of such extensions on either the real-valued or the complex-valued MLNN we have chosen to use standard BP with zero momentum. In all experiments we will train the networks for 100,000 epochs using a learning rate $\eta = 0.01$. An epoch is defined as one cycle through all examples in the learning set. Owing to the large number of epochs and small learning rate the momentum term may be omitted (Rumelhart *et al.*, 1986).

For a fair comparison of a real-valued MLNN and a complex-valued MLNN both networks must have received the same training effort. The training effort, i.e. the complexity of the BP algorithm, is directly related to the number of training epochs and the degrees of freedom in a MLNN, i.e. the weight and threshold values. Since the number of training epochs is fixed, the configuration of the real-valued MLNN and the complex-valued MLNN must be chosen so that they possess the same number of degrees

of freedom. A real-valued MLNN and a complex-valued MLNN with the same layer configuration have a different number of degrees of freedom. For instance, in the robot-arm experiment the complex-valued MLNN has two complex inputs and two complex outputs. Let the number of hidden neurons be n, then the network contains $2n + n2 = 4n$ complex weights and $n + 2$ complex thresholds. Since each complex weight or threshold has two degrees of freedom this amounts to $10n + 4$ degrees of freedom. The real-valued MLNN contains four real inputs and outputs corresponding to $9n + 4$ degrees of freedom.

The errors on the learning set and the test set in the real-valued MLNN are presented in Table 2.1. The experiment has been performed for 25 networks for $n = 2, 4, \ldots, 50$. In the first column of Table 2.1 the MLNN configuration is stated, in the second the corresponding number of degrees of freedom.

In Figure 2.5 a graphic representation of the errors in Table 2.1 is presented. Initially, the error on the learning and the test set drops drastically as the number of degrees of freedom increases. When ths number exceeds 100, both the learning error and the test error are approximately 0.025.

Figure 2.6 depicts the error graph of the real-valued BP algorithm applied to a 4-28-4 MLNN having 256 degrees of freedom (cf. Table 2.1). Two curves are drawn representing the learning error and the test error as a function of the number of epochs. The test curve closely resembles the learning curve indicating that the MLNN has generalized well.

Figures 2.7(a) and 2.7(b) depict the error spectrum after 100,000 epochs of the examples in the learning set and the test set respectively, for the real-valued MLNN. The spectra show that the performance of the real-valued MLNN is different for each example.

The results for the real-valued MLNN presented in Table 2.1 and Figures 2.5, 2.6 and 2.7 are also presented for the complex-valued MLNN in Table 2.2 and Figures 2.8, 2.9 and 2.10 respectively.

The learning error in the complex-valued MLNN graph depicted in Figure 2.8 decreases almost continuously in contrast to the real-valued MLNN graph in Figure 2.5. At 250 degrees of freedom the learning error of the complex-valued MLNN is approximately fifteen times smaller than the

Real-Valued MLNN			
Network	Freedoms	Learn	Test
4-2-4	22	0.0968	0.1069
4-4-4	40	0.0339	0.0401
4-6-4	58	0.0332	0.0358
4-8-4	76	0.0257	0.0284
4-10-4	94	0.0282	0.0316
4-12-4	112	0.0241	0.0251
4-14-4	130	0.0234	0.0247
4-16-4	148	0.0236	0.0248
4-18-4	166	0.0229	0.0246
4-20-4	184	0.0233	0.0260
4-22-4	202	0.0242	0.0256
4-24-4	220	0.0249	0.0268
4-26-4	238	0.0237	0.0254
4-28-4	256	0.0254	0.0268
4-30-4	274	0.0233	0.0254
4-32-4	292	0.0239	0.0259
4-34-4	310	0.0245	0.0268
4-36-4	328	0.0249	0.0269
4-38-4	346	0.0254	0.0282
4-40-4	364	0.0246	0.0267
4-42-4	382	0.0234	0.0256
4-44-4	400	0.0247	0.0263
4-46-4	418	0.0243	0.0265
4-48-4	436	0.0248	0.0273
4-50-4	454	0.0240	0.0262

Table 2.1: *Learning errors after 100,000 epochs ($\eta = 0.01$, $\alpha = 0$) of 25 real-valued MLNNs ranging from 22 to 454 degrees of freedom.*

Complex-Valued MLNN			
Network	Freedoms	Learn	Test
2-1-2	14	0.0934	0.1051
2-2-2	24	0.0424	0.0464
2-3-2	34	0.0316	0.0332
2-4-2	44	0.0182	0.0208
2-5-2	54	0.0133	0.0159
2-6-2	64	0.0099	0.0188
2-7-2	74	0.0084	0.0215
2-8-2	84	0.0065	0.0151
2-9-2	94	0.0058	0.0118
2-10-2	104	0.0056	0.0189
2-11-2	114	0.0054	0.0126
2-12-2	124	0.0043	0.0086
2-13-2	134	0.0030	0.0112
2-14-2	144	0.0025	0.0082
2-15-2	154	0.0036	0.0093
2-16-2	164	0.0031	0.0085
2-17-2	174	0.0029	0.0081
2-18-2	184	0.0030	0.0089
2-19-2	194	0.0020	0.0110
2-20-2	204	0.0017	0.0084
2-21-2	214	0.0021	0.0070
2-22-2	224	0.0020	0.0057
2-23-2	234	0.0018	0.0062
2-24-2	244	0.0015	0.0062
2-25-2	254	0.0013	0.0054

Table 2.2: *Learning errors after 100,000 epochs ($\eta = 0.01$, $\alpha = 0$) of 25 complex-valued MLNNs ranging from 14 to 254 degrees of freedom.*

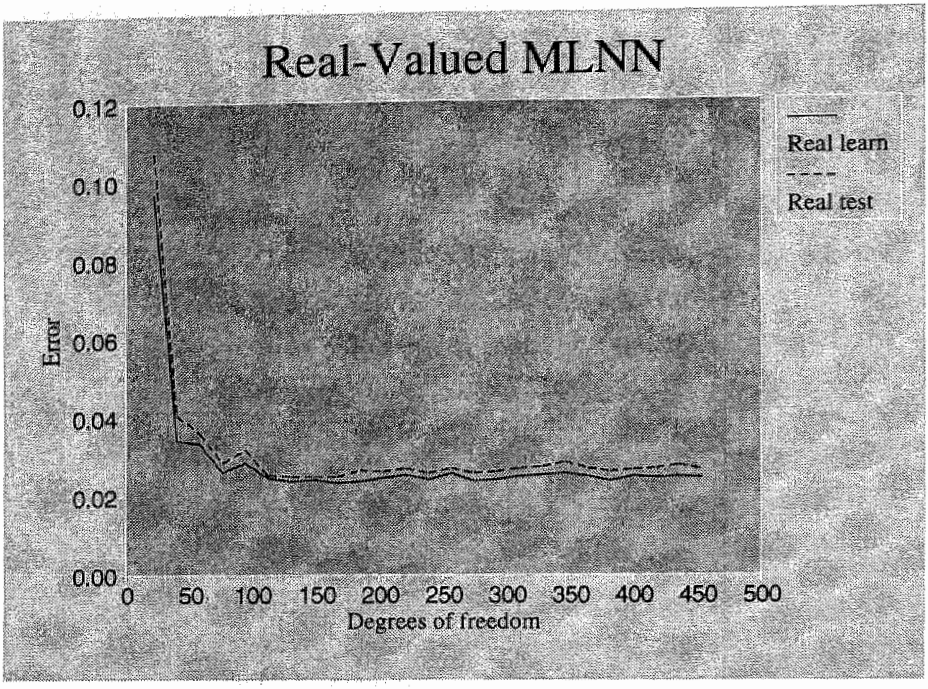


Figure 2.5: *Graph representing the errors of the real-valued MLNN as a function of the number of degrees of freedom.*

learning error of the real-valued MLNN. Moreover, a comparison of the error curves in Figure 2.6 and Figure 2.9 indicates that the error curve of the complex-valued MLNN has a much steeper initial descent than the real-valued MLNN. We note that the complex-valued MLNN studied in Figures 2.9 and 2.10 has 254 degrees of freedom; that is, almost equal to the real-valued MLNN corresponding to Figures 2.6 and 2.7, which has 256 degrees of freedom.

The error spectra of the complex-valued MLNN depicted in Figure 2.10 show that the errors on the individual examples in the learning and test set are indeed smaller than the errors in the spectra of the real-valued MLNN (cf. Figure 2.7). In this case it is also clearly visible that the mean error in

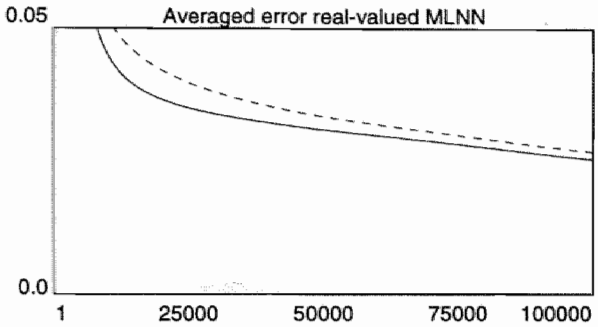


Figure 2.6: *Error curves of the real-valued Back Propagation adapting a 4-28-4 real-valued MLNN during 100,000 epochs ($\eta = 0.01$, $\alpha = 0$). The solid line represents the error on the learning set; the dashed line represents the error on the test set.*

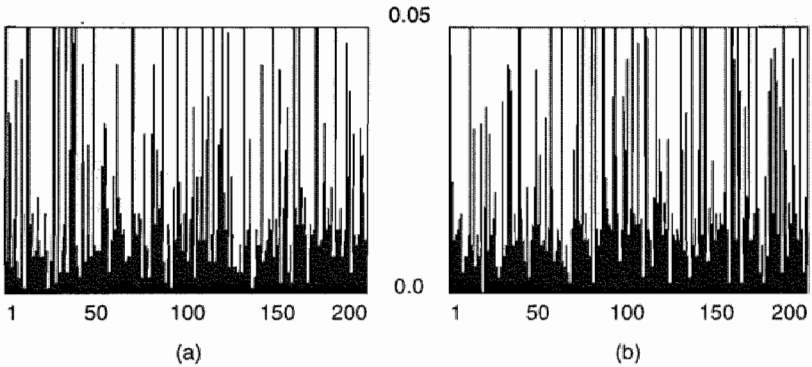


Figure 2.7: *Error spectra of the real-valued MLNN on a 4-64-4 network after 100,000 epochs ($\eta = 0.01$, $\alpha = 0$). A spectrum is displayed for the examples (a) in the learning set, and (b) in the test set.*

the test set is larger than the mean error in the learning set.

The graphs depicted in Figures 2.5 and 2.8 have been combined in a single graph that is depicted in Figure 2.11. Rather than taking the actual error values we have taken their natural logarithm to emphasize the difference.

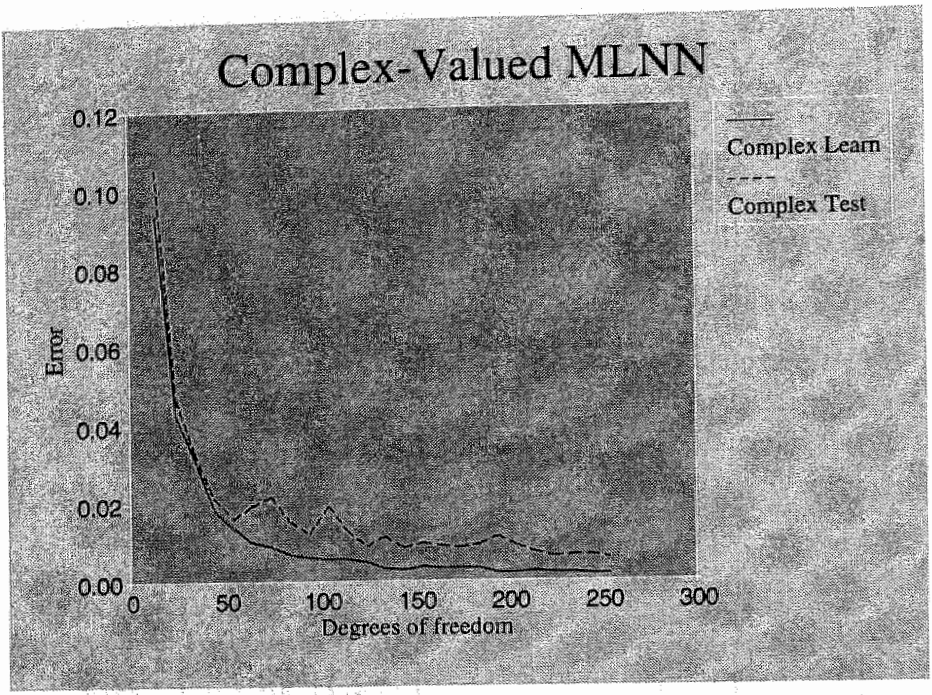


Figure 2.8: *Graph representing the errors of the complex-valued MLNN as a function of the number of degrees of freedom.*

Initially, the performance of the real-valued MLNN is only slightly worse than that of the complex-valued MLNN. As soon as the network has more than 50 degrees of freedom, the complex-valued MLNN performs considerably better than the real-valued MLNN.

Finally, we used a t-test (Mood *et al.*, 1950) for an objective comparison between the performance of the real-valued and the complex-valued MLNN. Let $\mathcal{H}_0$ denote the zero hypothesis that the mean test error of the real-valued MLNN μ_1 is equal to the mean error of the complex-valued MLNN μ_2. Let $\mathcal{H}_1$ denote the alternative hypothesis that $\mu_1 > \mu_2$, i.e. the complex-valued MLNN performs better. We want to reject $\mathcal{H}_0$ and accept $\mathcal{H}_1$ with a significance of $\alpha = 0.001$. Average μ_1 is calculated from the

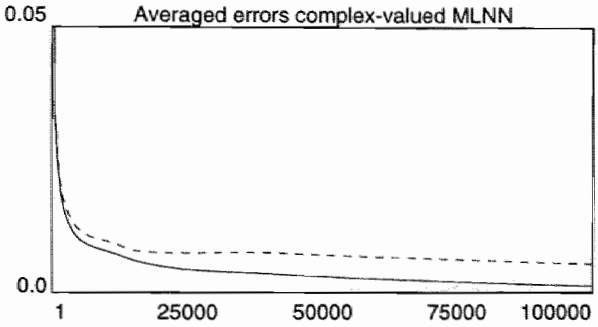


Figure 2.9: *Error curves of complex-valued Back Propagation adaption a 2-24-2 complex network over 100,000 epochs ($\eta = 0.01$, $\alpha = 0$). Solid line represents the error on the learning set; the dashed line represents the error on the test set.*

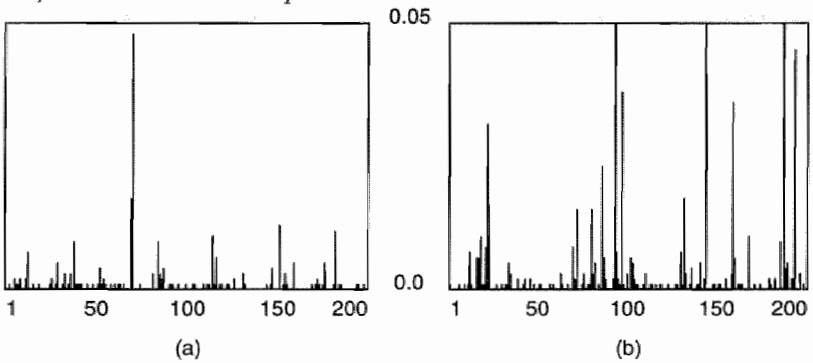


Figure 2.10: *Error spectra of the complex-valued 2-24-2 MLNN after 100,000 epochs ($\eta = 0.01$, $\alpha = 0$). A spectrum is displayed for the examples (a) in the learning set, and (b) in the test set.*

bottom 20 measurements reported in Table 2.1 yielding $\mu_1 = 0.0261$ with variance $\sigma_1 = 0.0009$ and sample size $n_1 = 20$. Similarly, mean μ_2 is calculated from the bottom 15 values of Table 2.2 yielding $\mu_2 = 0.0084$ with variance $\sigma_2 = 0.0019$ and sample size $n_2 = 15$. These particular ranges were chosen since they show little error decrease suggesting that the network error is almost stable. Using these values a t-test with 33 degrees

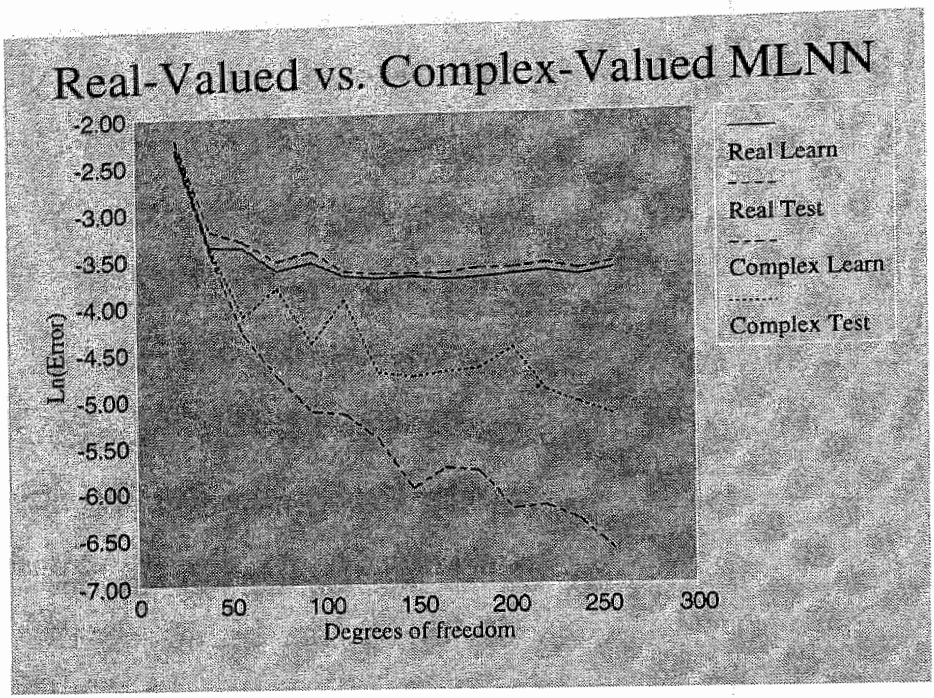


Figure 2.11: *Graph of the real-valued MLNN vs. the complex-valued MLNN error as a function of the number of degrees of freedom of the network.*

of freedom can be performed. Since we are dealing with a one-tailed test we may reject $\mathcal{H}_0$ with a significance of $\alpha = 0.001$ and accept $\mathcal{H}_1$ if the T-value is larger than $t_{0.001} = 3.385$. The calculated T-value is 37 and hence we may reject $\mathcal{H}_0$ and accept $\mathcal{H}_1$ meaning that the mean test error of the complex-valued MLNN is smaller than the mean test error of the real-valued MLNN.

2.5 Chapter Summary

In this chapter we have presented the functional architecture of complex-valued MLNNs. The connection weights and the neuron outputs in a complex-valued MLNN are represented by complex values. Hence, the weighted-input sum is a summation of complex multiplications and the neuron output function is a complex function. We have proven that a vectorized version of the generalized delta rule also applies to adaptation of complex weights in case of the neuron output function is a complex differentiable function.

In the real-valued MLNN, the neuron output function is bounded. In contrast, the analytic, i.e. differentiable, output function used in the complex MLNN is not bounded since Liouvillle's theorem states that all bounded functions that are everywhere differentiable must be constant. For example, the complex sigmoid function is not constant and must therefore be unbounded. A simple analysis of the complex sigmoid function indicates that it is singular at $ik\pi$ with k an odd integer value. Experiments showed that the singularities have a destructive impact on the numerical computer simulation unless a small learning rate is used. However, such a small learning rate slows down the learning process, making it less useful. Therefore, we have decided to use a non-analytic output function containing no singularities. A non-analytic function is not a direct function of the weighted input and is not complex differentiable. The proposed function is called a *complex squashing function*, as it squashes the complex *magnitude* of the input while letting its complex phase unchanged. If $\mathbf{Im}z = 0$ and $\mathbf{Re}z = x$ its behaviour is identical to the real-valued $\tanh(x)$ function, which is a variant of the sigmoid function used in real-valued MLNNs.

Using the complex squashing function, the generalized delta rule is no longer applicable because non-analytic functions are not complex differentiable. Hence, we have derived a generalized *complex delta* rule based on the partial derivatives of the complex squashing function. A complex-valued MLNN using the complex squashing function is compared to a real-valued MLNN using the $\tanh(x)$ function. In the experiment, the networks must learn to transform the angles of two direction sensors into two angles of a simple robot arm in such a way that its hand is placed at the focus point

of the direction sensors. In the examples, every angle ϕ is represented by its sine and cosine value, i.e. by $(\sin(\phi), \cos(\phi))$. When a complex MLNN is presented with a complex representation $e^{i\phi}$, the *dependence* between $\sin(\phi)$ and $\cos(\phi)$ is taken into account by the generalized complex delta rule. In contrast, a real-valued MLNN does not take into account this dependence when $\sin(\phi)$ and $\cos(\phi)$ are presented, since the generalized delta rule assumes network inputs and outputs are independent.

A learning and a test set were generated each containing 200 examples. Results of training complex and real MLNNs with these data show that in the case of at least the robot-arm experiment a complex-valued MLNN outperforms a real-valued MLNN with the same number of degrees of freedom. Moreover, in view of other experiments on the application of complex-valued MLNNs, e.g. Benvenuto *et al.* (1991), we expect that the complex-valued MLNN will also be useful for signal processing.

Chapter 3

Modular Self-Organizing Feature Maps

Systems containing a large number of components should be partitioned into functional modules so as to reduce their complexity. Without any doubt, the brain also consists of functional modules, e.g. vision, hearing, speech. Hence, it is quite natural to introduce a modular approach for ANNs as well (Fodor, 1983; Murre *et al.*, 1989). Some advantages of such an approach are a better understanding of the system and enhanced possibilities for implementation on existing parallel-processing hardware platforms. Moreover, the system may be adapted to specific input with less difficulty because of the *a priori* knowledge that can be embodied by the functional decomposition chosen. The research presented in this chapter is concentrated on the combination of multiple two-dimensional Self-Organizing Feature Maps (2D SOFMs) (Kohonen, 1982; Kohonen, 1984) into a modular system similar to the functional organization of the primary cortex of the brain.

The cortex of the brain has a strongly developed modular structure partitioned into three main modules corresponding to cortical zones (Kolb and Whishaw, 1990). In the *primary cortical zone* representations are formed of sensory experiences, e.g. the pressure received by a fingertip. Each type of representation is contained in a *cortical map* that may be thought of as a specific submodule of the primary cortical zone. These different rep-

resentations are combined in the *secondary cortical zone* (cf. Figure 1.6). Many different maps may be constructed from the primary information. For instance, the owl monkey has sixteen different retinotopic maps for interpreting visual information (Churchland, 1986). The mustached bat has tonotopic maps for the basilar membrane, echo-delays and Doppler effect (Suga, 1990). The barn owl combines a map of interaural delay and a map of interaural intensity difference into an auditory map of space (Knudsen *et al.*, 1987). Presumably, in the secondary auditory cortex of the human brain there are phonetic maps needed for speech recognition (Luria, 1973). The third cortical module is known as the *tertiary cortical zone.* This zone holds a large number of areas that are assumed to mediate a variety of cognitive functions (Luria, 1973; Goldman-Rakic, 1988).

In this chapter we focus on the representations formed in the primary cortical areas and on their modular organization within the primary cortical zone. This part of the cortex receives all sensory inputs and transforms them in such a way that they can be fused in the secondary and tertiary cortical zones. Both the reception and transformation requires that the primary cortex processes a large variety of signals with a considerable bandwidth, e.g. visual and audio information. For this reason we believe that the representation technique employed by the primary cortical areas has a great potential for designing new parallel data-processing techniques. There is evidence to show that primary cortical areas represent the state of a sensory input signal as a localized pattern of activity in the cortex (Penfield and Jasper, 1954; Hubel and Wiesel, 1963; Knudsen *et al.*, 1987). For each different input state a different location is activated. Due to the two-dimensional structure of the cortex the areas are actually *feature maps* representing a chart of possible input states. Moreover, the same evidence shows that two features mapped nearby on the same map represent similar input states. Hence, in addition to the feature representation the *topological organization* of the feature map also represents the *structure* of the input distribution.

In order to study the topological organization of feature maps in general we use the two-dimensional Self-Organizing Feature Map (SOFM) model as introduced by Kohonen (Kohonen, 1982; Kohonen, 1984). SOFMs can be used to generate feature maps from raw input data using a self-organizing principle. During self organization a SOFM will adapt its internal organiza-

tion so that a feature map emerges whose topological organization mirrors the structure of the input distribution. We have developed a number of techniques to visualize both the topological as well as the internal organization of a 2D SOFM. Using these techniques, we illustrate how a modular system containing multiple SOFMs may be configured for the representation of multiple input signals.

This chapter contains five sections. In Section 3.1 we describe the hierarchy of the cortical functional architecture (Luria, 1973). Within this hierarchy we focus on the primary cortical zone responsible for receiving information (Penfield and Jasper, 1954). The modular organization of the primary zone is illustrated by the existence of multiple representations for, among other things, somato sensory-input and motor functions. It appears that the main characteristics of these cortical areas can be modelled as a *computational map* (Knudsen *et al.*, 1987) because they compute a map representation for a collection of input signals. Subsequently, in Section 3.2, we introduce the Self-Organizing Feature Map (SOFM) (Kohonen, 1982; Kohonen, 1984). A two-dimensional SOFM is essentially a computational-map model. Moreover, the self-organizing principle underlying SOFMs provides a method for generating feature maps for arbitrary data distributions. Furthermore, when data distributions have many dimensions it is difficult to evaluate the structure of a feature map. In Section 3.3 we therefore present a number of new methods of visualizing the structure of two-dimensional SOFMs. In Section 3.4 we actually use these methods as a guide in designing a modular system containing several two-dimensional SOFMs to represent sensor data. Such a modular system *resembles* the representation of sensory information in the primary cortical areas. An essential part of our proposed approach is the *visualization* of correlations between sensors using a SOFM. We have used this approach to create a sensor-state feature map representing events occurring in 28 sensors at the same time. The data used for the experiments represent the state of a chemical process in a refinery at Dutch State Mines (DSM) over a period of five days. Finally, the main results of this chapter are summarized in Section 3.5.

3.1 Modularity in the Cortex

Brodmann's studies (Brodmann, 1909) have led to a detailed cytoarchitectonic map of the brain (cf. Figure 1.5). These early studies were possible since cytoarchitecture can be analyzed by visual inspection of dead brain tissue through a microscope. Brodmann's map shows that cortical brain tissue has a modular structure suggesting that each module has a different function.

In the 1950s the first detailed functional map of the human cortex was constructed by Penfield and his colleagues (Penfield and Rasmussen, 1950; Penfield and Jasper, 1954). Their experiments were performed on the brains of epileptic patients by electrically stimulating some cells in the cortex. By observing muscular reaction and experiences reported by the patient, they were able to label the points of stimulation. These labels form the functional map indicating that the cortex indeed has a modular organization. Moreover, it then became clear that even a *single* Brodmann area contains several functional modules.

Studies of the functional architecture of the cortex have led to a hierarchical system description of its operation introduced by Luria (1973). In this section we will outline this hierarchical system because it links the functional architecture of the cortex and the reception, analysis and storage of sensory input in the brain. In the context of this global description we then focus on the reception of information. Somehow the cortex successfully transforms a large variety of input signals into a mapped representation, enabling the integration of different signal types in subsequent stages. According to Knudsen *et al.* (1987), this transformation is performed by a *computational map* representing a map of features corresponding to signals of the same function, e.g. hearing. In this chapter we will concentrate entirely on the analysis of the representation formed in such a computational map. Because of its massively parallel operation it can be a valuable building block for real-time preprocessing of sensor data.

3.1.1 The Hierarchical System

The projection map of Figure 1.6 shows the primary, secondary and tertiary zones of the cortex. According to Luria (1973) these three kinds of area form a hierarchical system for receiving, analyzing and storing information. Comparing Brodmann's cytoarchitectonic map (cf. Figure 1.5) and the projection map of Figure 1.6 we see that the primary cortices are also distinguished by their cytoarchitecture. We recall that in Section 1.1.2 a specific relation between cortical cytoarchitecture and function was mentioned. The cytoarchitecture of the primary sensory cortices is marked by the large number of cells in layer IV which is therefore characterized as input layer. The primary motor cortices have predominantly neurons in layer V which is therefore called output layer. The primary areas are surrounded by secondary areas consisting mostly of association neurons located in layers II and III, hence, these are called association layers. If cells in the secondary zone are stimulated, more complex forms of sensation tend to appear.

Because of the associative nature of the secondary zones it is believed that the information *received* in the primary zones is analyzed in the secondary zones. Hence, the primary zones correspond to the lowest level in the hierarchical structure while the secondary zones are the first level upwards in the system hierarchy. However, the secondary zones are still modal. For instance, there is a secondary sensory cortex, secondary visual cortex and secondary auditory cortex. In the tertiary cortices the analyses of the various modalities are integrated. In human beings in particular the tertiary zones have developed to a considerable size. Placing such an integration in the broader context of mental activity, Luria (1973) says that "the tertiary zones play an essential role in the conversion of concrete perception into abstract thinking". Further, Luria distinguishes three basic laws that apply to the hierarchical working structure of the cortex in receiving, analyzing and storing information.

Law of hierarchical structure: This law states that the primary, secondary and tertiary cortical zones fit in a hierarchical working structure. That is, the primary zones receive information which is analyzed by the secondary zones. Subsequently, these analyses are integrated

in the tertiary zones.

Law of diminishing specificity: The representations in the primary zones are highly modal and are also topologically correct, i.e. the arrangement of sensors on the body is maintained by their mapping onto the cortex (Penfield and Jasper, 1954). The secondary zones are also modal but not topological as they combine the representations formed in the primary zones. Finally, the tertiary areas are even less specific, integrating different modalities.

Law of progressive lateralization of functions: The primary zones are simple representations of the sensory modalities, e.g. somatosensory, auditory, visual etc. Areas in the primary zones existing in the left and the right hemisphere of the brain are functionally equivalent. In the human brain there is an increasing *lateralization* of functions located in the secondary areas but even more so of those located in the tertiary areas. For instance, the function of speech in brains of right-handed people is often located in the dominant hemisphere, i.e. in the left part of the brain. As a result of the hierarchical structure such a lateralization is only effective in the secondary and tertiary zones.

The functional architecture of the cortex is not strictly hierarchical. Firstly, within the hierarchical functional architecture information may travel via multiple pathways. For instance, information in the visual primary cortex is processed by two major paths through the cortex (Mishkin *et al.*, 1983; Kolb and Whishaw, 1990). The first pathway is concerned with the *what* of visual information while the second pathway is concerned with the *where* of visual information (Goodale and Milner, 1992). Both pathways exist in parallel and hence the visual information is spreading into the higher zones rather than converging in them. Moreover, the information flow is not one-way; there are reciprocal connections that undoubtedly form some sort of feedback loop.

Secondly, new evidence shows that the secondary zones surrounding the primary ones also receive input projections identical to the ones received by the primary areas (Kolb and Whishaw, 1990). Hence, within the hierarchical working structure there is a parallel mode of input processing at

the lower levels. However, the input in the secondary zones is fed into layer I, rather than into layer IV. It is known that projections in layer I tend to terminate over a relatively wide area as they are located on the surface of the cortex. Consequently, the way input is processed in the secondary zones is almost certainly different from the way this input is processed in the primary zones.

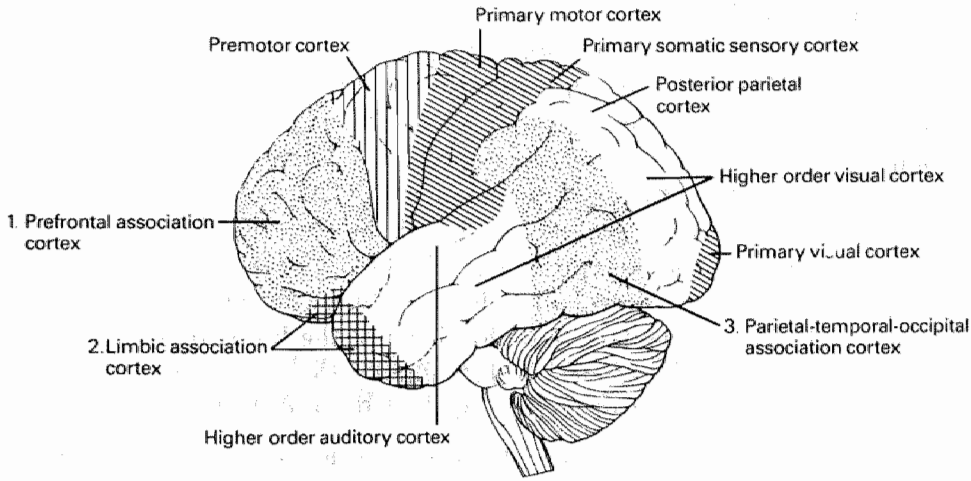


Figure 3.1: *Schematic drawing of the medial and lateral surface of the human brain showing the regions of the primary motor and sensory cortices, the higher order motor and sensory cortices, and the cortical association areas (from Kandel and Schwartz, 1982).*

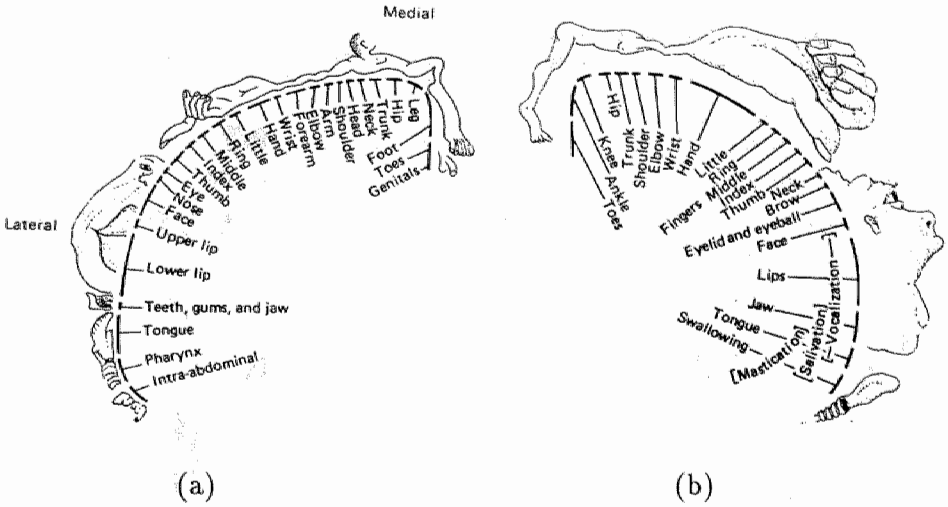
3.1.2 Receiving Information

The primary motor and somatosensory cortex provide a good illustration of the principle by which the brain receives information from the receptors in and on our body. The location of this part of the cortex has already been depicted in Figure 1.6 (Chapter 1), showing the projections from the motor and the somatosensory system. The primary motor and sensor cortex lie in the medial and lateral surface of the human brain surrounded by higher-order cortices (cf. Figure 3.1).

The cells of the primary motor and somatosensory cortex correspond to different parts of our body. Typically, upon stimulation of a cell in the sensory part, the patient would mention a kind of sensation in a leg, arm, face, etc. (Penfield and Jasper, 1954). When a cell in the motor area was stimulated, vocalization or general movement was observed. A map of these cortices can be prepared in many forms; one of the best known is the drawing of the *homunculus*. A homunculus is a figurine, i.e. a drawing of a body shape. Parts of the body are not well proportioned which is due to their different size of representation in the primary cortex. The homunculi, one for the somatosensory cortex and one for the motor cortex, are depicted in Figure 3.2(a) and (b) respectively. We note that the motor homunculus differs from the sensory homunculus. This can be explained by the fact that some parts of the body have little motor control while having a large number of sensory receptors, e.g. the lips. Conversely, some parts have considerable motor control while having almost no sensory receptors, e.g. the vocal chords. The fact that the shape of the body appears in the projection of input on the cortical surface implies that the topology of the input, e.g. the arrangement of sensors on the left hand, is preserved.

The homunculi drawn in Figure 3.2 are not accurate in the sense that they describe the cells in the cortex in detail. However, even the detailed maps drawn by Penfield and his coworkers are not very precise due to the fact that stimulation was done with large electrodes. Recent studies based on microelectrode stimulation have revealed that the somato-sensory area in the monkey is actually composed of three and perhaps four *smaller* representations of the body as illustrated in Figure 3.3 (Kaas *et al.*, 1979). Figure 3.3(a) shows the representation of the traditional homunculus found in the Monkey brain using large electrodes. Figure 3.3(b) depicts the major body parts which are represented as functional bands. Figure 3.3(c) shows the multiple representations which were discovered using microelectrodes. This discovery suggests strongly that the homunculus of the human brain (cf. Figure 3.2) is also composed of multiple representations, each one encoding some different aspect of somato-sensation (Kolb and Whishaw, 1990).

Beside the sensory and motor cortex Penfield and his coworkers also experimented on the autonomous system, e.g. respiration, the olfactory system (smelling), the speech system, the auditory system, the visual system,



(a) (b)

Figure 3.2: *The homunculus in the primary (a) somatosensory cortex and (b) motor cortex (from Penfield and Rasmussen (1950)).*

memory and the gustatory system (taste). For all those systems they found corresponding areas in the brain. So it may be safely concluded that the primary cortical zone has a clear functional organization. This organization may be interpreted as if each module encodes specific aspects of the input signals it receives in such a way that they may be integrated more easily in subsequent stages.

3.1.3 Computational Maps

Above we have indicated that the cortex contains a number of modules operating in a hierarchical working structure. Especially in the primary module, the most important task of the cortex is to represent a large variety of events so that they may be combined (fused) in secondary and tertiary stages. According to Knudsen *et al.* (1987) the primary areas should be called computational maps because they transform (compute) the input signal into a map-based representation. Also Durbin and Mitchison (1990)

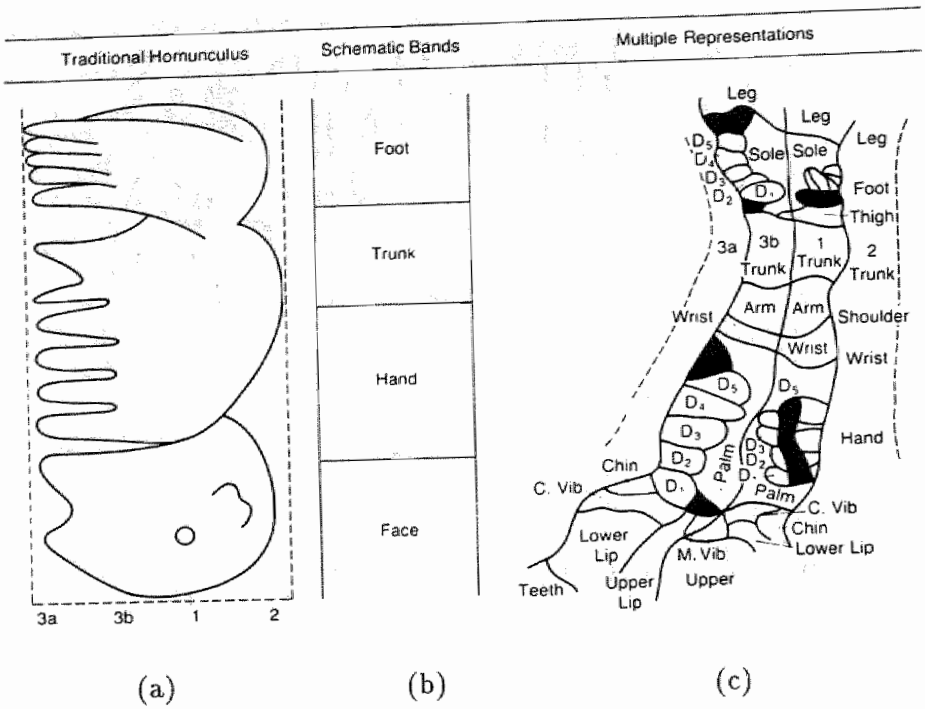


Figure 3.3: *Three conceptions of the organization of the somatosensory cortex in the monkey brain. (a) Representation of a traditional homunculus mapped with large electrodes. (b) Major body parts represented as functional bands. (c) Multiple representations of the body mapped with microelectrodes. (From Kaas et al. 1979).*

emphasize that the reduction of a large number of input sensors to a two-dimensional representation, e.g. a map, on the surface of the cortex is a necessary and valuable computation. For instance, in the auditory cortex of the barn owl two computational maps exist (Knudsen *et al.*, 1987). One primary map represents the interaural delay and a second one represents the interaural intensity difference. The neural activity pattern in these maps is combined in the secondary stage, enabling the barn owl to determine the

general orientation of a sound in space (relative to the ears). A signal that is presented to a computational map results in a cluster of activity at a position on the map. Hence, the computational map computes a *feature map* since it transforms a typical input state, i.e. a feature, by activating a location on the map. Moreover, a computational map represents similar inputs as activity clusters at similar locations on the map. Because of this additional property the computed feature map is said to be *topologically correct*. The feature map will play a central role in later sections.

Knudsen *et al.* (1987) mention four advantages of using computational maps in computing a feature-map representation. Firstly, information is processed rapidly because the computations are preset and are executed in parallel. Secondly, maps simplify the schemes of connectivity required for processing and using the information. Thirdly, a universal representation of the results of different kinds of computation allows the nervous system to employ a single strategy for reading and integrating this information. Fourthly, owing to its topographical structure a mapped representation benefits from several known classes of neuronal mechanisms, such as lateral interaction between neurons. For example, on receiving a *continuously* changing signal the mapped representation will show a gradually moving activity cluster. Lateral interaction between neurons and the topological representation of similar inputs enable the computational map to anticipate a transition of activity to a neighbouring cluster resulting in a quick response to a variation in the input.

Once the relevance of computational maps in computing efficient representations in the brain has been established, the next step is to model the computational map as an ANN so that their benefits can also be taken advantage of for parallel-processing tasks. One of the main questions is the origin of the computational maps in the cortex. A number of interesting mechanisms have been presented (Von der Malsburg and Willshaw, 1977; Kohonen, 1982), showing that computational maps may be the result of a self-organizing process. Using a self-organizing mechanism, the application of computational maps for generating topologically correct feature maps is virtually unlimited. In the remaining part of this chapter we use the Self-Organizing Feature Map (SOFM) (Kohonen, 1982) to analyze the representation of input signals in a system of two-dimensional feature maps that is similar to the representations in the primary areas in the cortex.

3.2 Kohonen Self-Organizing Feature Maps

The phenomenon of self-organized formation of topologically correct feature maps was introduced by Kohonen (1982). Kohonen discovered that the structure of any signal distribution can be captured in an n-dimensional array of processing units which did not correspond to this structure initially. The adaptation is based on self organization, which is why the feature maps are called Self-Organizing Feature Maps (SOFMs). SOFMs may be viewed as a generalization of the *computational maps* discussed in the previous section. However, computational maps in the cortex are likely to be determined genetically and are not formed by the self-organizing principle that applies to SOFMs. Yet, this does not exclude the possibility that basic neural structures are fine-tuned by a self-organizing mechanism based on sensory experiences after they were formed genetically. More importantly, such a mechanism would allow the formation of spatial maps for attributes and features, so it it may be used for the automatic formation of symbolic representations of concepts in such a way that it can be integrated with our primary senses.

In the introduction to the SOFM we first explain its relation to computational maps in the cortex. Then the functional architecture of a two-dimensional SOFM is presented explaining the connection architecture, the neuron-processing model and the activation dynamics of the system. For simulations a simplified algorithm may be used that does not require numerical integration of the differential equations underlying the neuron-processing model of a SOFM (Kohonen, 1982). This algorithm is explained and is used to demonstrate the formation of a feature map by self organization. Once a SOFM is organized, its essential operation is to reduce the dimension of the input similar to the reduction of sensory input to the two-dimensional surface of the cortex (Durbin and Mitchison, 1990).

3.2.1 A Topological Feature Map

A computational map is a neural network with a two-dimensional topology, i.e, the neurons are organized in a flat surface (cf. Figure 1.11 in Section 1.2.2). The neurons in the network receive input coded as a feature vector (cf. Equation 1.1 in Section 1.2.1). We recall that a feature

vector is a mathematical concept which stands for a collection of features each expressed by a numerical value as a vector component. For instance, a frequency spectrum is often used to represent sound. For simplicity we assume that all neurons in the map receive the same feature vector at the same time.

Input in a computational map results in a cluster of activated neurons while other neurons are not activated. Such a cluster may be visualized as a map location marked by a "bubble" of neural activity. The bubble may be sharpened by special neural mechanisms, for instance by lateral inhibition between neurons realizing a Winner-Takes-All architecture (cf. Section 1.2.2). Furthermore, a computational map represents *similar* feature vectors by activity clusters having their center of activity located on *neighbouring* map locations. This implies that the topology of the input distribution is preserved by the transformation in the computational map. The construction of such a transformation is realized automatically by the self-organizing process occurring in a SOFM. Generally speaking, the SOFM creates a *topological feature map* of the input distribution by preserving distances between the feature vectors it receives as input. If the input-distribution structure has more than two dimensions, the 2D SOFM attempts to create a feature map as accurate as possible. This problem resembles that of finding a method for creating a map of the earth. Ideally, the earth should be represented by a globe. However, it is much more convenient to carry a map of the earth in spite of the fact that it is not an exact projection. In the same way it is convenient to use mapped representations in the brain (Knudsen *et al.* (1987), cf. Section 3.1.3).

3.2.2 Functional Architecture

In Chapter 1 we introduced a framework for describing the functional architecture of ANN models. In this subsection the connection architecture, neuron-processing model and activation dynamics of the SOFM model are presented based on the introduction by Kohonen (1982). However, for simulating SOFMs a simplified computer algorithm is used. We feel that within the framework of this thesis a short description of the functional architecture should be presented. It may serve as a method for comparing a SOFM with other ANNs. Moreover, understanding the activation dynam-

ics of the neural system may help to understand the principles underlying the simplified algorithm that is presented later on in this section.

Connection Architecture

A SOFM is an ANN in which neurons are organized in an n-dimensional array. Most implementations in fact use a two-dimensional grid containing n neurons $\eta_1, \ldots, \eta_n$. Neuron η_a has output y_a and is located at network coordinates (a_x, a_y). The connection architecture specifies two types of inputs for each neuron: (1) input from outside the map, and (2) *lateral interaction* between neurons in the map. The lateral interaction is actually a combination of excitation and inhibition between neurons realizing a Winner-Takes-All (WTA) function (cf. Section 1.2.2) in which not one neuron but a cluster of neurons "wins".

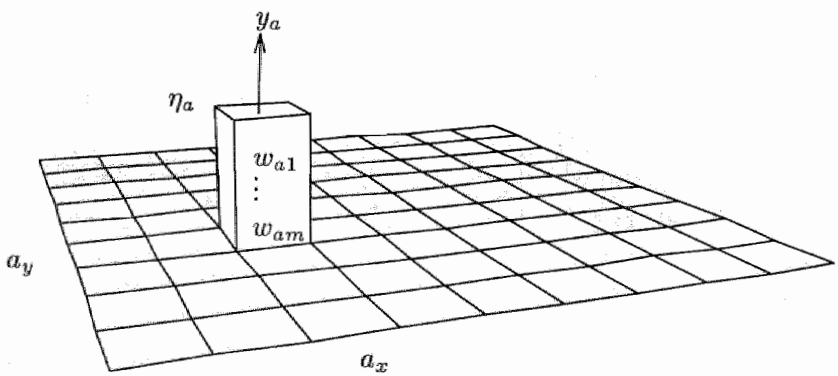


Figure 3.4: *A SOFM is a two-dimensional array of neurons. The square column at coordinates (a_x, a_y) represents neuron η_a.*

Figure 3.4 depicts a 9×9 SOFM. Neuron η_a is represented by the square column placed at grid coordinates (a_x, a_y).

Neuron Processing Model

The neuron-processing model of a SOFM corresponds to a dynamic system owing to the feedback connections existing between neurons in the form of lateral connections. The neurons in the SOFM receive *external* input in the same way as neurons in a Perceptron (cf. Chapter 1) or as neurons in the first layer of a MLNN (cf. Chapter 2). Neuron η_a has a m-dimensional weight vector $\mathbf{w_a} = w_{a1}, \ldots, w_{am}$ that is applied to the network input $\xi = \xi_1, \ldots, \xi_m$. Similar to the weights in the Perceptron and the MLNN, the weights $\mathbf{w_a} = w_{a1}, \ldots, w_{am}$ are variable. However, in contrast to neurons in a Perceptron or in a MLNN layer, neurons in the SOFM also receive input from each other through lateral connections. The weight of the lateral connection between neuron η_i and neuron η_j is denoted $A(i;j)$, i.e. the j-th element of the i-th row of the the network interconnectivity matrix $\mathbf{A}$. Matrix $\mathbf{A}$ is fixed in such a way that neighbouring neurons excite each other, i.e. have a positive weight, and neurons that are further away inhibit each other, i.e. having a connection with a negative weight. Moreover, the lateral connections are equivalent for each neuron and are symmetric. If the lateral interaction strength is displayed as a function of the distance, it resembles a "Mexican-hat" (cf. Figure 3.5) (Kohonen, 1982). The positive center area represents short-range lateral excitation. This area is surrounded by an area of inhibitory action which is again followed by an outer area of weaker excitatory action.

We use the three-stage processing model introduced in Section 1.2.2 (cf. Figure 1.12) to describe the dynamic neuron-processing model.

Stage I: In the first stage two different sums are calculated. Let $\sigma_a(t)$ denote the weighted sum of the inputs $\xi_1, \ldots, \xi_m$.

$$\sigma_a(t) = \sum_{k=1}^{m} w_{ak}\xi_k(t) \tag{3.1}$$

Let $\rho_a(t)$ denote the weighted sum corresponding to the feed-back from lateral connections.

$$\rho_a(t) = \sum_{k=1}^{n} A(k;a)y_k(t) \tag{3.2}$$

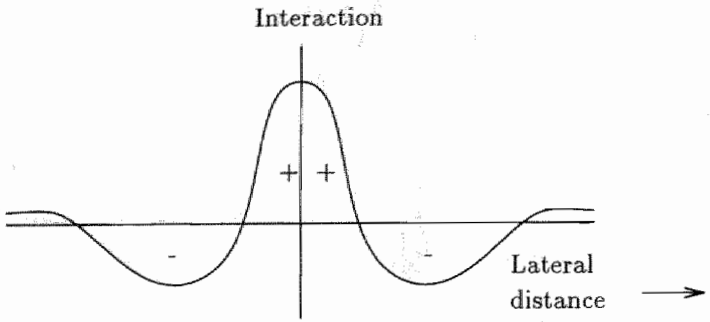


Figure 3.5: *The lateral interaction strength between neurons in a SOFM has the shape of a "Mexican-hat function".*

Stage II: In the second stage the internal state $x_a(t)$ of η_a is determined by summing σ_a and $\rho_a(t - \Delta t)$. Hence, the internal state of a neuron is determined by a difference equation if Δt approaches 0:

$$\lim_{\Delta t \downarrow 0} x_a(t) = \sigma_a(t) + \rho_a(t - \Delta t) \tag{3.3}$$

Stage III: Finally, the output $y_a(t)$ of neuron η_a is a non-linear function $f()$ of $x_a(t)$. This function may, for instance, be a piece-wise linear sigmoid-like function :

$$y_a(t) = \begin{cases} 0 & x_a(t) < 0 \\ x_a(t) & 0 \leq x_a(t) \leq 1 \\ 1 & x_a(t) > 1 \end{cases} \tag{3.4}$$

Activation Dynamics

The neuron-processing model of a SOFM constitutes a dynamic system since the non-linear, first-order differential equation in the second stage receives lateral feed-back. Kohonen (1982) shows that the network-activation pattern develops a cluster of activity, i.e. an activity *bubble*, centered

around that part of the network having the largest response to the input. The "Mexican-hat"-shaped lateral interaction (cf. Figure 3.5) inhibits neurons surrounding this cluster even if they respond to the input. Hence, the boundary of a cluster is sharpened by lateral inhibition. Thus, the state of a SOFM is subject to a dynamic system with an attracting equilibrium state determined by the network properties and the input. As soon as new input is presented to the network a new stable state is entered. The time interval required for responding to new input is much smaller than the time interval between different inputs. Hence, when a stable cluster emerges it represents the *static* input $\xi_1, \ldots, \xi_m$. Essentially, this representation is determined by the projection of the input onto the neurons by their weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$. The activation dynamics of the SOFM may be *simulated* by applying the input to all neurons and selecting the neuron which has the highest output.

3.2.3 Self Organization

The main objective of Kohonen's early work on SOFMs was to demonstrate that external signal activity alone is sufficient for enforcing maps of patterns relating to an abstract representation of an arbitrary feature space (Kohonen, 1982). Essentially, this means that a SOFM should organize *itself*. Hence, the SOFMs are *unsupervised* networks while, for instance, MLNNs are *supervised* networks.

The self organization of SOFMs occurs by adaptation of the weight vectors using a Hebbian-type learning rule. Hence, weight w_{ij} of neuron η_i corresponding to input component ξ_j is adapted as follows :

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \xi_j y_i(t) \tag{3.5}$$

We note that for the adaptation of $w_{ij}(t)$ no target is required in Equation 3.5. Hence, it is an *unsupervised* learning rule. In addition to the learning rule the weight vector of a single neuron must have unit length, i.e. $\sum_{j=1}^{m} w_{ij}^2 = 1$. If this constraint is not added, weights may grow infinitely according to Equation 3.5. The restriction may be combined with Equation 3.5 into an adaptation rule that automatically normalizes the

weight vectors after they have been adapted (Kohonen, 1982).

$$w_{ij}(t+1) = \frac{w_{ij}(t) + \alpha \xi_j y_i(t)}{\|\mathbf{w}_i + \alpha y_i(t)\xi\|} \tag{3.6}$$

In a SOFM consisting of only one neuron, this adaptation rule would adapt (the only) weight vector $\mathbf{w}$ to the normalized vector corresponding to the *average* direction, i.e. a feature of the inputs ξ. However, in a SOFM with more than one neuron, not all weight vectors are adapted equally since its activation dynamics forms activity bubbles. Due to the presence of $y_i(t)$ in Equation 3.6 only weight vectors of neurons located in a bubble are adapted. Hence, the weight vectors in the SOFM become differentiated in such a way that not only one average direction of the input is represented but several characteristic features will be represented. It is easy to see that this self-organizing process tends to reinforce the differentiation because the larger $y_i(t)$ is for a particular input ξ, the more its weight vector $\mathbf{w}_i$ will be tuned to ξ and the larger $y_i(t')$ will be the next time $t' > t$ the same input occurs.

Owing to the adaptation of a cluster of neighbouring neurons a smooth ordering is forced to emerge. The smoothness expresses the *topography* of the input signals. This means that neighbouring neurons on the map are sensitive to similar input signals. This ordering follows directly from the adaptation and the activation dynamics described above.

3.2.4 Similarity Matching

We are more interested in understanding the computational map as a parallel computing method than in understanding its exact dynamics. More precisely, our main interest lies in the representation of information by using feature maps. In the primary cortex such feature maps are implemented by computational maps. Other, non-neural implementations are possible as well. As Kohonen (1982) points out, the logic underlying the neural model of SOFMs does not necessarily have to be restricted to neural mechanisms. Moreover, it appears that a simple algorithm operating on an n-dimensional array of data elements is capable of forming feature maps. This algorithm is based on a procedure called *similarity matching* and is sometimes referred

to as the *short-cut algorithm*. The computational complexity of this algorithm is low compared to the computational complexity of an algorithm for numerical integration. For this reason, we will use the similarity-matching procedure rather than simulating the actual SOFM processing model.

The lateral connection architecture and the neuron-processing model of a SOFM are intended for the formation of activity clusters on the map. Only neurons in these clusters are adapted. When the purpose of simulating a SOFM is to obtain a topologically-ordered feature map, it is not necessary to integrate the system of non-linear differential equations responsible for the formation of activity clusters. Another possibility is to find the neuron that is most similar to the current input and adapt the weight vector of this neuron and its neighbours in the network. The similarity may be measured as the inner product of the input vector and the weight vector. We note that this would be equal to the neuron output if the lateral interaction and the non-linear output function were eliminated. In this interpretation each neuron is a similarity matching device which measures the similarity between the input and its own weight vector.

If the weight vectors and the input vectors are normalized, the similarity may also be measured as the Euclidean distance between $\mathbf{w}$ and ξ, i.e. $\|\mathbf{w} - \xi\|$. In that case the value 0 denotes maximum similarity. For the similarity matching algorithm the following static neuron-processing model defines the output of neuron η_a as $\|\mathbf{w}_a - \xi\|^2$:

$$y_a = \|\mathbf{w}_a - \xi\|^2 = \sum_{i=1}^{m} (w_{ai} - \xi_i)^2 \tag{3.7}$$

Based on this processing model each neuron can be tuned to a particular input vector ξ. We note that the Euclidean similarity measure used in Equation 3.7 does not require that the weights are normalized. The similarity-matching approach has replaced the differential equations by a sequential algorithm finding the best-matching unit (bmu). Neuron η_a is bmu if for all $b \in [1, n] : y_a \leq y_b$. If there are multiple winners, we choose the neuron with the lowest index. Let h_{ab} be the neighbourhood function. This function can either be a characteristic function, i.e. $h_{ab} = 1$ if η_b is inside the neighbourhood of η_a and 0 otherwise, or it can specify a

bubble-shaped function, e.g.

$$h_{ab} = \begin{cases} 2^{-\|a-b\|_N} & \|a-b\|_N \leq \beta \\ 0 & \|a-b\|_N > \beta \end{cases} \tag{3.8}$$

The parameter β is called *neighbourhood size* because it defines the range in which neighbouring neurons of the bmu are adapted. The measure $\|a-b\|_N$ denotes the *neighbourhood distance* between a and b, i.e. let $a = (a_x, a_y)$ and $b = (b_x, b_y)$ then $\|a-b\|_N = MAX(|a_x - b_x|, |a_y - b_y|)$. If η_a is the winning neuron, the adaptation rule for the neurons $b \in [1, n]$ is

$$\Delta \mathbf{w_b} = \alpha h_{ab}(\xi - \mathbf{w_b}) \tag{3.9}$$

Here α is the learning rate (or gain) determining the speed of the adaptation process. Both α and β should decrease with the time in order to obtain a stable map configuration. We note that the neighbourhood function h_{ab} has replaced the clustering that otherwise would have emerged from lateral interaction. Furthermore, the normalization that was introduced in Equation 3.6 has been eliminated. This is possible because now similarity is measured as the Euclidean length of $\|\xi - \mathbf{w}\|$. Initially, normalization was necessary since the weight adaptation rule presented in Equation 3.6 could lead to infinite weight values without this constraint. In contrast, weights adapted according to Equation 3.9 are bounded to the extreme input values which are supposed to be finite.

We conclude this section with two examples that illustrate the operation of a SOFM. Using the Euclidean distance measure, one might present input corresponding to points randomly drawn from the unit square in R^2, i.e. $(\xi_1, \xi_2) \in [0,1] \times [0,1]$ to a two-dimensional SOFM with only two inputs. The self organization is visualized by interpreting the two weights of a neuron as a pair of coordinates referring to a *weight point* in R^2. Subsequently, straight lines are drawn between two weight points corresponding to neurons that are either left, right, upper or lower neighbours in the network. For instance, if neuron $(1,1)$ has weight point $(0.4, 0.5)$ and neuron $(2,1)$ has weight point $(0.1, 0.3)$ then a straight line is drawn between $(0.4, 0.5)$ and $(0.1, 0.3)$. An animation of the self organization in this SOFM is depicted

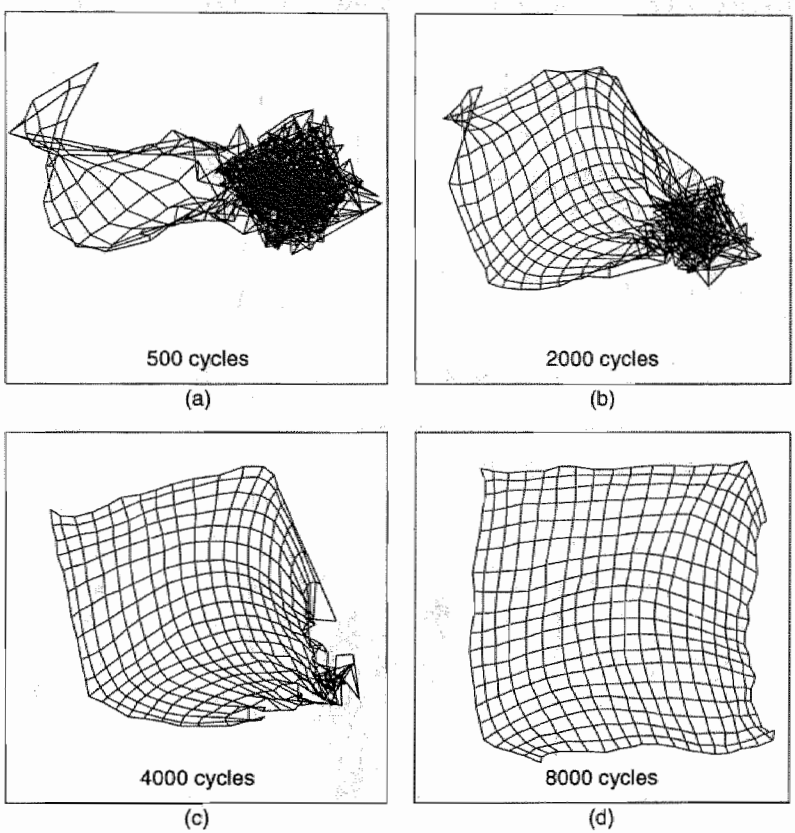


Figure 3.6: *Animation of the self-organizing process during which the random map unfolds itself into a regular grid representing the unit square in* R^2.

in Figure 3.6. In Figure 3.6(a) many lines are crossed because initially (at 0 cycles) all weights were randomized. When the self-organizing process adapts the weights, the network becomes ordered. The process starts in a small part of the network and slowly expands to all other neurons. Finally, in Figure 3.6(d) the weight-point distribution is ordered. Hence, we may conclude that neighbouring neurons in the network represent similar input

vectors since their weight vectors are close together.

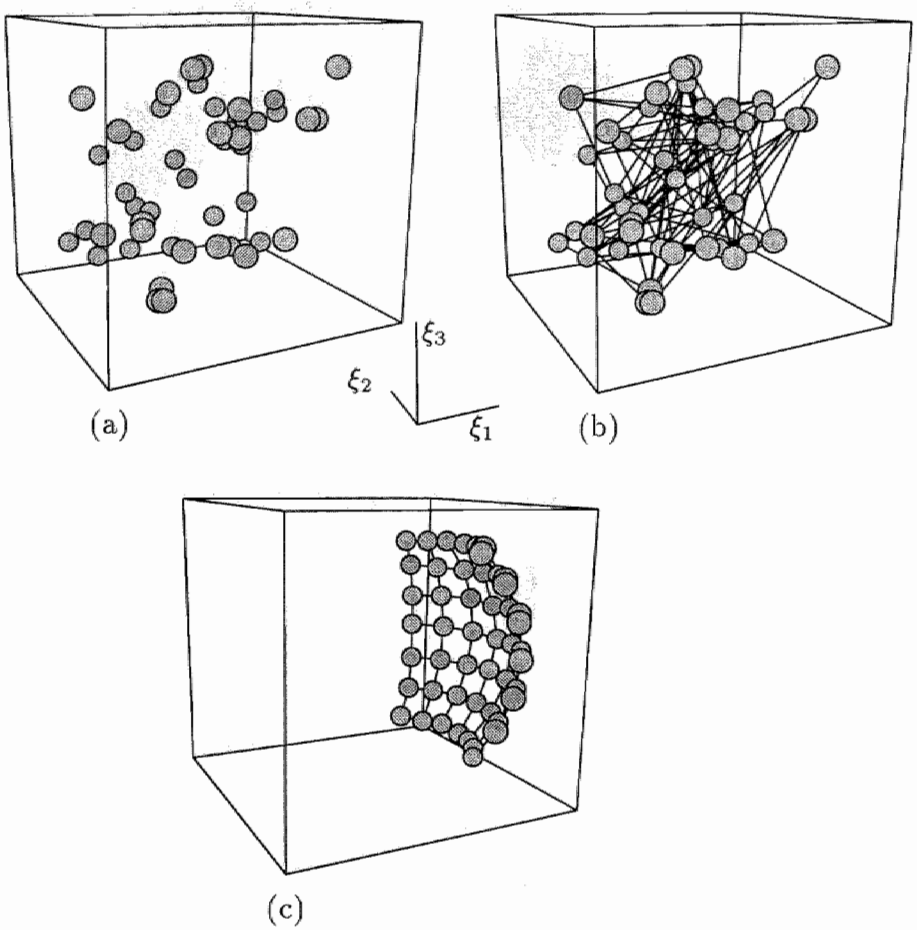


Figure 3.7: *Visualization of a two-dimensional SOFM with input dimension three drawn from the unit sphere in R^3 showing (a) the initial weight points, (b) straight lines between weight points of neighbouring neurons, and (c) the SOFM after it has been organized.*

In the second example the input distribution is part of the unit sphere in

R^3 from which randomly drawn points are presented to a two-dimensional SOFM. Consequently, the input dimension of the network is three and each neuron has three weights. When the size of the input distribution is relatively small compared to the surface of the unit sphere, e.g. size of the north pole compared to the earth globe, the distances between the distribution points in R^3 are almost the same as if the surface part was a rectangular surface in R^2. Again, the self organization may be visualized by interpreting the weight vector of a neuron as a weight point. However, in this example neurons have three weights and thus their weight point is located in three-dimensional space (cf. Figure 3.7(a)). Connecting weight points in the same way as in the previous example results in a random set of lines (cf. Figure 3.7(b)). After self organization the grid lines have unfolded themselves into a regular grid. In Figure 3.7(c) the organization is depicted for a situation in which the input distribution is located on a small part of the unit sphere.

Finally, we mention that the organization of a SOFM with an input dimension higher than three may not be so simple to visualize. The main problem is to calculate a projection matrix in such a way that the two-dimensional organization of the weights in R^m is visualized. In the previous examples such a projection could be selected manually (cf. Figures 3.6 and 3.7). However, if there are, say 100 inputs, this is no longer feasible. We address this problem in the next section and present an algorithm for the visualization of SOFMs with an arbitrary input dimension.

3.3 Visualization of SOFMs

A two-dimensional SOFM creates a feature map of the input vectors it has received during self organization. The map is actually a knowledge representation that is organized such that it may be interpreted easily by human beings who are used to reading maps. In Section 3.3.1 this property is underlined by an ideal example illustrating the expressive power of mapped representations. Usually, SOFMs are represented by charts on which neurons are labelled with references to specific input classes. However, such a labelled representation does not visualize the underlying weight organization and the excellence of the map cannot be assessed. In this section we

present two measures for excellence: (1) map tension, and (2) the neuron depth derived from a weight projection.

Firstly, *map tension* is discussed in Section 3.3.2. The tension of a neuron represents how well its weight vector represents the best matching example. Typically, graphic visualization of map tension will show cracks running over the SOFM surface when the input distribution is not strictly two dimensional. These cracks may be used for determining the boundaries of event classes characterizing the input.

Secondly, a method for visualising the SOFM excellence is a projection of the weights to three main components in the weight-point space. This method is more fundamental than the first one because it is based entirely on the contents of the map and does not require any recollection of previously learned examples. In Figure 3.7 the main components are automatically set to ξ_1, ξ_2 and ξ_3 eliminating the need for a projection operator since all weight points are located in R^3. In Section 3.3.3 we present an algorithm for deriving an appropriate projection operator in case the SOFM has more than three input dimensions. The algorithm generates two orthonormal projection vectors in the weight-point space representing the average two-dimensional weight-point plane. A third orthonormal projection vector is generated that may be used to visualize deviations from this plane. In order to visualize the resulting weight projection two techniques are presented. The first technique generates a two-dimensional view of the three-dimensional structure from an *interactively* chosen viewpoint. The second technique generates a depth chart of the SOFM by visualizing the projection of the weight vectors on the third projection vector as a grey color.

3.3.1 Visualization of the Feature Map

In order to appreciate the feature map representation provided by a two-dimensional SOFM we will perform an experiment that has been described by Weijters (1991). Consider the table of distances (by road) between 24 major cities in the Netherlands presented in Table 3.1.

A distance table is most useful as a look-up table, i.e. for determining the distance between two cities. If a more qualitative interpretation of the table

	Alkmaar	Amersfoort	Amsterdam	Apeldoorn	Arnhem	Assen	Deventer	Dordrecht	Eindhoven	Enschede	's-Gravenhage	Groningen	Haarlem	's-Hertogenbosch	Hilversum	Leeuwarden	Maastricht	Middelburg	Nijmegen	Roermond	Rotterdam	Tilburg	Utrecht	Zwolle
Alkmaar	.	93	39	132	141	181	150	125	163	203	75	175	33	128	75	112	252	212	163	217	101	156	80	159
Amersfoort	93	.	47	42	48	135	60	80	109	113	86	160	66	74	18	158	186	189	70	157	79	102	20	69
Amsterdam	39	47	.	86	98	179	106	96	119	157	56	198	19	85	32	135	208	183	120	173	72	113	37	113
Apeldoorn	132	42	86	.	25	108	16	128	104	72	130	133	107	90	59	131	178	237	43	130	123	115	64	42
Arnhem	141	48	98	25	.	133	38	103	79	87	119	161	114	65	67	160	153	212	18	105	117	90	61	71
Assen	181	135	179	108	133	.	95	217	212	115	223	26	200	198	152	66	286	326	151	238	216	223	157	67
Deventer	150	60	106	16	38	95	.	141	117	57	148	122	125	103	77	126	191	250	56	143	141	128	82	38
Dordrecht	125	80	96	128	103	217	141	.	90	189	46	242	92	64	80	226	179	110	99	144	23	59	59	151
Eindhoven	163	109	119	104	79	212	117	90	.	165	132	240	136	32	109	239	97	155	61	52	109	34	88	150
Enschede	203	113	157	72	87	115	57	189	165	.	201	142	178	151	130	157	239	304	104	191	194	176	135	68
's-Gravenhage	75	86	56	130	119	223	148	46	132	201	.	248	42	106	74	187	221	134	141	186	22	101	65	157
Groningen	175	160	198	133	161	26	122	242	240	142	248	.	206	226	177	57	313	351	179	265	241	251	182	92
Haarlem	33	66	19	107	114	200	125	92	136	178	42	206	.	101	48	143	225	179	136	190	68	129	53	134
's-Hertogenbosch	128	74	85	90	65	198	103	64	32	151	106	226	101	.	74	225	121	146	47	86	78	25	53	136
Hilversum	75	18	32	59	67	152	77	80	109	130	74	177	48	74	.	149	198	189	89	163	79	102	20	86
Leeuwarden	112	158	135	131	160	66	126	226	239	157	187	57	143	225	149	.	310	322	178	265	211	250	180	90
Maastricht	252	186	208	178	153	286	191	179	87	239	221	313	225	121	198	310	.	244	135	48	198	123	177	224
Middelburg	212	189	183	237	212	326	250	110	155	304	134	351	179	146	189	322	244	.	193	209	102	124	168	260
Nijmegen	163	70	120	43	18	151	56	99	61	104	141	179	136	47	89	178	135	193	.	87	113	72	83	89
Roermond	217	157	173	130	105	238	143	144	52	191	186	265	190	86	163	265	48	209	87	.	163	88	142	176
Rotterdam	101	79	72	123	117	216	141	23	109	194	22	241	68	78	70	211	198	102	113	163	.	78	58	150
Tilburg	156	102	113	115	90	223	128	59	34	176	101	251	129	25	102	250	123	124	72	88	78	.	81	161
Utrecht	80	20	37	64	61	157	82	59	88	135	65	182	53	53	20	180	177	168	83	142	58	81	.	91
Zwolle	159	69	113	42	71	67	38	151	150	68	157	92	134	136	86	90	224	260	89	176	150	161	91	.

Table 3.1: *Table of road distances between 24 major cities in the Netherlands.*

is desired, it turns out to be difficult to derive the topology of the cities in a geographic map of the Netherlands. However, such a map is a perfect tool for visualizing the numbers presented in Table 3.1. For those not familiar with the geographic map of the Netherlands we have outlined the map in Figure 3.8 and marked the positions of the cities mentioned in Table 3.1.

Figure 3.8: *Geographic map of the Netherlands used in the experiment.*

The cities in Table 3.1 are represented by vectors in R^{24}, for instance, Amsterdam is represented by the third row (or column) in the distance table. Hence, we can specify 24 input vectors that may be presented to a

SOFM. The SOFM will try to preserve the topology underlying these 24 vectors in their representation on the map. Since the underlying structure is the geographic map we expect that the feature map resembles Figure 3.8. Using a random selection from the 24 input vectors we have depicted two feature maps resulting from a 20×20 SOFM after 10,000 cycles ($\alpha = 0.2$ and $\beta = 10$) using different initializations in Figure 3.9.

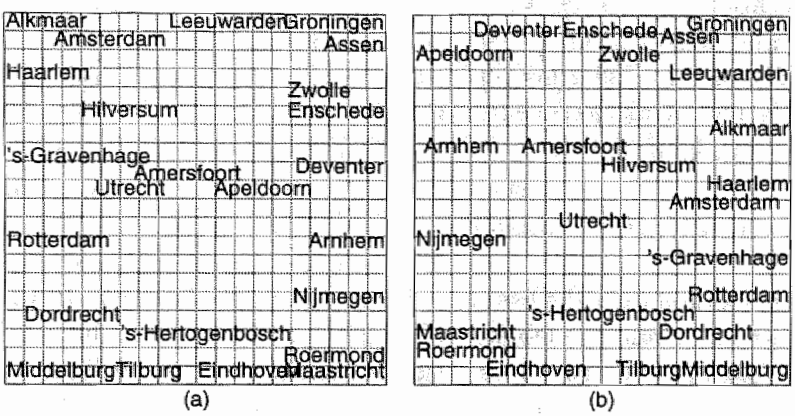


Figure 3.9: *Visualization of a labelled feature map representing locations of 24 Dutch cities. (a) best obtained result corresponding almost exactly to the geographic map, (b) other result indicating that depending on the random initialization also another stable mapping may emerge.*

Despite the fact that Table 3.1 contains distances by road, there is an unmistakable correspondence between the geographic map (cf. Figure 3.8) and the feature maps (cf. Figure 3.9). This experiment has been repeated several times with different random initializations; each time rotated but correct feature maps were found. However, if the neighbourhood-size set was too small, long-range relations between cities were found to be distorted. A typical distortion was, for instance, mapping the north part from left to right while mapping the south part from right to left. We note that the feature map may have a different orientation because the map orientation is not specified by the distance table (cf. Figure 3.9(b)).

The visualization of the city maps depicted in Figure 3.9 is a simple method for interpreting the organization of the SOFM. The map is constructed by labelling neurons with the name of the city for whose input vector they are the best matching unit. Let $\xi^{(1)}, \ldots, \xi^{(24)}$ denote the input vectors corresponding to the cities in Table 3.1, respectively. Furthermore, we introduce a label l_k for each input vector $\xi^{(k)}$, e.g. l_5 ="Arnhem". Let us assume example $\xi^{(k)}$ is presented to the network and let $y_a(k)$ be the output of η_a on input $\xi^{(k)}$, i.e. $y_a(k) = \|\mathbf{w}_a - \xi^{(k)}\|$. We recall that neuron η_a is the best matching unit for $\xi^{(k)}$ in case its output $y_a(k)$ (cf. Equation 3.7) is less than or equal to all other outputs $\forall b : y_b$ in the network. Subsequently, label l_k is placed at map coordinates (a_x, a_y). Essentially, the labelling operation can be described by the following proposition:

$$\forall k \in [1, 24] : \{l_k \leftarrow (a_x, a_y) | \forall b \in [1, n] : y_a(k) \leq y_b(k)\} \tag{3.10}$$

Using this method only the best matching neurons are labelled. In this case only 24 map locations are labelled while 376 locations remain blank. When an interpretation of *every* map location is desired another labelling strategy should be used. For instance, labelling each map location as the city for which it is most sensitive *regardless* of the fact if it is a best matching unit. The label for map location (a_x, a_y) is l_k if $y_a(k)$ is less than or equal to all other $\forall l : y_a(l)$. Similar to Proposition 3.10 this labelling strategy can be formalized as

$$\forall a \in [1, n] : \{l_k \leftarrow (a_x, a_y) | \forall l \in [1, 24] : y_a(k) \leq y_a(l)\} \tag{3.11}$$

The first labelling strategy is useful for labelling single locations on the map. In contrast, the second strategy may be used to determine *areas* on the map. As an illustration of the second labelling strategy we have substituted the city names by the name of the province the city is located in. The resulting feature map is presented in Figure 3.10 corresponding to the feature map depicted in Figure 3.9(a).

When a particular input vector is presented to the organized city feature map not only the best matching unit is activated but also neurons in its neighbourhood. In the labelled feature map only the location of maximum similarity is shown. If the activity bubble has a clear peak at this location

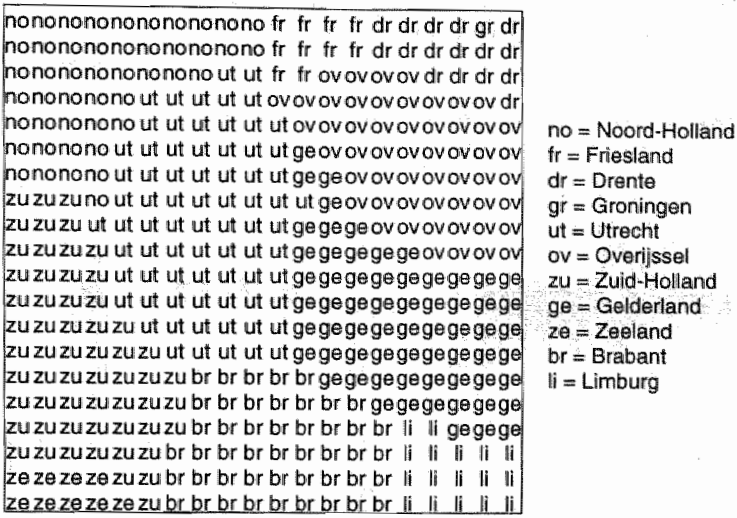


Figure 3.10: *Map of the provinces in the Netherlands.*

such a representation is sufficient. However, in case there exists no clear peak of activity, for instance, when the activity bubble is shaped as an activity plateau or ridge, the placing of a label may prove to be confusing. Therefore it may be useful to visualize the activity pattern as a *bubble-shaped* surface pattern. For this purpose we redefine the activity function (cf. Equation 3.7) as follows:

$$y_a = f(\xi) = e^{-\|\mathbf{w}_a - \xi\|^2/\sigma} \tag{3.12}$$

In this case $y_a = 1$ corresponds to maximum similarity between $\mathbf{w}_a$ and ξ, i.e. $\|\mathbf{w}_a - \xi\| = 0$. We emphasize that this activity function is used for visualization purposes only and *not* during self organization. When similarity decreases the output will decrease as a Gaussian function of the similarity. Hence, in an ordered SOFM input ξ will activate a cluster of neurons since the adaptation process places similar neurons together. The activity pattern is Gaussian, e.g. bubble, shaped and the slope of the bubble is determined by the distribution of the weight vectors in the network

and the parameter σ. In Figure 3.11 the activity pattern for Amsterdam, Groningen, Maastricht and Rotterdam is visualized.

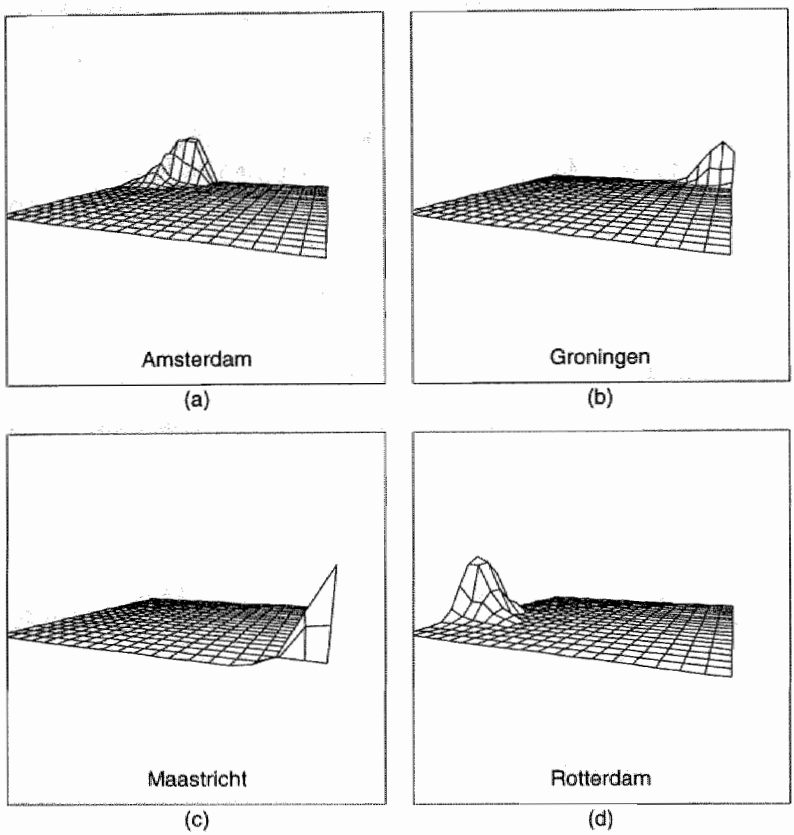


Figure 3.11: *Visualization of output activity. (a) Amsterdam, (b) Groningen, (c) Maastricht and (d) Rotterdam.*

We note that all activity patterns displayed in Figure 3.11 show clear peaks and hence there is no reason to question the location of the cities in Figure 3.9(a). If an activity pattern shows a large plateau of equal activities this indicates that for the presented input the SOFM is not very selective. Often this occurs when a certain input has a high probability or when the structure of the input distribution has less than two dimensions. If the

activity pattern shows several peaks this indicates that the SOFM has difficulties in finding a suitable two-dimensional representation for the input distribution.

3.3.2 Visualization of the Map Tension

The various feature maps shown for the city example do not indicate how well a neuron matches a particular example. Using the output function defined in Equation 3.7 we know that $y_a \geq 0$ with $y_a = 0$ equal to maximum similarity and thus corresponding to the minimal error. As long as $y_a > 0$ neuron η_a experiences *tension* owing to the adaptation process. The tension t_a of neuron η_a is defined as y_a/m where y_a is the smallest possible output for η_a and m is the input dimension. Output y_a is divided by m to enable comparisons between maps having different input dimensions. A map of tension values for all neurons may be calculated in much the same way as the second labelling method described in Equation 3.11. Let $y_a^{(k)}$ denote the output of neuron η_a on input $\xi^{(k)}$, then t_a may be calculated as follows.

$$\forall a \in [1,n] : \{t_a = y_a^{(k)}/m | \forall l \in [1,24] : y_a^{(k)} \leq y_a^{(l)}\} \tag{3.13}$$

The resulting *tension map* corresponding to the labelled feature map depicted in Figure 3.10 is shown in Figure 3.12.

We observe that tension is restricted to a few number of areas in the feature map. In general, there are two reasons why tension in a map occurs. Firstly, in this case, tension arises owing to a lack of input examples. The city SOFM has interpolated such that the tension areas in Figure 3.12 represent parts of the Netherlands in which cities are located that are not mentioned in Table 3.1. We observe that interpolation may fail if the input distribution allows more than one approximate two-dimensional arrangement of features. These areas have been generated by the self organization to create a consistent two-dimensional representation of the input data. Since there are no cities for labelling these locations their tension is high, finally resulting in the lines visible in Figure 3.12.

Secondly, tension may result from a competition between two types of input representing a multi-dimensional structure exceeding the two-dimensional

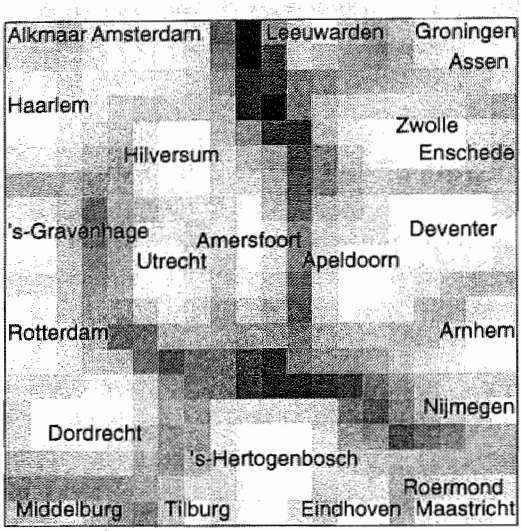


Figure 3.12: *Visualization of tension in a feature map.*

representation capacity of a 2D SOFM. Typically, this kind of tension is displayed as sharp contours on the tension map. It is not displayed in Figure 3.12 because the underlying topology of the input vectors is mainly two dimensional. To illustrate the second type of tension we generate a data distribution represented by a linear space $\mathcal{L} \in R^9$ constructed from three orthogonal planes $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$. Each of these planes is a square two-dimensional distribution described by the following vector equations.

$$\mathcal{V}_1 = \{\xi | \xi = \mathbf{p}_1 + \lambda_1 \mathbf{p}_2 + \mu_1 \mathbf{p}_3; \lambda_1, \mu_1 \in [0,1]\} \tag{3.14}$$

$$\mathcal{V}_2 = \{\xi | \xi = \mathbf{p}_4 + \lambda_2 \mathbf{p}_5 + \mu_2 \mathbf{p}_6; \lambda_2, \mu_2 \in [0,1]\} \tag{3.15}$$

$$\mathcal{V}_3 = \{\xi | \xi = \mathbf{p}_7 + \lambda_3 \mathbf{p}_8 + \mu_3 \mathbf{p}_9; \lambda_3, \mu_3 \in [0,1]\} \tag{3.16}$$

The vectors $\mathbf{p}_1, \ldots, \mathbf{p}_9$ are equal to the nine unit vectors in R^9 as described in Table 3.2.

R^9								
$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_3$	$\mathbf{p}_4$	$\mathbf{p}_5$	$\mathbf{p}_6$	$\mathbf{p}_7$	$\mathbf{p}_8$	$\mathbf{p}_9$
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1

Table 3.2: *Unit vectors in R^9 used to construct $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ which lie in $\mathcal{L} \in R^9$.*

We have drawn 100 random vectors from $\mathcal{L}$ labelling each example either as $\mathcal{V}_1$, $\mathcal{V}_2$ or as $\mathcal{V}_3$ depending on its position in $\mathcal{L}$. In Figure 3.13(a) the resulting feature map is presented showing that the SOFM has correctly clustered the three classes in the input distribution. However, the tension map presented in Figure 3.13(b) shows that cracks in the feature map exist indicated by the dark lines in the tension map. We note that in this example the tension is much sharper than the diffuse tension areas in Figure 3.12. In this case tension is caused by competition between three classes that cannot in anyway be represented in two dimensions.

We would like to draw attention to the robustness of the 2D SOFM in the last example. In spite of the fact that the input distribution is actually a six dimensional one, a correct two-dimensional feature map is produced. A pure linear projection of the data vectors would have failed to produce a similar result.

We emphasize that the tension map only visualizes the clustering structure of the SOFM indicating which neurons are likely to represent features and which are not. However, we would also like to know how accurately the

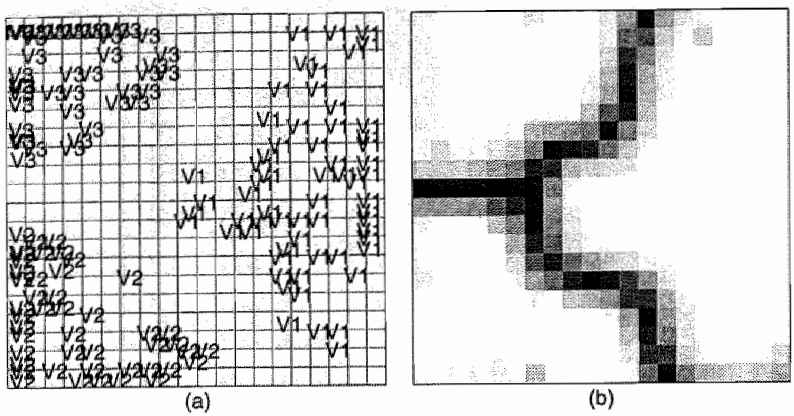


Figure 3.13: *Visualization of a two-dimensional SOFM with input distribution $\mathcal{L}$. (a) Feature map showing $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$. (b) Tension map showing sharp tension contours indicating class boundaries.*

SOFM has represented all input examples. For instance, in the simulations presented in the next section, we will have to compare how well features, i.e. input examples, are mapped in different maps. Knowing the exact tension levels of all examples provides a good indication. For that purpose we introduce the concept of *feature tension*. The calculation of all feature tensions resembles the second labelling method (cf. Figure 3.9) as stated in Proposition 3.10. Feature tension $f_t(k)$ of input $\xi^{(k)}$ is equal to the output of neuron η_a divided by the input dimension m. Neuron η_a is located at the position of $\xi^{(k)}$ on the labelled feature map. The division by m is introduced to enable a comparison between maps having different input dimensions. To compare the feature mapping quality of different maps, e.g. by using a t-test, we only need to estimate the mean and standard deviation of f_t's distribution as well as the sample size, i.e. the number of examples.

3.3.3 Visualization of the Weights

We recall that a SOFM represents two different aspects of the input distribution: (1) features of the input, and (2) the topology of the input distribution. The tension map and feature tension distribution introduced in the previous section may be used to analyze the quality of the feature representation. In this subsection and the next subsection visualization methods are presented to analyze the weight-point structure indicating how well the topology of the input distribution is represented by the SOFM.

In this subsection a weight-point projection method is presented that can be used for interactive visualisation of the SOFM topology. We recall that in the previous section the organization of a two-dimensional SOFM was visualized by a projection of its weight points. The input dimension of the SOFM depicted in Figure 3.6 is two dimensional and, hence, the weight points can be visualized without any problem. The input dimension of the SOFM depicted in Figure 3.7 is three dimensional. We have developed an interactive program to display a three-dimensional weight-point structure on a computer display. With this program it is possible to "fly" around the three-dimensional weight points in order to comprehend their combined structure. In Figure 3.7 a wire frame of a cube is shown to emphasize the three-dimensional structure.

The weight-point organization of a 2D SOFM with input dimension $m > 3$ may have an m-dimensional structure. Since the human vision system is limited to perception of three-dimensional structures, an m-dimensional structure must be transformed into a three-dimensional structure before it can be visualized. Hence, the visualization of weight points $\mathbf{w}_a \in R^m$ as $(x, y, z)_a \in R^3$ requires a projection matrix $\mathbf{P}$ from $R^m \rightarrow R^3$. In general this projection may destroy some structure owing to the dimension reduction. However, an organized 2D SOFM has a two-dimensional structure and hence it is often possible to calculate a projection matrix $\mathbf{P}$ that preserves the weight-point structure. The characteristics of $\mathbf{P}$ are determined by the weight-point organization in R^m which is not known beforehand. Hence, visualizing, for instance, the unfolding grid lines (cf. Figure 3.6) is not as simple as for a SOFM with input dimension two or three. Moreover, as the weight-point organization evolves, $\mathbf{P}$ must be adjusted continuously.

We note that $\mathbf{P}$ is a projection from $R^m \rightarrow R^3$ while the weight-point structure of an organized 2D SOFM is two-dimensional. The third dimension provided by $\mathbf{P}$ is used to display the average deviation from the main two-dimensional structure. When this deviation is small the three-dimensional visualization depicts a flat surface and hence the SOFM is organized in a two-dimensional structure.

Prior to projection by operator $\mathbf{P}$ a translation vector is added to the weight points to center the weight-point structure around the origin. The translation facilitates the localization of the projection in R^3. For example, by taking the weight vector of the center neuron $\eta_{n/2}$ as translation vector the weight points will be centered around the origin whence the map is organized. For simplicity we will assume here that $\mathbf{w}_{n/2}$ is always equal to O^m, the origin in R^m.

Projection matrix $\mathbf{P}$ is constructed from three orthonormal vectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$. The projection of $\mathbf{w}_a$ results in a three-dimensional weight point $(x, y, z)_a$ as follows:

$$(x, y, z)_a = \mathbf{w}_a^{\mathrm{T}}\mathbf{P} = (\mathbf{w}_a^{\mathrm{T}}\mathbf{v}_1, \mathbf{w}_a^{\mathrm{T}}\mathbf{v}_2, \mathbf{w}_a^{\mathrm{T}}\mathbf{v}_3) \tag{3.17}$$

Weight points in an organized 2D SOFM are arranged in a grid-like structure (cf. Figure 3.7(c)) which should be visualized by the projection on $\mathbf{v}_1$ and $\mathbf{v}_2$. Hence, $\mathbf{v}_1$ and $\mathbf{v}_2$ may be calculated using the horizontal and vertical axis of the grid-like structure of a 2D SOFM. We will denote the weight vector $\mathbf{w}_a$ of neuron η_a as $\mathbf{w}(i, j)$ where $i = a_x$ and $j = a_y$. The horizontal direction is determined by the vector between horizontal neighbours, i.e. $\mathbf{w}(i+1, j) - \mathbf{w}(i, j)$. The vertical direction is determined by the vector between vertical neighbours, i.e. $\mathbf{w}(i, j+1) - \mathbf{w}(i, j)$. This is illustrated in Figure 3.14. We note that in this figure $\mathbf{w}_{n/2} \neq O^m$ for a better illustration of the construction of vectors $\mathbf{v}_1$ and $\mathbf{v}_2$.

The horizontal direction $\mathbf{v}_1$ may be determined as follows :

$$\begin{aligned} \mathbf{v}_1' &= \sum_{i=1}^{n_x-1}\sum_{j=1}^{n_y} \mathbf{w}(i+1, j) - \mathbf{w}(i, j) = \sum_{j=1}^{n_y} \mathbf{w}(n_x, j) - \mathbf{w}(1, j) \\ \mathbf{v}_1 &= \frac{\mathbf{v}_1'}{\|\mathbf{v}_1'\|} \end{aligned} \tag{3.18}$$

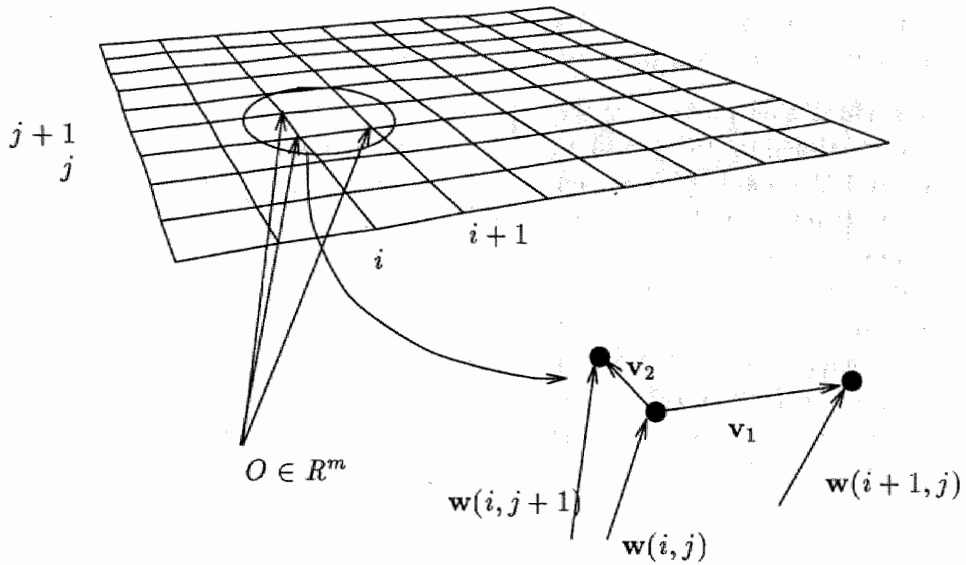


Figure 3.14: *Calculation of horizontal and vertical projection axis from weights in an organized SOFM.*

Similarly, the vertical direction $\mathbf{v}_2$ may be determined :

$$\begin{aligned}
\mathbf{v}_2' &= \sum_{i=1}^{n_x}\sum_{j=1}^{n_y-1} \mathbf{w}(i,j+1) - \mathbf{w}(i,j) = \sum_{i=1}^{n_x} \mathbf{w}(i,n_y) - \mathbf{w}(i,1) \\
\mathbf{v}_2'' &= \mathbf{v}_2' - \mathbf{v}_1\mathbf{v}_1^{\mathrm{T}}\mathbf{v}_2' \\
\mathbf{v}_2 &= \frac{\mathbf{v}_2''}{\|\mathbf{v}_2''\|}
\end{aligned} \tag{3.19}$$

We note that an additional operation involving $\mathbf{v}_2''$ has been inserted to make $\mathbf{v}_1 \perp \mathbf{v}_2$.

Finally, we must determine a third projection axis $\mathbf{v}_3$. Unlike $\mathbf{v}_1$ and $\mathbf{v}_2$, vector $\mathbf{v}_3$ is not dictated by a direction in the grid-like structure since a two-dimensional SOFM grid is restricted to a horizontal and a vertical direction.

Hence, $\mathbf{v}_3$ may be chosen arbitrarily with the single constraint that it is orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$. Many schemes can be used for calculating $\mathbf{v}_3$ depending on the purpose of the visualization.

In general this purpose is the visualization of the weight-vector components orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$. On the one hand, if all weight vectors in the SOFM do not contain components orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$ the weight-point structure and consequently also the input-distribution structure are two dimensional. On the other hand, if such components do exist, i.e. are visible, it follows that the input distribution must have more than two dimensions. This can indicate, for instance, that in parts of the feature map where such components are visible the map representation is not topologically correct. The components may have different directions in R^m and hence a logical choice for $\mathbf{v}_3$ is the average weight vector orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$.

$$\begin{aligned} \mathbf{v}_3' &= \sum_{a=1}^{n} \mathbf{w}_a \\ \mathbf{v}_3'' &= \mathbf{v}_3' - \mathbf{v}_1\mathbf{v}_1^T\mathbf{v}_3' - \mathbf{v}_2\mathbf{v}_2^T\mathbf{v}_3' \\ \mathbf{v}_3 &= \frac{\mathbf{v}_3''}{\|\mathbf{v}_3''\|} \end{aligned} \tag{3.20}$$

Using this method $\mathbf{v}_3$ represents the direction of the average component in the weight vectors orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$. Hence, owing to the average operation this direction is determined by the most important deviations of the $\mathbf{v}_1$ and $\mathbf{v}_2$ plane. Because the two main principal components are expressed by $\mathbf{v}_1$ and $\mathbf{v}_2$ the deviation of a weight vector from the ideal plane is visualized as a height value by its projection onto $\mathbf{v}_3$.

In Figure 3.15 an animation of the self organization of the city feature map (cf. Figure 3.9) is presented. Before each frame the projection operator $\mathbf{P}$ is calculated so that an up-to-date projection of the weight-point structure is visualized. A more detailed inspection of Figure 3.15(d) using the interactive visualization program reveals that the structure is nearly flat. Hence, as expected the input distribution is two dimensional.

Also the weight-point structure of the feature map depicted in Figure 3.13

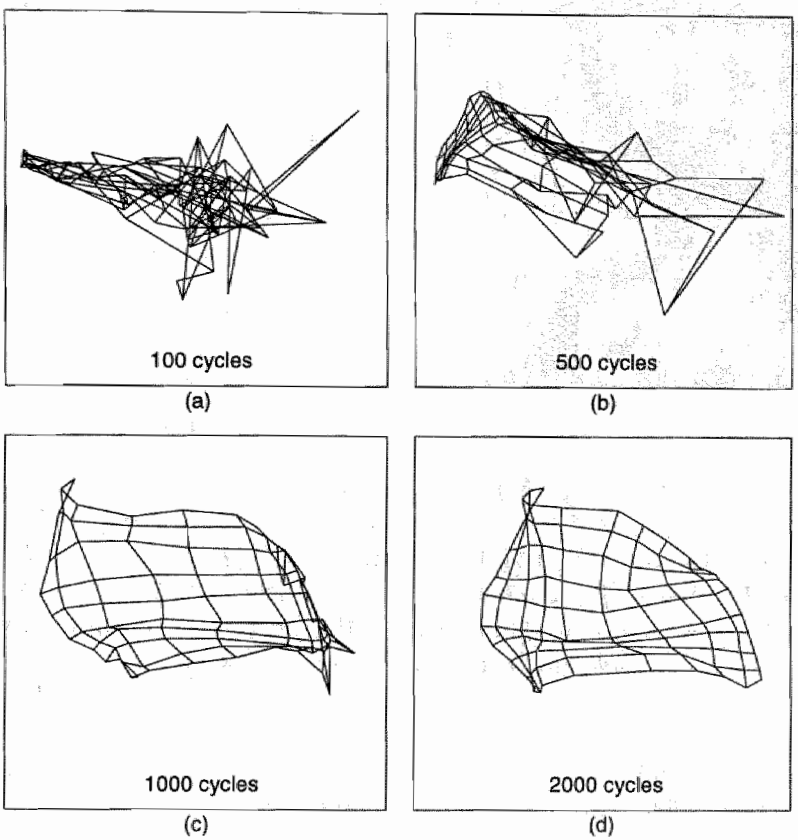


Figure 3.15: *Animation of the self organization in the city feature map at 100, 500, 1,000 and 2,000 iterations.*

can be visualized using this technique. In Figure 3.16(a) a combination of this feature map and the corresponding tension map is shown.

The weight-point structure depicted in Figure 3.16(b) appears to be triangular shaped. Moreover, the weight points are concentrated in the corners labelled $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ respectively. These points correspond to neurons in the feature map depicted in Figure 3.16(a). The projection of weight points in the corners is so dense that they can no longer be distinguished from each other. Weight points at the edges and in the center of the projected

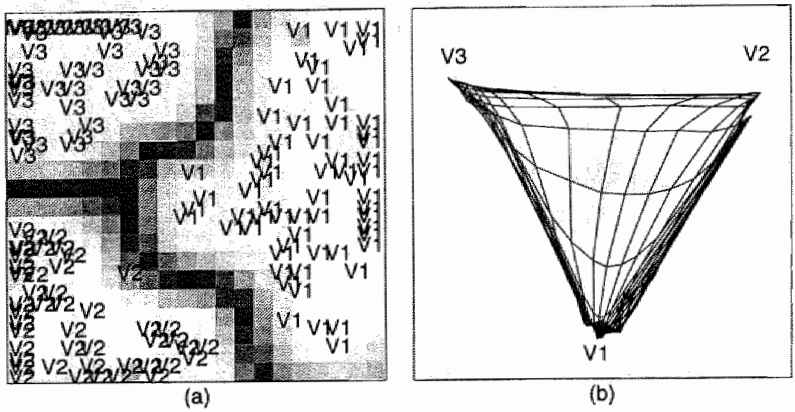


Figure 3.16: *Weight projection of a SOFM receiving input data having a six-dimensional structure. (a) Feature map of $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ with tension lines. (b) Weight-point projection of the weights showing a triangular-shaped weight distribution. The weight projection is dominated by the "tension" neurons that lie far apart and thus dominate the average weight direction.*

triangle correspond to neurons at the tension lines, i.e. the dark lines in Figure 3.16(a). They are much wider spaced than the weight points in the corners. Hence, neurons in the tension area are much less similar than neurons within the same corner, i.e. an area with no tension. This property can also be exploited for visualization purposes. For instance, a grey-level map similar to the tension map can be calculated in which each grey value represents the average Euclidean distance between a weight point and its four neighbouring neurons (Kraaijveld *et al.*, 1992; Scholtes, 1993).

The visualization of the weight points in Figure 3.16(b) does not show the weight-point structure of areas representing a single class. In order to visualize the weight-point structure of a single class the calculation of $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ should be restricted to weight points located in the same corner. In Figure 3.17(a) the neurons corresponding to the weight points in the upper-left corner of the triangle are marked. We note that this corner corresponds to class $\mathcal{V}_3$. In Figure 3.17(b) the projection of these weight

points is shown in case the calculation of $\mathbf{P}$ is restricted to these weight points. Similarly, the weight points in the lower corner and the upper-right corner result in Figure 3.17(c)+(d) and (e)+(f), respectively. All three corners appear to have a two-dimensional structure which is correct since classes $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ are two-dimensional planes (cf. Equation 3.14, 3.15 and 3.16, respectively).

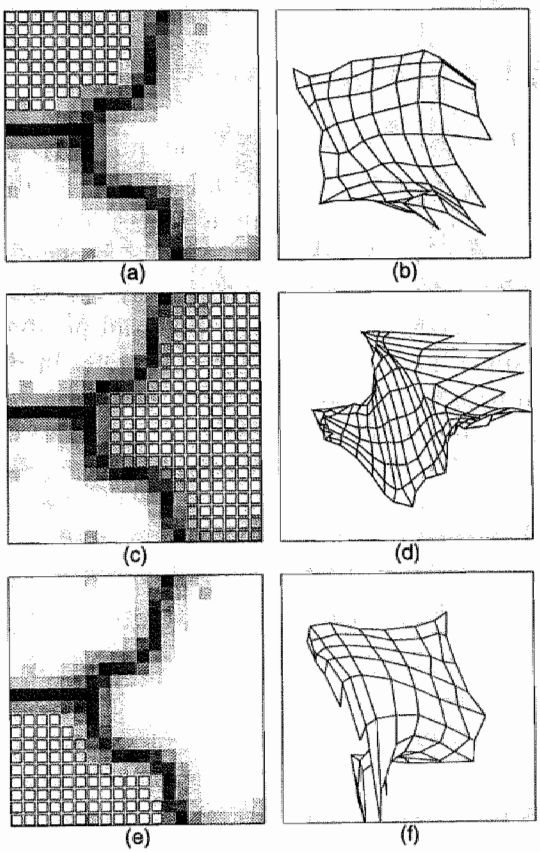


Figure 3.17: *Marked neurons on the left represent classes (a) $\mathcal{V}_1$, (c) $\mathcal{V}_2$ and (e) $\mathcal{V}_3$. (b), (d) and (f) show weight-point projection restricted to each class respectively.*

Comparing Figure 3.16(a) and (b) we may conclude that a visualization of the weight points provides a measure for the separation of feature classes. Moreover, the individual classes may be further analyzed by a restricted calculation of the projection operator (cf. Figure 3.17). This is a useful addition to the tension map since such a map cannot always be calculated. When the number of different input vectors is infinite it is impossible to calculate a tension map owing to Proposition 3.13. For instance, the SOFM depicted in Figure 3.6 can receive any point from a small part of the unit square. This area may be small but it contains infinitely many points.

3.3.4 In-depth and Light-fall Visualization

The graphical visualization of the weight-point projection is particularly valuable if the viewer is presented with enough angles to capture the three-dimensional structure. Ideally, a holographic image of the data should be produced such that a full three-dimensional view is possible. Using a computer, a very good alternative is to fly interactively through the projected weight points in R^3. The combination of motion and perspective produces a convincing three-dimensional image. Unfortunately, in documents it is not (yet) possible to present animated illustrations and it is too expensive to produce holograms. Hence, other alternatives should be found to enable a practical use of the weight-point projection. In this concluding subsection we present two methods that enhance the interpretation of a static weight-point projection. Firstly, we present the in-depth projection that only visualizes the projection of weight points on $\mathbf{v}_3$. Secondly, we describe how light fall may be used to enhance depth perception in a weight-point projection.

Sometimes it is sufficient to visualize how much a weight point deviates from the main two-dimensional structure. Depending on how the third projection vector $\mathbf{v}_3$ is calculated, the deviation of a weight point is represented by its projection of the weight point on $\mathbf{v}_3$. Since this projection measures the height of a weight point relatively to the main two-dimensional structure, we refer to this method as *in-depth* visualization. It resembles the tension map using *neuron depth* instead of neuron tension as a grey value. An example is presented in Figure 3.18(a) depicting an in-depth visualization and in Figure 3.18(b) a weight-point projection of a SOFM

receiving three inputs $\xi = (\xi_1, \xi_2, \xi_3) \in [0,1] \times [0,1] \times [0,0.1]$. Because the distribution of the third input ξ_3 is small compared to the distribution of the first two inputs ξ_1 and ξ_2, the weight-point projection is dominated by the two-dimensional organization of ξ_1 and ξ_2. Stripes emerge in the "in-depth" visualization owing to the slight variation in ξ_3.

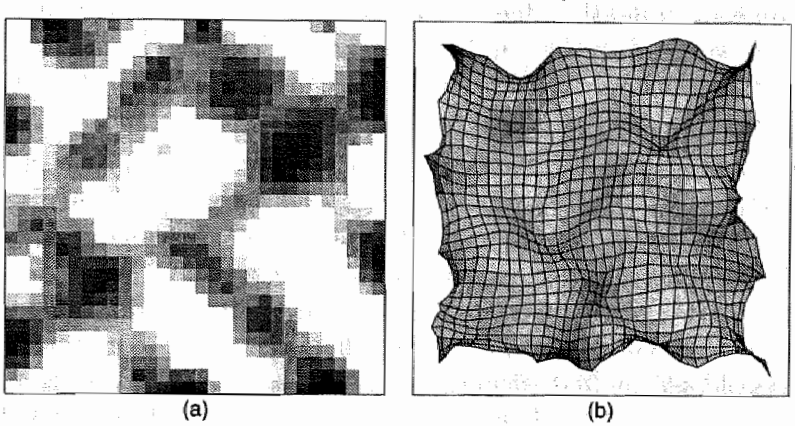


Figure 3.18: *In-depth visualization of a SOFM receiving three inputs having a two-dimensional topology with slight variations in the third dimension resulting in stripes on the surface. (a) In-depth visualization showing depth only. (b) Weight-point projection with light-fall representation enhances depth perception.*

In this example, the input to the map is three dimensional and the projection of $\mathbf{w}_a$ onto $\mathbf{v}_3$, i.e. z_3 in Equation 3.17, automatically represents the deviation of the total deviation in all directions orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$. However, in case the input dimension is larger then three, i.e. $m > 3$, $\mathbf{w}_a^{\mathrm{T}}\mathbf{v}_3$ represents only part of the error for weight point $\mathbf{w}_a$. This is due to the calculation of $\mathbf{v}_3$ defined in Equation 3.20. Hence, for the in-depth visualization we will calculate z_a as the length of the component of $\mathbf{w}_a$ orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$. Let (x_a, y_a) denote $(\mathbf{w}_a^{\mathrm{T}}\mathbf{v}_1, \mathbf{w}_a^{\mathrm{T}}\mathbf{v}_2)$ then

$$z_a = \|\mathbf{w}_a - x_a\mathbf{v}_1 - y_a\mathbf{v}_2\| \tag{3.21}$$

Using this definition of z_a the deviation of weight point $\mathbf{w}_a$ can be defined as $z_a/\sqrt{m}$. We divide z_a by $\sqrt{m}$ to eliminate the influence of the number of input dimensions so that in-depth comparisons between maps having different input dimensions are possible. We recall that in the section on map tension we have introduced feature-tension statistics to enable an objective, hypothesis-driven comparison. Similarly, we introduce in-depth statistics. The sample of in-depth values contains all weight points and hence the sample size is equal to the number of neurons in the SOFM. The average in-depth value and its standard deviation can be calculated directly from $\{z_1/\sqrt{m}, z_2/\sqrt{m}, \ldots\}$.

We note that in Figure 3.18(b) the weight-point projection is also grey coloured indicating the *light fall* on the surface of the weight-point structure. Hence, it is called a light-fall visualization. This has been realized by defining a normalized *light vector* representing the direction from which light strikes the surface. The weight-point projection is constructed from polygons corresponding to the cells in the grid-like structure. Hence, a polygon is defined by four weight points, namely, $\mathbf{w}(a_x, a_y)$, $\mathbf{w}(a_x + 1, a_y)$, $\mathbf{w}(a_x + 1, a_y + 1)$ and $\mathbf{w}(a_x, a_y + 1)$. Its grey colour is determined by the inner product of the light vector and the normal vector of its surface. This normal vector is defined as the normalized outer product of the horizontal and vertical direction of the polygon in the weight-point structure. The horizontal direction is calculated by subtracting the R^3 projection of η_a and its horizontal successor in the map : $(x, y, z)_{a_x+1, a_y} - (x, y, z)_a$. In the same way the vertical direction is determined as $(x, y, z)_{a_x, a_y+1} - (x, y, a)_a$. By choosing a particular light angle, surface parts having a flat structure can be distinguished since they display a uniform brightness. Such parts could be called *map facets*.

Finally, we mention that it is also possible to project the weights using different projections for the left and right eye. If these projections are presented separately to each eye, the depth perception is virtually real as each eye looks from a slightly different angle to the weight-point projection. We have programmed this option in the flight-simulator program using a red and green colour on a computer display to realize the separate presentation to the eyes. Unfortunately, it is not possible to produce an example fit for illustration in the thesis, since it requires color printing.

These descriptions of the in-depth and light-fall visualization technique conclude the section on visualization of 2D SOFMs. The visualization methods will be used in the next section where an application will be presented illustrating the fusion of multiple sensors into a modular system containing multiple SOFMs.

3.4 Modular SOFMs

As we have shown in the first section of this chapter, sensory events are presented in the cortex by primary maps. The functional decomposition of non-correlated inputs into distinct maps is the basis for a successful reduction of inputs to the surface of the cortex (Kohonen, 1982; Durbin and Mitchison, 1990). For instance, when one considers the entire tactile system of the body there are thousands of input variables. When these inputs are processed by a single SOFM this will undoubtedly yield a poor dimension reduction. However, owing to the existence of a functional mapping that partitions the body surface into many small topological maps, the problem is simplified. For instance, the signals derived from a single fingertip are highly correlated and are much easier to map in two dimensions than the combination of a fingertip and, e.g. the upper lip. Hence, the functional partitioning in the primary zones is an excellent preprocessing stage representing various input signals by standard cortical maps.

Inspired by the functional partitioning of the primary cortical zones, our approach is aimed at decomposing input sensors over a number of 2D SOFMs. This approach should not be confused with other approaches in which a hierarchical system of SOFMs is built (Rojer and Schwartz, 1989; Koikkalainen and Oja, 1990). In a hierarchical system the map surface is decomposed into a tree of smaller map surfaces. In contrast, our modular SOFM is a collection of separate 2D SOFMs and resembles the approach followed by Ritter (1989). However, in Ritter (1989) SOFMs are used as adaptive function tables without use for their topology-preserving mapping property. By contrast, our main criterion for decomposition is related to this property as we try to (visually) optimize the two-dimensional topology of each SOFM module.

One type of SOFM application that strongly resembles the function of the

primary cortex is the representation of sensor data, for instance, from industrial processes. Using two-dimensional SOFMs as a standard representation for all kinds of sensors has much the same benefits as using primary maps in the cortex. In general, a feature map representation of sensor data has the following advantages.

- A mapped representation provides a uniform interface (Knudsen *et al.*, 1987) in which all data types are transformed to features on a feature map regardless of their nature or scale. This will facilitate, e.g. the fusion of sensor data in subsequent processing stages.
- The SOFM is tuned automatically to an efficient representation of all relevant features. In some cases preprocessing with a SOFM may enhance the performance of, for instance, a Multi-Layer Neural Network (Holdaway, 1989).
- A single SOFM can represent multiple sensors as long as they are correlated (Kohonen, 1982).
- The map operates in parallel so that real-time transformation of sensor data is feasible (Knudsen *et al.*, 1987). Moreover, real-time representation of continuous-time state sequences is facilitated as a continuously changing state corresponds to a gradually moving activity cluster on the map.

In addition to the advantages of using a feature map representation, using a *modular* SOFM, i.e. a system containing multiple SOFMs, has the following advantages.

- In general, a modular approach is useful for maintaining the feature maps. For instance, when sensors are eliminated or new sensors are added, only those maps involved need to be recalculated. In a single map system, the entire map would need to be recalculated.
- Non-correlated signals may be represented. When non-correlated signals are presented to a single SOFM it is forced to separate the input components and represent non-correlated components on different

sections of the map. For instance, in the example presented in Figure 3.13, class $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ were mapped on separate areas. When it is known beforehand that the classes are represented by (ξ_1, ξ_2, ξ_3), (ξ_4, ξ_5, ξ_6) and (ξ_7, ξ_8, ξ_9), respectively, the single map could have been replaced by three separate maps. Such *a priori* knowledge will result in a better feature map.

- A modular SOFM is more robust than a single SOFM as each map operates independently.

Of course, using a modular SOFM also has some inevitable disadvantages, such as

- The sensor input is represented by multiple feature maps making it more difficult to interpret the relation between different sensor events.
- It is difficult to determine a well-chosen modular configuration, i.e. which sensors should be combined, and which not. In a single SOFM all sensors are input to the same map.

We feel that the advantages of the modular SOFM approach outweigh the disadvantages stated here. Especially since the first disadvantage is inherent to the capability of the modular SOFM to represent non-correlated sensors. Concerning the second disadvantage we will introduce a simple heuristic that assists in finding a suitable *modular* configuration.

In this section we wish to demonstrate that multiple two-dimensional SOFMs can be adapted to represent input distributions having more than two dimensions. The proposed system is a *modular SOFM* because it contains several SOFMs, each receiving a number of sensors. Ideally, only one map is required to represent all sensors. In the worst case, one map is needed for every two sensors if there is no correlation at all between input sensors. In many practical situations, for instance, in the brain, a compromise is possible by fusing only correlated sensors in a single map. However, the assignment of particular sensors to a SOFM is not a trivial task. In the cortex, the structure and the partitioning of primary maps are genetically determined. When designing a modular SOFM for sensor representation a suitable configuration should be determined, indicating

which sensors may be combined in a single map. The basic information needed for realizing a successful configuration is represented by the correlations between sensors. We will show that this task may be solved by using a SOFM which represents the sensor-correlation matrix. Tension lines on the resulting correlation-feature map indicate which sensors may be assigned to the same map. Besides the tension map other visualization techniques are used to evaluate the weight-point structures of the maps in the modular SOFM. The proposed method will be tested on sensor data of a small part of a refinery of the Dutch State Mine (DSM) plant. We begin presenting some background information on the nature of the sensors.

3.4.1 Background Information

Figure 3.19 depicts a schematic drawing of the refinery process. It indicates the positions of 28 sensors $s_1, \ldots, s_{28}$ used to monitor the process state. The sensors either measure temperature (TS), pressure (PS), flow (FS) or chemical analysis (AS). For our experiments we used 3540 samples representing the process state over a period of five days. Using a single 2D SOFM, a feature map can be computed representing all states which occurred during these five days. The aim of the experiment conducted here is to compare the dimension reduction in such a map with the dimension reduction in a modular 2D SOFM containing several feature maps each receiving a selected number of sensors.

In Table 3.3 we have presented a statistical analysis of each sensor calculating its average value, standard deviation, and maximum and minimum value for the data set chosen.

Before the experiments are conducted the data of each sensor is scaled to the interval $[-1, 1]$ by subtracting the average μ_i and dividing this number by the difference between maximum value of $|max_i - \mu_i|$ and $|min_i - \mu_i|$. By doing so we assure ourselves that the input distribution is not biased by a particular sensor having a large deviation from its average value. If the input vectors are denoted $\xi^{(1)}, \ldots, \xi^{(3540)} \in R^{28}$, then scaling is performed as follows

$$\xi_i^{(t)} = \frac{s_i(t) - \mu_i}{max(|max_i - \mu_i|, |min_i - \mu_i|)}, \tag{3.22}$$

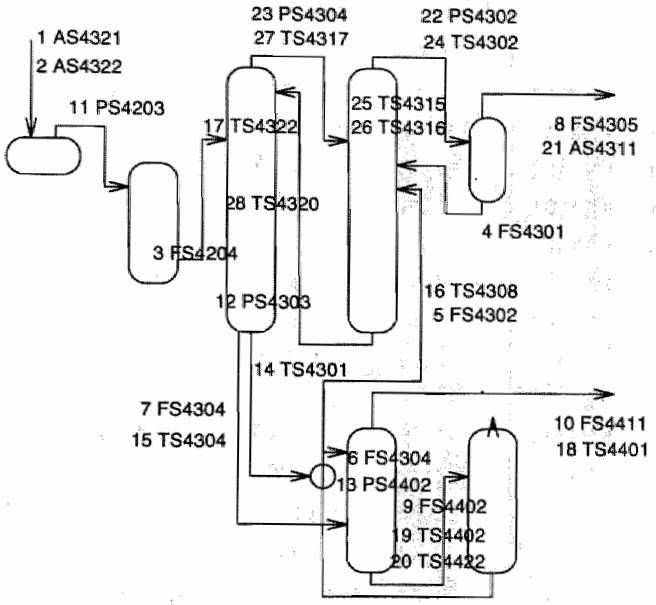


Figure 3.19: *Schema of the sensor locations in the refinery.*

where $s_i(t)$ denotes the value of sensor i (cf. Table 3.3) at time $t \in [1, 3540]$.

3.4.2 Visualizing Correlations

A very simple method for visualizing the correlations between the input sensors is to calculate the correlation matrix of the input vectors. This matrix is square and symmetric having elements ρ_{ij} denoting the statistical correlation between $s_i(t)$ and $s_j(t)$ (Mood *et al.*, 1950):

$$\rho_{ij} = \frac{1}{3540\sigma_i\sigma_j} \sum_{t=1}^{3540} (s_i(t) - \mu_i)(s_j(t) - \mu_j) \tag{3.23}$$

We note in passing that this correlation measure does not account for correlations between delayed sensor readings. In Figure 3.20 a graphical representation for the correlation matrix of the sensor data is represented.

Name (i)	Name (s_i)	Average (μ_i)	Deviation (σ_i)	Maximum (max_i)	Minimum (min_i)
1	AS4321	1.43	0.09241	1.70	1.30
2	AS4322	80.87	125.74612	1680.00	2.30
3	FS4204	74.37	6.23819	82.40	57.80
4	FS4301	5.35	0.21678	5.90	4.70
5	FS4302	24.32	0.81185	26.10	22.50
6	FS4303	6.98	1.84654	10.50	2.80
7	FS4304	27.50	0.96406	29.90	25.00
8	FS4305	72.15	5.92849	79.70	56.20
9	FS4402	32.72	1.95770	39.10	26.50
10	FS4411	1825.30	116.95012	2134.60	1559.30
11	PS4203	24.97	0.08328	25.20	24.60
12	PS4303	18.71	0.11942	19.00	18.30
13	PS4402	0.30	0.00000	0.30	0.30
14	TS4303	41.36	0.68567	46.00	38.30
15	TS4304	17.54	0.61128	20.00	15.20
16	TS4308	-13.55	0.48484	-9.80	-15.70
17	TS4321	-26.23	0.28399	-25.60	-27.10
18	TS4401	-46.46	0.64726	-45.00	-47.70
19	TS4402	54.10	2.46476	62.10	50.60
20	TS4422	49.15	2.54988	56.20	45.40
21	AS4311	0.14	0.58059	9.90	0.00
22	PS4302	0.22	0.03714	0.30	0.20
23	PS4304	0.30	0.00291	0.40	0.30
24	TS4302	-23.17	0.63673	-21.60	-25.20
25	TS4315	-12.47	1.06265	-9.50	-14.00
26	TS4316	-14.98	0.74490	-12.80	-16.30
27	TS4317	-14.66	0.72436	-12.60	-16.00
28	TS4320	-26.13	0.27742	-25.50	-27.00

Table 3.3: *Statistics for the DSM sensor data.*

The interpretation of the correlation matrix is similar to the interpretation of the distance table presented in Table 3.1 which was visualized in two di-

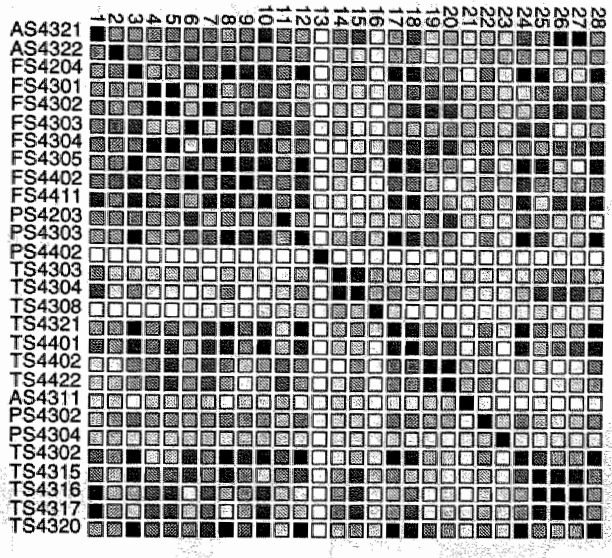


Figure 3.20: *Graphical representation of the correlation matrix of the sensor data. At the vertical axis the sensor names are denoted. At the horizontal axis the sensor names have been replaced by their index number according to Table 3.1.*

mensions using a SOFM (cf. Figure 3.9). In the same way we can present the row vectors of the correlation matrix to a SOFM in order to visualize the correlations between the sensors. A sensor is characterized by its correlations with the other sensors in a similar way as a city is characterized by its distances to other cities. We note that sensor 13 (PS4402) is not correlated because it is constant (cf. Tabel 3.3). Therefore, mapping this sensor in a SOFM will not present any problems in spite of its seeming not to be correlated with any other sensor (cf. Figure 3.20).

The correlation matrix holds only one correlation vector for each sensor. In order to obtain a more realistic correlation feature map we will generate a

number of correlation matrices by randomly selecting samples from the data set. The random sampling is simulated by inserting a binary probability factor depending on t in Equation 3.23. By using multiple, randomly generated correlation matrices the SOFM will be able to model slight variations within sensor correlations. Moreover, owing to the noise that is introduced by the randomly calculated correlation vectors the self-organizing process is less easily trapped in a suboptimal weight-point organization.

We demonstrate the two-dimensional visualization of the correlation matrix by using the input classes $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ (cf. Equations 3.14, 3.15 and 3.16) are represented by the feature map depicted in Figure 3.13. The correlation matrix of the 100 examples is presented in Figure 3.21(a) and the corresponding SOFM visualization of these correlations is depicted in Figure 3.21(b).

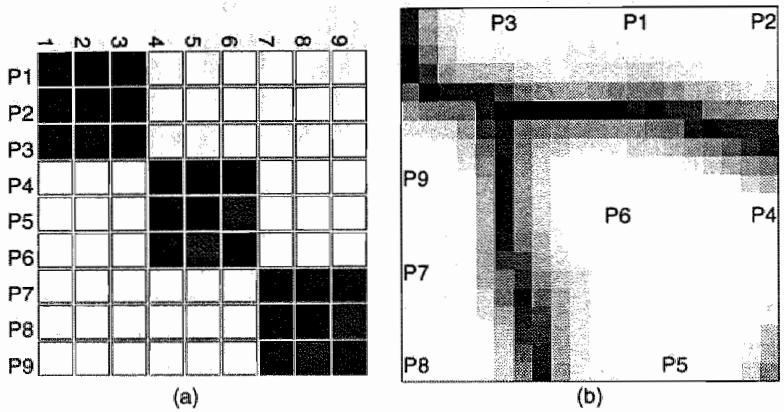


Figure 3.21: *Correlations between the components representing classes $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$. (a) Correlation matrix. (b) Labelled correlation feature map with tension showing that the input vector may be separated into three classes: $\{p_1, p_2, p_3\}$, $\{p_4, p_5, p_6\}$ and $\{p_7, p_8, p_9\}$.*

In this simple example the correlation matrix already suffices to determine which sensors may be combined, namely $\{p_1, p_2, p_3\}$, $\{p_4, p_5, p_6\}$ and $\{p_7, p_8, p_9\}$. This separation is confirmed by the definition of the input

classes in Equations 3.14, 3.15 and 3.16.

The next step towards a modular SOFM for representing the sensor data is to determine how many SOFMs are required and which sensors each map should represent. We emphasize that although the tension lines on the correlation map indicate which sensors are clustered, such a map does not give an exact description. Hence, the correlation feature map should be used as a heuristic for determining which sensors are best clustered. We strongly recommend that any knowledge regarding the sources of the sensors should be taken into account if available. Furthermore, using the visualization techniques introduced in the previous section it is possible to visually inspect the result of a partioning. Based on this inspection, partionings may be changed interactively. In Figure 3.22 we have depicted the correlation feature map of the sensor data as well as the corresponding tension map.

Based on the correlation feature map we have determined seven sensor clusters. We note that four clusters depicted for sensors 11, 13, 16 and 23 are not useful. From the statistics presented in Table 3.3 it follows that sensor 13 has a constant value while sensor 23 is nearly constant deviating little from the average value. Hence, we have chosen to combine 23 with 14, 15 and 21 while combining 13 with 4, 5 and 7. Some additional experiments have shown that sensors 11 and 16 are acceptably modelled when combined with sensors 19 and 20. Finally, we have computed an 10×10 correlation feature map representing sensors 3, 8, 12, 10, 17, 18, 22 and 24 only since they form a rather large cluster in Figure 3.22(c) and are not organized very well. This restricted correlation feature map is depicted in Figure 3.23. We note that sensor 28 (TS4320) is represented in (a) but not in (b). Apparently, sensor 17 and 28 are highly correlated which is confirmed by the correlation matrix depicted in Figure 3.20. Together with 10, 12 and 18 they are combined. Sensor 22 is highly separated. Additional experiments showed that insertion of 22 in 3, 8 and 24 is least harmful.

In total we have divided the 28 sensors into 7 clusters based on their arrangement in Figure 3.22 taking into account the above mentioned adjustments. The clusters are numbered $1 \ldots 7$ starting in the top-left corner of Figure 3.22(d) to the top-right corner etc.

1. 1 (AS4321), 25 (TS4315), 26 (TS4316), and 27 (TS4317).

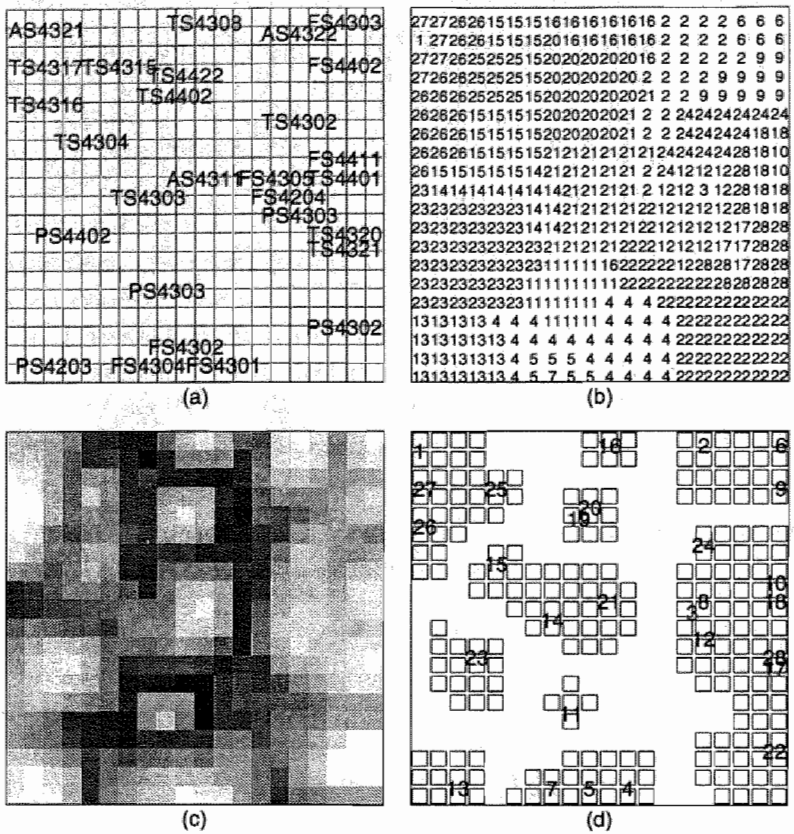


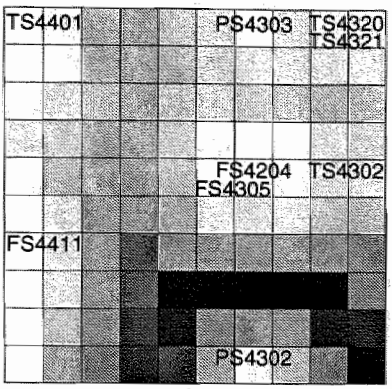
Figure 3.22: *Visualization of the sensor correlation matrix using a SOFM. (a) Feature map. (b) Full-labelled feature map. (c) Tension map. (d) Areas of low tension.*

2. 11 (PS4203), 16 (TS4308), 19 (TS4402), and 20 (TS4422).

3. 2 (AS4322), 6 (FS4303), and 9 (FS4402).

4. 14 (TS4303), 15 (TS4304), 21 (AS4311) and 23 (PS4304).

5. 10 (FS4411), 12 (PS4303), 17 (TS4321), 18 (TS4401), and 28 (TS4320).



(a)

18	18	18	24	12	12	12	12	17	17
18	18	18	24	24	12	12	12	17	17
18	18	18	24	3	3	3	12	12	12
18	18	18	24	3	3	3	3	3	24
10	10	18	24	8	8	8	8	8	24
10	10	10	24	8	8	8	8	8	24
10	10	10	24	8	8	8	8	8	24
10	10	10	24	18	24	24	24	24	8
10	10	10	18	22	22	22	22	22	24
10	10	10	10	22	22	22	22	22	22

(b)

Figure 3.23: *Feature map restricted to sensors 3, 8, 12, 10, 17, 18, 22, 24 and 28. (a) Tension map indicates sensor 22 (PS4302) is much different from the other sensors. (b) Full-labelled feature map showing the selectivity of neurons in the correlation feature map shown in (a).*

6. 4 (FS4301), 5 (FS4302), 7 (FS4304), and 13 (PS4402).

7. 3 (FS4204), 8 (FS4305), 22 (PS4302), and 24 (TS4302).

Before we continue the sensor-data experiment, we remark that a correlation feature map may be useful in a large number of applications. For instance, it is possible to calculate the correlation matrix of the distance table presented in Table 3.1 which will also yield a map of the Netherlands. Also we will use the correlation feature map in Chapter 4 to classify temporal sequences.

3.4.3 The Modular Sensor Map

Using the sensor separation presented above, we have computed a feature map for the sensor states. Initially, a single SOFM was used, representing all sensors in one feature map. Before comparing this map with the modular

approach we have depicted the weight-point projection with light fall, the tension map, the in-depth visualization and a line graph representing the sequence of all represented states of a 50×50 SOFM representing all sensor states $\xi^{(1)}, \ldots, \xi^{(3540)}$ in Figure 3.24(a), (b), (c) and (d), respectively.

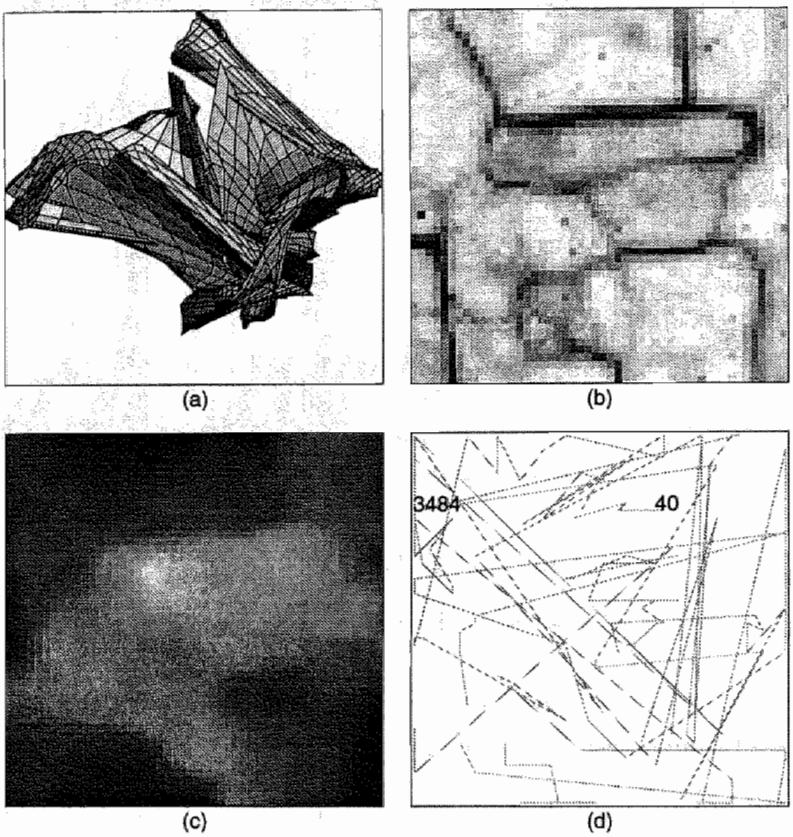


Figure 3.24: *State feature map computed by a* 50×50 *SOFM* ($\alpha = 0.1$, $\beta = 5$). *(a) Weight-point projection with light fall. (b) Tension map, (c) In-depth visualization. (d) Sequential representation of the sensor states.*

We note that the weight projection (cf. Figure 3.24(a)) shows various

hades of grey indicating that a large number of different surface parts are ısed to represent different state classes. This is confirmed by the tension nap (cf. Figure 3.24(b)) showing dark contours revealing a large number of classes. The depth variation in the weight projection is visualized in he in-depth projection (cf. Figure 3.24(c)). Clearly, the variation in the veight-point projection is also represented by different depth levels. Finally, n Figure 3.24(d) a new type of feature map visualization is displayed. This ype has not been introduced in the previous section since it can only be ıpplied to SOFMs representing ordered examples. The lines on the map orm a path representing the development of the process state starting at he first sample and finishing at the last sample. This representation may e used to visualize temporal behaviour of the sensors. Since we are only lealing with visualization and representation of the sensors an analysis of heir behaviour is beyond the scope of this chapter.

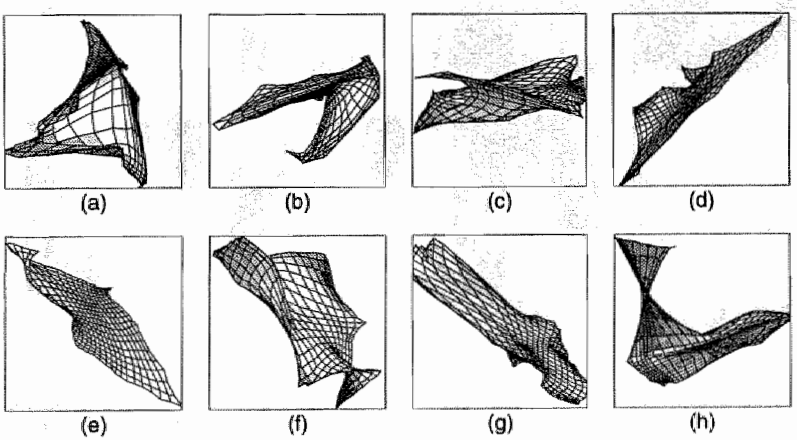


Figure 3.25: *Eight weight projections corresponding to (a) the feature map for all sensor sates while (b)-(h) correspond to the modular SOFM containing seven separated maps.*

Next, we compare the single feature map approach with the modular SOFM containing seven maps corresponding to the seven classes introduced above. In Figure 3.25 we have depicted eight weight projections of a 20×20 SOFM.

The projection in Figure 3.25(a) corresponds to the feature map representing all sensors. We note that the projection appears better than the projection depicted in Figure 3.24(a). This is caused by the reduced number of neurons in the SOFM prohibiting the map to adapt to detailed sensor-state variations. Moreover, the figure scale is much smaller, hiding many projection details. The weight-point projections of the seven maps of the modular SOFM are depicted in figures 3.25(b)-(h). These figures already indicate that the maps in the modular SOFM have a more uniform weight-point structure. In Figure 3.25(a) the grid points in the center of the weight projection are widely spaced similar to the weight projection of the $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ example (cf. Figure 3.16(b)).

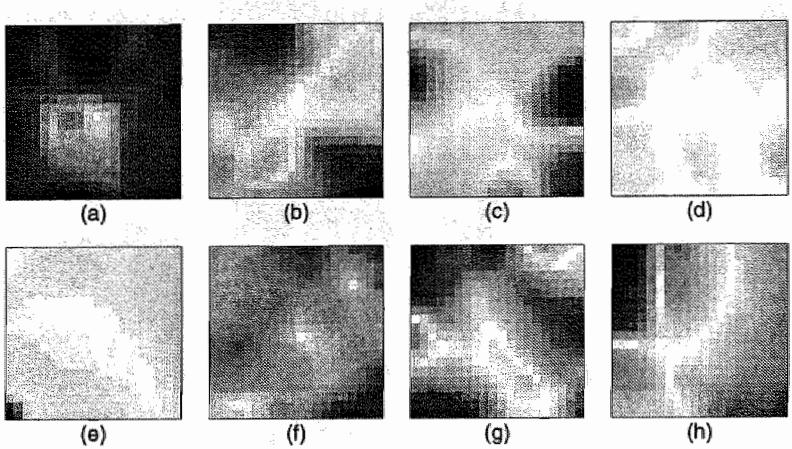


Figure 3.26: *In-depth visualization corresponding to (a) the feature map for all sensor states while (b)-(h) correspond to the modular SOFM containing seven separated maps.*

The difference of the weight projections becomes more clear using the in-depth visualization. In Figure 3.26 we have depicted the in-depth visualization for each weight projection. For an objective comparison between the in-depth visualization in Figure 3.26 we have presented in Table 3.4 the mean and standard deviation for in-depth visualizations (a)-(h). These values may be used to reject the zero hypothesis $\mathcal{H}_0$ that the mean depth of

map (a) μ_a is equal to μ_x for every map $x \in \{b, \ldots, h\}$. Let $\mathcal{H}_1$ denote the alternative hypothesis that $\mu_a > \mu_x$, i.e. the maps in the modular system are better than the single map. The corresponding t-test has 798 degrees of freedom (Mood *et al.*, 1950). The T value for each test is presented in the third column of Table 3.4. With a significance $\alpha = 0.001$ we require that $T > t_{0.001} = 3.090$ and hence we can reject $\mathcal{H}_0$ and accept $\mathcal{H}_1$ in each test. Hence, we conclude that weight-point structures (b)-(h) (cf. the projections shown in Figure 3.25(b)-(h)) are "flatter" than weight-point structure (a) (cf. the projection shown in Figure 3.25(a)).

In-depth statistics			
Map	μ	σ	T
a	0.174	0.062	-
b	0.056	0.053	45
c	0.048	0.051	49
d	0.009	0.008	412
e	0.009	0.014	236
f	0.060	0.031	74
g	0.045	0.035	74
h	0.061	0.055	71

Table 3.4: *Statistics for the in-depth visualizations depicted in Figure 3.26.*

Finally, in Figure 3.27 the tension maps corresponding to the visualizations in Figure 3.25 and 3.26 are presented. Each separate map in the modular SOFM shows much less tension areas than the tension map depicted in Figure 3.27(a). This difference becomes even more clear when the tension maps are compared to the tension map depicted in Figure 3.24(b). Once again we would like to draw attention to the similarities between the weight-point projections in Figure 3.25, the in-depth visualizations depicted in Figure 3.26 and the tension maps depicted in Figure 3.27. They show that tension is indeed an indication of an inhomogeneity in the weight-point structure and vice versa.

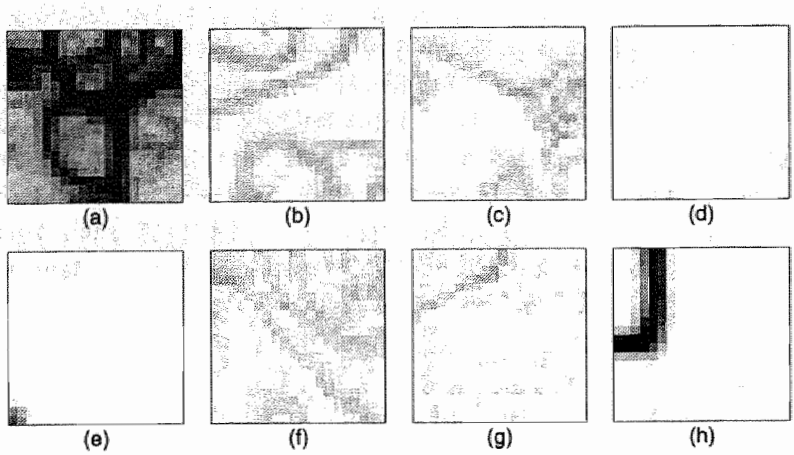


Figure 3.27: *Eight tension maps corresponding to (a) the feature map for all sensor states while (b)-(h) correspond to the modular SOFM containing seven separate maps.*

Again, for objective comparison of the feature map quality we have calculated the feature-tension statistics for the input examples. The feature-tension mean and standard deviation for the maps shown in Figures 3.25, 3.26 and 3.27 (a)-(h) are presented in Table 3.5. In this case a t-test with 7078 degrees of freedom can be applied to reject $\mathcal{H}_0$ and accept $\mathcal{H}_1$ if $T > t_{0.001} = 3.090$. The T-values are presented in the third column of Table 3.5 and in each test we reject $\mathcal{H}_0$. Hence we conclude that inputs in maps (b),...,(h) are better represented than in map (a).

This concludes the section on modular SOFMs. Using the visualization techniques presented in the previous section we compared the organization of a SOFM receiving 28 sensors with a modular system containing seven SOFMs each representing a selection of sensors. On the one hand, the modular SOFM yields better two-dimensional representations for states of correlated sensors. On the other hand, representing all sensors in a single map provides a better visualization of the temporal sequence.

Feature-tension statistics			
Map	μ	σ	T
a	0.115	0.040	
b	0.046	0.041	100
c	0.037	0.040	116
d	0.026	0.044	120
e	0.017	0.035	167
f	0.073	0.040	62
g	0.051	0.039	98
h	0.026	0.020	265

Table 3.5: *Feature-tension statistics.*

3.5 Chapter Summary

In this chapter we have described the modular architecture of the cortex. Based on neurophysiological knowledge about the various pathways in the cortex and the theory of Luria (1973) it was adopted that the modules are combined in a hierarchical working structure for receiving, analyzing and storing information. The modules in the primary cortex of the brain can be described by computational maps that represent instantaneous transformations of input stimuli into a two-dimensional map representation. The self-organizing feature map (SOFM) (Kohonen, 1982; Kohonen, 1984) is capable of automatic generation of computational maps such that the application of feature maps is virtually unlimited. Mimicking the modular architecture of the primary cortex we propose a modular system containing multiple 2D SOFMs such that input distributions which have more than two dimensions may be represented by multiple feature maps.

When dealing with such complex data distributions there is a need for methods that visualize the organization and activity of the SOFM. We have presented a method for visualizing tension in a feature map. Tension lines indicate possible class boundaries. Furthermore, we have derived a general weight-point projection algorithm for visualizing the three largest

components represented by the weights in the map. Using these techniques it is possible to analyze SOFMs enabling a comparison between the single and the modular SOFM approach. One of the main problems in the modular approach is to determine which input sensors may be represented in a single feature map. For this purpose it is possible to visualize the correlations between sensors using a two-dimensional SOFM. The tension map of the resulting correlation feature map indicates useful sensor combinations for determining a configuration of maps. In case large clusters emerge, sensors in such clusters may be represented in an additional correlation feature map. Using this method we designed a modular system containing seven SOFMs for representing 3540 samples of 28 sensors corresponding to chemical analysis, temperature, pressure and flow in part of a refinery. The dimension reduction by the modular system has been compared to the dimension reduction by a single SOFM. Comparing the weight projections, tension maps and in-depth visualizations, the modular system showed a better dimension reduction. Without the visualization methods and their derived statistics presented in Section 3.3 such a comparison would not have been possible. Moreover, based on the visualizations it can also be concluded that tension in a 2D SOFM indicates an inhomogeneity in the weight-point structure and vice versa. This inhomogeneity can also be visualized by displaying distances between neighbouring weight-points as grey values (Kraaijveld *et al.*, 1992; Scholtes, 1993).

Chapter 4

The Membrain Model

In this chapter[1] we study the nature of temporal representation in an ANN with an oscillating activation pattern. Earlier work of Braspenning (Braspenning, 1982; Stefanou *et al.*, 1987) in the field of solid-state physics inspired us to pursue a more wave-mechanical approach towards ANN modelling. Already in 1950, Lashley presented a metaphor of waves on the surface of a liquid to envision resonance between neurons as a basic mechanism for memory recall (cf. Chapter 1). Recently, the introduction of a wave metaphor has been justified experimentally. Nunez (1988) showed that both temporal and spatial properties of EEG data can be predicted using an oscillatory model of the neocortex. Continuing this line of research, Ingber and Nunez (1990) conjecture that EEG is global resonance of primarily long-ranged cortical interactions and, if so, relatively short-ranged local firing patterns can effectively modulate this frequency and its harmonics, in order to enhance information processing on a wider scale. Interestingly, Pribram (1991) proposes a quantum-mechanical model in which information processing is also represented by frequency modulation. Alternatively or possibly combined with modulation, Ingber and Nunez (1990) conjecture that global cortical interactions imply boundary conditions on collective states of local firings.

We find both modulation and boundary conditions important principles by

[1] A large part of this chapter has been published in the International Journal of Circuit Theory and Applications, Vol.20, pp. 483-496 (Henseler and Braspenning, 1992).

which representations in an oscillating neural network may be organized. To study these principles we introduce an ANN model that is based on a network of linear-coupled oscillators. Its activation dynamics resemble a vibrating membrane which is why it is called the *Membrain* model. The Membrain connection architecture resembles that of a Cellular Neural Network (CNN) (Chua and Yang, 1988a) (cf. Section 1.2.2). An important characteristic of a CNN is its cellular structure in which all neurons are cell *clones*. Unlike the MLNN and the SOFM, the original CNN model was not equipped with an adaptation mechanism and consequently it is less flexible. However, CNNs have received considerable attention. Owing to the cellular structure and the special neuron-output function, CNNs are relatively easy to build using standard VLSI techniques or array computers. Designing a CNN for an application one should find the template values for the cell clones determining the dynamic behaviour of the network. For instance, some successful templates have been derived from known digital image-processing algorithms (Chua and Yang, 1988b). Others have been developed experimentally by extensive simulations. Meanwhile, Zou *et al.* (1991) introduced a learning algorithm for CNNs. Since we focus on the CNN processing model rather than on the CNN connectivity matrix, we will not deal with this learning algorithm.

Section 4.1 introduces the functional architecture of CNNs (Chua and Yang, 1988a) dealing with the connection architecture, neuron architecture and activation dynamics. Section 4.2 presents a modification to the neuron architecture of the CNN model resulting in the Membrain model. Following the formal description of the Membrain model, the underlying system of linear, partial differential equations is solved analytically. Section 4.3 describes a specific Membrain CNN (MCNN) for preprocessing grey-level images by generating *translation-invariant* image representations. The output of this MCNN can be classified using traditional ANN models. In particular we show that: (1) a SOFM clusters correlated images, and (2) a MLNN detects translation-invariant features. In Section 4.4 the results are summarized.

4.1 Cellular Neural Networks

In this section we introduce the cellular structure of CNNs and subsequently provide a detailed description of their functional architecture. Following this description an outline of their activation dynamics is presented. Concluding this introduction we briefly discuss the realization of CNNs in hardware and their application to image processing.

4.1.1 Cellular Structure

A CNN is characterized by its two-dimensional cellular structure, that is, a network constructed from *identical* neurons placed on a two-dimensional grid in such a way that only neurons on neighbouring grid points are connected (cf. Figure 4.1). The two-dimensional CNN topology resembles the topology of neurons in a two-dimensional SOFM described in Chapter 3 (cf. Figure 3.4). However, in contrast to the local interconnections in a CNN, a SOFM has global interconnections to realize the lateral interaction between neurons. The lateral interaction weights are fixed so that neurons in the SOFM compete with each other, i.e. realizing a kind of "winner-takes-all" mode of operation. Based on this competition, SOFM neurons are *different* whereas CNN neurons are *identical* forming a cellular structure.

The cellular structure makes CNNs suited for processing input data that is represented by a two-dimensional array of data elements having a one-to-one correspondence to the neurons in a CNN (cf. Figure 4.2). Moreover, the processing task must be suited for parallel processing in such a way that only neighbouring data elements have to interact. Image processing is a typical domain where applications meet these requirements. An image can be described as a two-dimensional array of pixels to be used for mapping an image as input onto the neurons in a CNN. In that way tasks requiring local interaction between pixels can be executed in parallel, e.g. noise removal, edge extraction, edge detection (Chua and Yang, 1988b) and preprocessing for optical character recognition (Matsumoto *et al.*, 1990).

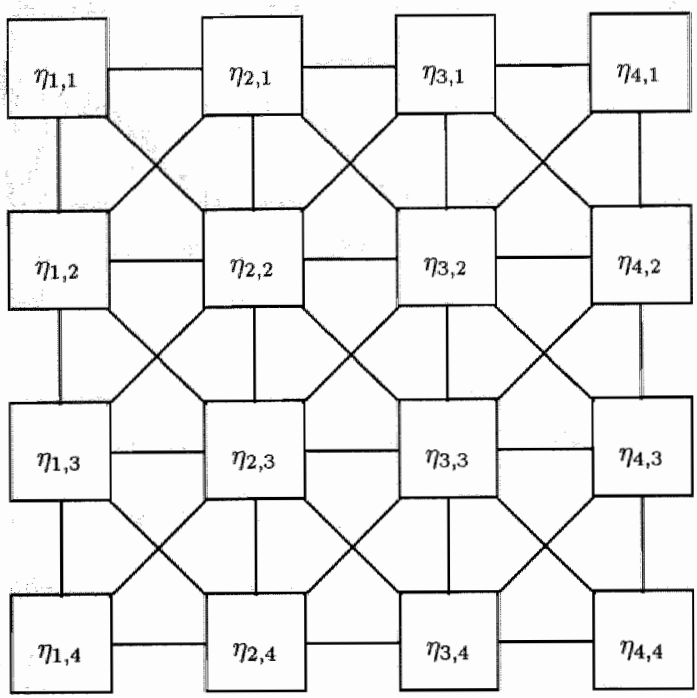


Figure 4.1: *A two-dimensional cellular neural network containing 4×4 cells represented by the squares. The lines indicate the local interaction structure between neighbouring cells.*

4.1.2 Functional Architecture

The functional architecture of a CNN is separated into the connection architecture and the neuron architecture as proposed in Section 1.2.2. After describing these two architectures the nature of the emerging activation dynamics is outlined. The description presented below is based on Chua and Yang's (1988a) seminal article.

Connection Architecture

Since the neurons in a CNN are identical they do not only share the same processing model but also have the same set of weights attached to their connections. In other ANN models, e.g. the MLNN and SOFM model, each neuron has its individual set of weights. Consequently, the connectivity matrix of a CNN can be obtained by cloning a single *weight template*, i.e. a description of the connection weights of a single neuron to its neighbours.

A cellular structure may appear in many different shapes. However, we will only consider CNNs with a two-dimensional grid-like organization, such as depicted in Figure 4.1 showing a 4×4 CNN. Connections between neurons are restricted to neighbouring neurons, i.e. the eight surrounding neurons, and itself. For instance, neuron $\eta_{2,2}$ is connected to neurons $\eta_{1,1}$, $\eta_{2,1}$, $\eta_{3,1}$, $\eta_{1,2}$, $\eta_{2,2}$, $\eta_{3,2}$, $\eta_{1,3}$, $\eta_{2,3}$ and $\eta_{3,3}$. Owing to this local interaction pattern, a CNN may be enlarged by adding neurons to the sides of the network. Moreover, Chua and Yang (1988a) note that two-dimensional CNNs may be interpreted as layers in a three-dimensional CNN having local connections between subsequent layers in a vertical direction. In such a way a two-dimensional CNN can be extended to higher dimensions without difficulty.

The *weight template* of a two-dimensional CNN specifies eighteen weights used for calculating the weighted-input sum of a neuron. Actually, these eighteen weights specify two operators containing nine weights each. The first one is a *feed-back operator* defining how input coming from the eight surrounding neurons and itself must be weighted. The second operator is called a *control operator* defining how a selected area from the input data must be weighted. In Figure 4.2 the connections implied by the control and feed-back operator are visualized.

Neuron Architecture

The processing model in the neuron architecture of CNNs contains three processing stages (cf. Table 1.2, example 5). The first stage calculates a weighted input sum, i.e. summing the weighted outputs of neighbouring neurons and itself resembling a *spatial integration*. A first-order differential equation in the second stage performs a *temporal integration*, i.e. integrat-

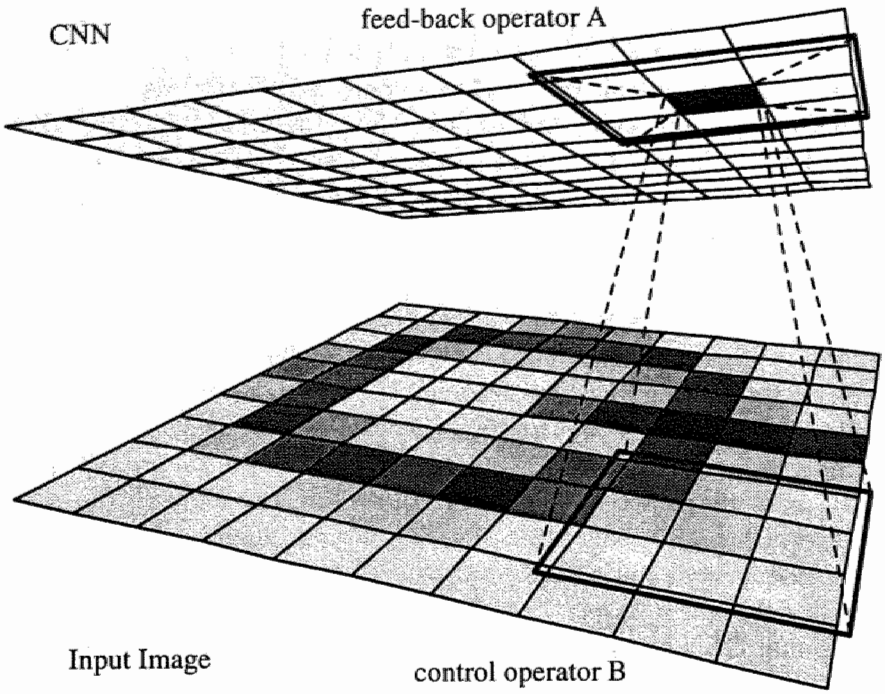


Figure 4.2: *Small squares in the upper plane represent CNN neurons. Small squares in the lower plane represent image pixels used as input for the CNN neurons. Both large squares indicate the receptive field of the black-coloured neuron. The large square in the plane of the CNN corresponds to the feed-back operator while the one in the lower plane corresponds to the control operator applied to* 3×3 *pixels.*

ing the output of the first stage in time resulting in the neuron's internal state. In the third stage the neuron state is transformed into the neuron output using a piece-wise linear sigmoid-like function. A detailed description of each stage is presented below.

Stage I

The weighted-input sum calculated in the first stage is determined by the

feed-back operator and control operator depicted in Figure 4.2. The feed-back operator is used to construct the connectivity matrix[2] **A** and the control operator is used to construct the input matrix **B**. Owing to the two-dimensional topology of the network, the elements in **A** and **B** are both indexed by a four-dimensional index. For instance, the weight of the connection from neuron η_{kl} to neuron η_{ij} is denoted $A(i,j;k,l)$. Chua and Yang (1988a) assume that the connectivity matrix is symmetric, i.e. $A(i,j;k,l) = A(k,l;i,j)$. Let us assume that the CNN has width W and height H in such a way that it contains $W \times H$ neurons. Furthermore, let us assume that the input is a grey-level image having the same dimensions as the CNN. For instance, in Figure 4.2 $W = H = 10$. Let ξ_{ij} denote the grey level of pixel (i,j) and let $y_{ij}(t)$ denote the output of neuron η_{ij} at time t. The weighted input $\sigma_{ij}(t)$ of neuron η_{ij} is calculated as follows

$$\sigma_{ij}(t) = \sum_{k=1}^{W} \sum_{l=1}^{H} A(i,j;k,l) y_{kl}(t) + B(i,j;k,l) \xi_{kl}(t) \tag{4.1}$$

The calculation of $\sigma_{ij}(t)$ corresponds to a process of spatial integration owing to the local connection pattern (cf. Figure 4.2).

Stage II

A first-order differential equation in the second stage controls the internal neuron state. The state is a *temporal integration* of the output of the first stage. The differential equation contains a decay factor and the effect of the input at time t will slowly leak away. Let $x_{ij}(t)$ denote the state of neuron η_{ij}, the second stage of the neuron processing model can then be described as follows:

$$\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + \sigma_{ij}(t) \tag{4.2}$$

Stage III

The third stage of the processing model consists of a piece-wise linear sigmoid-like output function depicted in Figure 4.3. The neuron output

[2] Throughout this chapter bold-face characters are used to denote vectors and matrices.

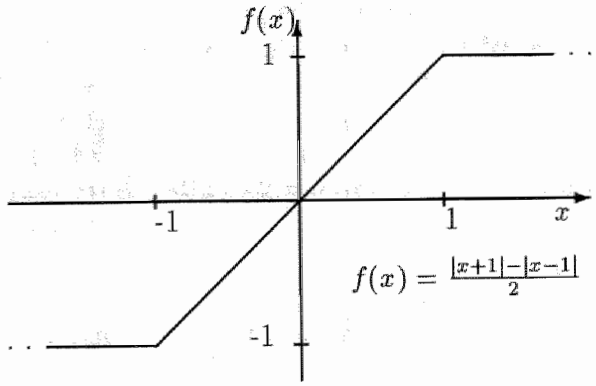


Figure 4.3: *Piece-wise linear output function used in CNNs.*

$y_{ij}(t)$ is -1 for $x_{ij}(t) < -1$, it is 1 for $x_{ij}(t) > 1$, while it is equal to $x_{ij}(t)$ when $-1 \leq x \leq 1$.

$$y_{ij}(t) = f(x_{ij}(t)) = \begin{cases} -1 & x_{ij}(t) < -1 \\ x_{ij}(t) & -1 \leq x_{ij}(t) \leq 1 \\ 1 & x_{ij}(t) > 1 \end{cases} \tag{4.3}$$

Activation Dynamics

After the connection architecture and the neuron architecture we describe the activation dynamics emerging from the specifications. Firstly, it should be noted that a CNN contains recurrent connections and hence that it is a dynamic system. The analysis of dynamic systems may be difficult, particularly when the systems are nonlinear such as CNNs. Chua and Yang (1988a) analyze the nonlinear (piece-wise linear) cell-circuit dynamics of a CNN using Lyapunov's method (Hirsch, 1989). Because the connectivity matrix **A** is *symmetric*, it is possible to show that in due time the CNN arrives at a stable state. This property is essential for the application of CNNs. Moreover, the internal neuron states $x_{ij}(t)$ are bounded, which is essential for a physical realization; if $x_{ij}(t)$ is *not* bounded, the power

dissipation of the network may become extremely large. Furthermore, Chua and Yang note that the activity dynamics of a CNN resemble heat diffusion when using an appropriate weight template. The neuron-processing model is then characterized by the heat equation from physics provided that the internal state remains in the linear regime of the output function.

4.1.3 Realization and Application

We conclude this section by emphasizing that in spite of the simple cellular structure CNNs can perform complex operations in real time. For this purpose both analog and digital CNN designs have been realized. The analog designs are based on a silicon-circuit implementation of the neuron architecture (Chua and Yang, 1988a; Halonen *et al.*, 1991). The digital designs are based on a parallel configuration of digital signal processors (Roska *et al.*, 1991) or on a two-dimensional array of, e.g. transputers (see Appendix A). Until now, only CNNs with few neurons have been realized in hardware. However, such hardware components can be combined into a larger CNN and hence their size is virtually unlimited.

Scalability is an advantage that makes, for instance, image preprocessing a very successful application area of CNNs. MLNNs and SOFMs have great difficulty when processing relatively small images, e.g. an image containing 100×100 pixels. In contrast, a CNN processes $1,000 \times 1,000$ pixel images without any problem. We hasten to add that the CNN is restricted to elementary image processing while the MLNN and the SOFM are capable of pattern recognition. Hence, a CNN should be used for preprocessing images compressing them to their essential features to be processed in a subsequent stage by, e.g. a MLNN or a SOFM. This approach is described in Section 4.3 after introducing a slightly different CNN in Section 4.2.

4.2 The Membrain Model

The functional architecture of the Membrain model strongly resembles the functional architecture of CNNs described in Section 4.1. The main difference between the CNN model introduced by Chua and Yang (1988a) and the Membrain model pertains to the characteristic processing equa-

tion which is based on a wave equation instead of a heat equation. The neurons in a Membrain CNN (MCNN) are *oscillators* and not integrators. The connection weights are configured in such a way that the states of the oscillating neurons are bounded. Hence, the piece-wise linear neuron characteristic function may be chosen such that it always operates in the linear regime. Consequently, a MCNN can be described by a system of coupled, *linear* second-order partial differential equations which can be solved analytically (Courant and Hilbert, 1953).

The line of our description is as follows. Firstly, we present an intuitive description of a MCNN introducing a vibrating-membrane metaphor. Analogous to heat diffusion on a heated surface illustrating the dynamics of a CNN, wave propagation on a vibrating membrane illustrates the dynamics of a MCNN. Secondly, the Membrain model is formalized and represented by a matrix equation. Finally, this equation is solved analytically revealing the two-dimensional and temporal nature of the emerging wave pattern.

4.2.1 The Vibrating-Membrane Metaphor

When a smooth water surface is stirred, wave fronts emerge which travel across the surface in all directions. The waves are scattered by reflection, for instance against a bank, and interfere with other waves. Such an emerging wave pattern can be viewed as a temporal representation of the initial stirring of the surface, while depending further on the actual boundary conditions.

In Figure 4.4 a simulation of such a system is shown in the form of an arbitrary pattern of travelling waves at four different time instances. For each time instance a grey-level image is displayed as well as a projection of the grid. In the image, black represents a surface deflection ≥ 1 whereas white represents a deflection ≤ -1. Figure 4.4(a) and (e) show the chosen *initial* deflection pattern of the network activity consisting of two peaks. Each peak acts as the source of a wave front which is propagated radially outwards as shown in Figure 4.4(b). In Figure 4.4(c) the transversally travelling waves have begun to interfere. After a while, waves are reflected at the boundary of the network and a complex pattern of continuously interfering waves evolves as shown in Figure 4.4(d). Figures 4.4(e)–(h) show

the two-dimensional projections of the surface at the same time instances. The simulation also illustrates the energy conservation which is typical for a wave equation (Boyce and DiPrima, 1965), that is, the tall peaks shown in Figure 4.4(e) are spread over the surface resulting in much lower amplitudes.

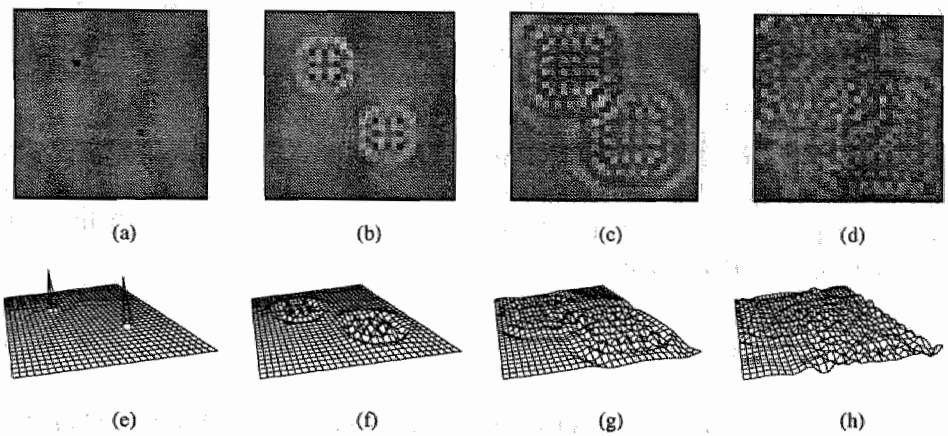


Figure 4.4: *Animation of travelling wave patterns evolving from an initial deflection at $t = 0$ shown in (a) through (d). Figures (e) through (h) show a perspective projection corresponding to the states depicted in (a) through (d), respectively.*

Initially, the system may be formalized by a two-dimensional wave equation describing the appearance of transversally travelling waves in a continuous vibrating medium, e.g. a membrane. However, the continuous wave equation may be *discretized* into small spatial intervals, yielding a system of coupled oscillators. The oscillators vibrate orthogonally to the surface, i.e. in a vertical direction if the surface is placed horizontally. Due to the (vertical) oscillations, waves emerge that travel *transversally* to the oscillating direction, i.e. in horizontal directions. In the next subsection we present an ANN model based on such a system of coupled oscillators. ANNs constructed according to this model resemble a vibrating membrane, since their dynamics correspond to wave propagation in a two-dimensional lattice. Therefore we call this model the *Membrain* model.

4.2.2 Heat Equation versus Wave Equation

Chua and Yang (1988a) have identified the relationship between CNNs and the heat equation from physics. This relationship is based on the system of partial differential equations underlying a CNN, assuming there is no external input, i.e. matrix **B** is void. The heat equation is

$$\frac{1}{k}\frac{du(x,y,t)}{dt} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} \tag{4.4}$$

Here k is a constant called the thermal conductivity. The solution $u(x,y,t)$ of the heat equation is a continuous function of the time variable t and the space variables x and y. If the function $u(x,y,t)$ is approximated by a set of functions $u_{ij}(t)$ defined as

$$u_{ij}(t) = u(ih, jh, t) \tag{4.5}$$

where h is a uniform space interval on the x and y coordinates, then the sum of the partial derivatives of $u(x,y,t)$ with respect to x and y can be replaced by

$$\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} =$$

$$\lim_{h \downarrow 0} \frac{1}{h^2}[u_{i-1j}(t)+u_{i+1j}(t)+u_{ij-1}(t)+u_{ij+1}(t)-4u_{ij}(t)] \tag{4.6}$$

This is essentially a first-order Taylor expansion of the right-hand side of Equation 4.4 in the spatial variables x and y. The relationship as identified by Chua and Yang (1988a) is that CNNs have a similar processing structure obtained by setting h in Equation 4.6 equal to 1 and defining the feed-back operator such that each neuron sums the output activities of its vertical and horizontal neighbours minus four times its own activity.

$$\frac{1}{k}\frac{du_{ij}(t)}{dt} = u_{i-1j}(t) + u_{i+1j}(t) + u_{ij-1}(t) + u_{ij+1}(t) - 4u_{ij}(t) \tag{4.7}$$

Now, instead of using a heat equation, we will consider a wave equation comparable to Equation 4.4. The difference between *heat diffusion* and *propagating waves* is that the left-hand side of the wave equation contains a second-order time derivative rather than a first-order time derivative:

$$\frac{1}{a^2}\frac{d^2u(x,y,t)}{dt^2} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} \tag{4.8}$$

The velocity of the waves on the membrane surface is $a = \sqrt{\tau/\rho}$, where τ stands for tension (force) per unit length and ρ stands for mass per unit area. Analogous to the heat equation, the wave equation underlying the continuous vibrating membrane may be discretized. The result is a system of coupled *oscillators*.

$$\frac{1}{a^2}\frac{d^2u_{ij}(t)}{dt^2} = u_{i-1j}(t) + u_{i+1j}(t) + u_{ij-1}(t) + u_{ij+1}(t) - 4u_{ij}(t) \tag{4.9}$$

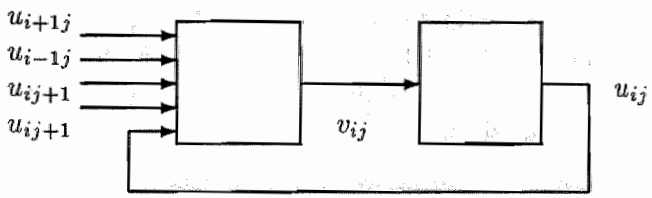


Figure 4.5: *Oscillator neuron constructed from two "integrator" neurons.*

This particular processing equation represents the processing mode of a MCNN with a *specific* template. In the general Membrain model defined below (cf. Equation 4.15) the right-hand side of Equation 4.9 may contain inputs from arbitrary neurons as long as the characteristic oscillatory properties of the network are maintained. It can be shown that the MCNN specified in Equation 4.9 is a CNN if and only if the CNN output formula is determined by a first-order differential equation (Roska and Chua, 1991) together with the assumption that the neuron state always remains in the linear part of the sigmoid output function. This is illustrated in Figure 4.5,

which depicts a second-order neuron processing equation (cf. Equation 4.9) as two first-order processing stages in series. Interpreting $u_{ij}(t)$ as neuron output $y_{ij}(t)$ and its first-order derivative $\dot{u}_{ij}(t)$ as internal neuron state $v_{ij}(t)$, Equation 4.9 may be described using the generalized CNN model (Roska and Chua, 1991) as

$$\begin{cases} \dfrac{dv_{ij}(t)}{dt} = u_{i-1j}(t) + u_{i+1j}(t) + u_{ij-1}(t) + u_{ij+1}(t) - 4u_{ij}(t) \\ \dfrac{du_{ij}(t)}{dt} = v_{ij}(t) \end{cases} \tag{4.10}$$

In contrast to a CNN, a Membrain neural network is described by a system of coupled *linear* differential equations. This means that an estimate of the dynamic range (Chua and Yang, 1988a; Roska and Chua, 1991) may be replaced by an accurate analytical solution using a standard mathematical procedure (Courant and Hilbert, 1953). For this purpose we introduce a vector notation representing the neuron states in a two-dimensional grid. Let us assume the grid consists of $W \times H$ neurons; then the double index (i, j) of a neuron is replaced by a single index n with $1 \leq n \leq N = WH$ such that $n = (j-1)W + i$. To illustrate the *sparse* connection structure, an example is presented below.

Example

For a network with 3×3 neurons the representation of the network activity looks as follows :

$$\begin{array}{|ccc|} \hline u_1(t) & u_2(t) & u_3(t) \\ u_4(t) & u_5(t) & u_6(t) \\ u_7(t) & u_8(t) & u_9(t) \\ \hline \end{array}, \qquad \mathbf{u}(t) = \begin{pmatrix} u_1(t) \\ \vdots \\ u_9(t) \end{pmatrix} \tag{4.11}$$

In this example we assume that the network has a *toroidal* connection structure, i.e. the neurons at the borders are connected to the neurons at the opposite border. Using a vector representation, the total system is

described by the following set of second-order linear differential equations:

$$\begin{cases} \frac{d^2u_1(t)}{dt^2} &= -4u_1(t) + u_2(t) + u_3(t) + u_4(t) + u_7(t) \\ \frac{d^2u_2(t)}{dt^2} &= u_1(t) - 4u_2(t) + u_3(t) + u_5(t) + u_8(t) \\ \frac{d^2u_3(t)}{dt^2} &= u_1(t) + u_2(t) - 4u_3(t) + u_6(t) + u_9(t) \\ \frac{d^2u_4(t)}{dt^2} &= u_1(t) - 4u_4(t) + u_5(t) + u_6(t) + u_7(t) \\ \frac{d^2u_5(t)}{dt^2} &= u_2(t) + u_4(t) - 4u_5(t) + u_6(t) + u_8(t) \\ \frac{d^2u_6(t)}{dt^2} &= u_3(t) + u_4(t) + u_5(t) - 4u_6(t) + u_9(t) \\ \frac{d^2u_7(t)}{dt^2} &= u_1(t) + u_4(t) - 4u_7(t) + u_8(t) + u_9(t) \\ \frac{d^2u_8(t)}{dt^2} &= u_2(t) + u_5(t) + u_7(t) - 4u_8(t) + u_9(t) \\ \frac{d^2u_9(t)}{dt^2} &= u_3(t) + u_6(t) + u_7(t) + u_8(t) - 4u_9(t) \end{cases} \tag{4.12}$$

This system of equations can be described as a matrix equation by introducing a *connectivity matrix* **A**. The matrix for the toroidal connection structure (using the template defined in Equation 4.9) for a 3×3 network is given by

$$\mathbf{A} = \begin{pmatrix} -4 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & -4 \end{pmatrix} \tag{4.13}$$

We draw attention to the sparseness and symmetry (both along the main and other diagonal) of this matrix.

End of the Example

In general, using the connectivity matrix $\mathbf{A}$ and the vector notation $\mathbf{u}(t)$ for the network state, the system of coupled linear differential equations (cf. Equation 4.9) may be written as

$$\frac{d^2\mathbf{u}(t)}{dt^2} = \mathbf{A}\mathbf{u}(t) \tag{4.14}$$

This differential equation is *homogeneous*, i.e. the system described by the equation receives no external input. However, we may generalize this processing equation by introducing an external input $\xi(t)$ and a damping coefficient α. In addition to the interacting forces between neurons, the acceleration of $\mathbf{u}(t)$ is determined by $\xi(t)$, which as an external driving force "pushes" against the membrane. The input is scattered locally over the surface by a control operator denoted by input matrix $\mathbf{B}$, analogously to the input entering a CNN (cf. Equation 4.1). Moreover, the coefficient α can be viewed as indicating the amount of absorption of travelling waves by the system. Thus, the *general* Membrain model is defined as

$$\frac{d^2\mathbf{u}(t)}{dt^2} = -\alpha\frac{d\mathbf{u}(t)}{dt} + \mathbf{A}\mathbf{u}(t) + \mathbf{B}\xi(t) \tag{4.15}$$

Matrix $\mathbf{A}$ is the connectivity matrix describing a Membrain connection structure with arbitrary dimensions. We note that $\mathbf{A}$ is always a *square* matrix. Moreover, we require that the connections be real-valued and symmetric, i.e. the connection between the neurons i and j is the same as the opposite connection between j and i. Hence, $\mathbf{A}$ is always a square, symmetrical matrix.

4.2.3 Analytical Solution

The system of partial differential equations described in Equation 4.15 may be solved by a linear transformation of $\mathbf{A}$. Since $\mathbf{A}$ is symmetric, there exists an orthogonal projection matrix $\mathbf{E}$ so that matrix $\mathbf{\Gamma} = \mathbf{E}\mathbf{A}\mathbf{E}^{-1}$ is diagonal, i.e. $\mathbf{A}$ is *orthogonally diagonizable*. This is a well known mathematical

property of symmetric matrices (see for instance Lipschutz (1991)). We can choose the rows of matrix $\mathbf{E}$ to be normalized orthogonal *eigenvectors* of $\mathbf{A}$, then $\mathbf{E}^{-1} = \mathbf{E}^{\mathrm{T}}$ and the diagonal elements of $\mathbf{\Gamma}$ are the corresponding eigenvalues.

By using this property of symmetric matrices, Equation 4.15 may be transformed into a system of *uncoupled* equations by a projection of $\mathbf{u}(t)$ into the distinct orthogonal spaces labelled by the eigenvalues of $\mathbf{A}$. This projection is accomplished by an operator $\mathbf{E}$, which is a matrix whose rows correspond to N orthornormal eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_N$ of matrix $\mathbf{A}$ ($\mathbf{e}_i^{\mathrm{T}}\mathbf{e}_j = \delta_{ij}$). Let the distinct eigenvalues of $\mathbf{A}$ be $\lambda_1, \ldots, \lambda_n$ and let $\mathcal{L}_1, \ldots, \mathcal{L}_n$ be all linear subspaces of $\mathbf{A}$ such that $\mathcal{L}_i$ corresponds to $\{\mathbf{x}|\mathbf{A}\mathbf{x} = \lambda_i\mathbf{x}\}$. Hence the dimension of $\mathcal{L}_i$ is determined by the multiplicity of λ_i. Let $\mathbf{\Gamma}$ be a diagonal matrix with diagonal elements $\gamma_1, \ldots, \gamma_N$ so that $\gamma_i \in \{\lambda_1, \ldots, \lambda_n\}$ and $\mathbf{e}_i \in \mathcal{L}_j$ if and only if $\gamma_i = \lambda_j$. Then $\mathbf{A}$ may be written as:

$$\mathbf{A} = \mathbf{E}^{\mathrm{T}}\mathbf{\Gamma}\mathbf{E}, \quad \mathbf{E}^{\mathrm{T}} = [\mathbf{e}_1, \ldots, \mathbf{e}_N], \quad \mathbf{\Gamma} = \mathrm{diag}\{\gamma_1, \ldots, \gamma_N\} \tag{4.16}$$

Equation 4.15 may be transformed into a system of N *uncoupled* second-order differential equations by multiplying both sides with $\mathbf{E}$ and substituting $\mathbf{E}\mathbf{u}(t)$ by $\mathbf{x}(t)$ and $\mathbf{E}\mathbf{B}\boldsymbol{\xi}(t)$ by $\mathbf{y}(t)$:

$$\frac{d^2\mathbf{x}(t)}{dt^2} = -\alpha\frac{d\mathbf{x}(t)}{dt} + \mathbf{\Gamma}\mathbf{x}(t) + \mathbf{y}(t) \tag{4.17}$$

Because $\mathbf{\Gamma}$ is a diagonal matrix, the components of $\mathbf{x}(t)$ are not coupled and their solutions may be derived independently. In Appendix B the second-order linear differential equation described by Equation 4.17 is solved for an arbitrary component $x(t)$ of $\mathbf{x}(t)$ using a Laplace transform. Subsequently, by the inverse transformation the general solution $\mathbf{u}(t)$ of Equation 4.15 may be obtained from $\mathbf{x}(t)$ by multiplication with $\mathbf{E}^{\mathrm{T}}$:

$$\mathbf{u}(t) = \mathbf{E}^{\mathrm{T}}\mathbf{x}(t) = \sum_{i=1}^{N} x_i(t)\mathbf{e}_i \tag{4.18}$$

We note that the result in Appendix B shows three different regimes for the solution of Equation 4.15. We will only consider the regimes where $\forall i$:

$\gamma_i \leq 0$, because if $\gamma_i > 0$, non-oscillating, exponential solutions also exist. As an illustration we derive the solution of a freely vibrating MCNN, i.e. there is no damping and the control operator $\mathbf{B}$ is void. Alternatively, input is presented to the network by coding it in the system's initial state, i.e. in $\mathbf{u}(0)$. The solution can be obtained from the general solution by letting $\alpha = 0$ and $\xi(t) = 0$. In other words we return here to the homogeneous system described by Equation 4.14.

Let $\dot{\mathbf{u}}(t)$ be the first-order time derivative of $\mathbf{u}(t)$. Using the inverse transformation and the result for $x_i(t)$ (cf. Appendix B) for $\alpha = 0$ and frequency $\omega_i = \sqrt{-\gamma_i}$ the (piece-wise linearized) wave $\mathbf{u}(t)$ can be written explicitly as

$$\mathbf{u}(t) = \sum_{i=1}^{N} \left(\frac{1}{\omega_i} \sin(\omega_i t)\dot{x}_i(0) + \cos(\omega_i t) x_i(0) \right) \mathbf{e}_i \tag{4.19}$$

We recall that $\mathbf{x}(t)$ was obtained from $\mathbf{u}(t)$ by projection with operator $\mathbf{E}$. Hence the initial values $x_i(0)$ and $\dot{x}_i(0)$ can be determined

$$x_i(0) = \mathbf{e}_i^{\mathrm{T}} \mathbf{u}(0), \quad \dot{x}_i(0) = \mathbf{e}_i^{\mathrm{T}} \dot{\mathbf{u}}(0) \tag{4.20}$$

We conclude this section with a visualization of the mathematical analysis presented above. Figure 4.6 depicts a pattern corresponding to one of the *eigenvectors* of a connection matrix describing a 10×10 network. Such a pattern is called a *natural mode of vibration* which vibrates with its *natural frequency*. When a single natural vibration is present at the surface of a membrane, there are no transversally travelling waves on the surface. Such a natural mode of vibration corresponds to a *standing wave*. A *travelling wave*, however, is a *multimode oscillation*, i.e. a superposition of standing waves having different frequencies. Indeed, when a superposition of several natural vibrations with distinct frequencies is present, the network of oscillators gives the impression of waves being propagated along the surface (cf. Figure 4.4). Superposition of distinct natural vibration modes is an interpretation of the linear transformation effected by operator $\mathbf{E}$.

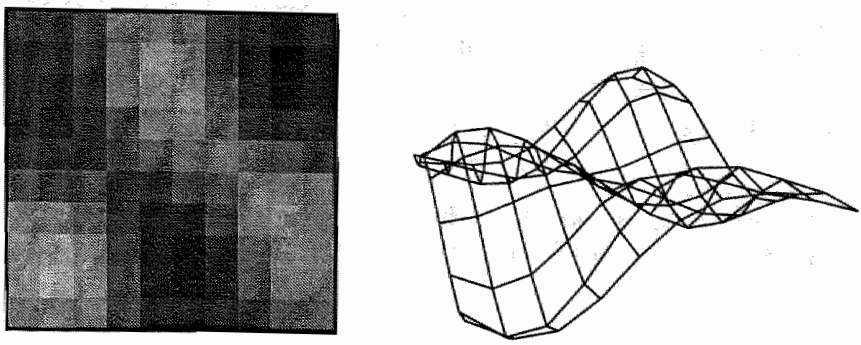


Figure 4.6: *A natural vibration mode with frequency 1.0078 in a* 10×10 *network.*

4.3 Application of the MCNN

A well-known problem from pattern recognition concerns recognition of patterns regardless of their translation. A pixel representation is not fit for this purpose because a translation of the image also effects the pixel representation. In this section, an invariant representation that does not change under translation is described. It is based on the toroidal connection structure described in the example presented in the Section 4.2 (cf. Equation 4.13). Simulation results are presented showing that the toroidal MCNN can be combined with a traditional neural network for translation-invariant pattern classification.

4.3.1 Translation Invariance

Consider again matrix $\mathbf{A}$ presented in Equations 4.13 and 4.14. We recall that $\mathbf{A}$ describes the toroidal connection structure of a two-dimensional network. Since the surface of a torus is periodic and the cell structure is uniform, it is evident that the whole connection structure as expressed by matrix $\mathbf{A}$ is invariant under any translation over the grid points of the network. Treating the subject formally, we introduce a translation operator $\mathbf{T}$ being a permutation matrix, i.e. each row and column of $\mathbf{T}$ contain

exactly one 1 while all other elements are 0. The inverse translation $\mathbf{T}^{-1}$ is $\mathbf{T}^{\mathrm{T}}$, i.e. the transpose of the permutation matrix. The invariance of $\mathbf{A}$ under translation may be expressed as

$$\mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \mathbf{A} \tag{4.21}$$

Using this property, we can show that the linear subspace $\mathcal{L}_\lambda$ that corresponds to $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ is closed under the translation operation. Let $\mathbf{e}$ be any eigenvector of $\mathbf{A}$:

$$\mathbf{A}\mathbf{e} = \lambda\mathbf{e} \tag{4.22}$$

Translation of $\mathbf{e}$ yields $\mathbf{T}\mathbf{e}$. Using Equation 4.21, we can show that :

$$\mathbf{A}\mathbf{T}\mathbf{e} = \mathbf{T}\mathbf{A}\mathbf{e} = \mathbf{T}\lambda\mathbf{e} = \lambda\mathbf{T}\mathbf{e} \tag{4.23}$$

Consequently, although $\mathbf{e}$ is translated, it remains associated with the same eigenvalue λ. Moreover, depending on the amount of point-group symmetry of the network, this is also true for rotations. For instance, a square network is also invariant under rotation over 90, 180 and 270 degrees.

4.3.2 Toroidal MCNN

In the case of a MCNN we can achieve translation-invariant classification by classifying the temporal representation of the input, i.e. the network's initial state. For this reason we require the connection structure of the network to be toroidal. If the structure is not toroidal but, e.g. flat, then the network surface will not be periodic. Having boundaries, any translation of $\mathbf{u}(0)$ will result in a different wave pattern owing to different reflections against the network boundaries. In a toroidal connection structure, however, the interference pattern will be translated but the pattern itself will be unchanged.

This property results from the fact that the linear subspace $\mathcal{L}_i$ that corresponds to $\mathbf{A}\mathbf{x} = \lambda_i\mathbf{x}$ is closed under the translation operation as shown in

the previous subsection (cf. Equation 4.23). In order to explore this property, we will examine a freely vibrating, toroidal MCNN, i.e. no damping and no external force (cf. the processing model in Equation 4.14 and analytical solution in Equation 4.19). If we assume additionally that the initial oscillator velocity $\dot{\mathbf{u}}(0) = 0$ and the initial deflection is $\mathbf{u}(0)$, then $\mathbf{u}(t)$ becomes

$$\mathbf{u}(t) = \sum_{i=1}^{N} \mathbf{e}_i^{\mathrm{T}} \mathbf{u}(0) \mathbf{e}_i \cos(\omega_i t) \tag{4.24}$$

The number of terms in Equation 4.24 may be reduced if eigenvalues with multiplicity greater than unity exist. Let us denote the projection of the initial deflection $\mathbf{u}(0)$ on $\mathcal{L}_i$ by $\mathbf{p}_i$. Moreover, let $\nu_i = \sqrt{-\lambda_i}$; then Equation 4.24 can be written as a *natural mode expansion* of the system.

$$\mathbf{u}(t) = \sum_{i=1}^{n} \mathbf{p}_i \cos(\nu_i t) \tag{4.25}$$

Since $\mathcal{L}_i$ is closed under translation, the projection of $\mathbf{Tu}(0)$, i.e. the translation of $\mathbf{u}(0)$, on $\mathcal{L}_i$ equals $\mathbf{Tp}_i$. This is a fine result, since the behaviour of the system with initial deflection $\mathbf{Tu}(0)$ is $\mathbf{Tu}(t)$, i.e. the interference pattern of waves has been translated without destroying its shape.

4.3.3 Translation-Invariant Representation

We note that the projections $\mathbf{p}_1, \ldots, \mathbf{p}_n$ (cf. Equation 4.25) form a complete description of $\mathbf{u}(0)$. Because they result from a projection on linear spaces which are closed under translation, they are useful for invariant pattern classification. The square Euclidean length of the projections, i.e. $||\mathbf{p}_1||^2, \ldots, ||\mathbf{p}_n||^2$, is a *power spectrum* of the natural modes of vibration and may be interpreted as a "fingerprint" of $\mathbf{u}(0)$. By "fingerprint" we mean that the description is not complete, i.e. it will not be sufficient to reconstruct a pattern but it is sufficient to *discriminate* between different patterns. Moreover, because of translation invariance of $\mathcal{L}_1, \ldots, \mathcal{L}_n$, the "fingerprint" also does not change when $\mathbf{u}(0)$ is translated. This follows

from the observation that $||\mathbf{T}\mathbf{p}_i|| = ||\mathbf{p}_i||$, i.e. the length of $\mathbf{p}_i$ is constant under translation. Hence the *power spectrum* is a translation-invariant representation of $\mathbf{u}(0)$.

We will now show that the power spectrum of $\mathbf{u}(0)$ in terms of $\mathbf{p}_1, \ldots, \mathbf{p}_n$ is temporally encoded in a one-dimensional signal $\hat{u}(t) = ||\mathbf{u}(t)||^2$:

$$\begin{aligned}
\hat{u}(t) &= ||\mathbf{u}(t)||^2 = \sum_{k=1}^{N} u_k^2(t) \\
&= \sum_{k=1}^{N} \sum_{i=1}^{n} p_{ik} \cos(\nu_i t) \sum_{j=1}^{n} p_{jk} \cos(\nu_j t) \\
&= \sum_{i,j=1}^{n} \cos(\nu_i t) \cos(\nu_j t) \underbrace{\sum_{k=1}^{N} p_{ik} p_{jk}}_{=0 \text{ if } i \neq j} \\
&= \sum_{i=1}^{n} ||\mathbf{p}_i||^2 \cos^2(\nu_i t) \qquad (4.26)
\end{aligned}$$

It transpires that $\hat{u}(t)$ is also invariant under a translation of $\mathbf{u}(0)$. From the definition of $\mathcal{L}_i$ it follows that the frequencies $\nu_1, \ldots, \nu_n$ are all distinct. Hence, $||\mathbf{p}_i||^2$ is modulated in $\hat{u}(t)$ with a unique frequency. Consequently, analysis of $\hat{u}(t)$, e.g. by using a Fourier Transform, yields a static, i.e. non-temporal, translation-invariant "fingerprint" of $\mathbf{u}(0)$.

The cellular structure enables a MCNN to process large images. Since each cell requires only four connections, a simple connection structure suffices, e.g. a matrix structure. Moreover, when other non-digital solutions are considered, e.g. analog VLSI, optics, fluid mechanics, etc., it is useful to consider that $\hat{u}(t)$ can be calculated from the elastic energy of the system.

4.3.4 Simulations

A MCNN with a toroidal connection structure has been applied to generate translation-invariant representations of *handwritten signature images*. Three persons were asked to draw their signature four times at different

positions within a 2 by 1 inch^2 area. The 12 different images are depicted in Figure 4.7. Figures 4.7(a)-(d), (e)-(h), and (i)-(l) correspond to persons 1, 2 and 3 respectively. We emphasize that small differences between the signatures of a single person exist because they are drawn at different times.

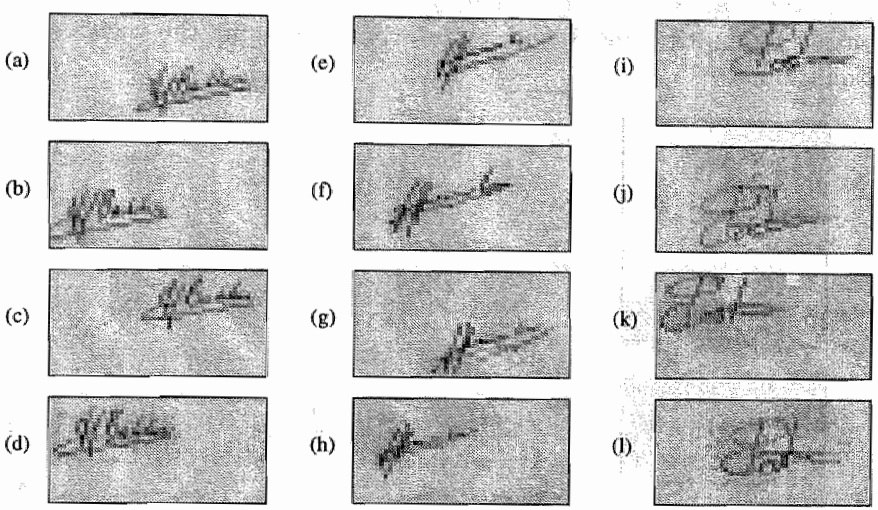


Figure 4.7: *Membrain initial deflection patterns representing signatures from three different persons. Each person was asked to draw his signature four times : (a)-(d) person 1, (e)-(h) person 2, and (i)-(l) person 3.*

The images were scanned at 25 dots per inch, resulting in 50×25 pixels at 256-grey levels, i.e. 1 byte per pixel. Each image is normalized and subsequently presented to a 50×25 Membrain with a toroidal connection structure as in the example of Section 4.2.2. Simulations of the MCNN were performed using a fourth- and fifth-order Runge-Kutta-Fehlberg integration procedure with a variable step size and a tolerance of 10^{-5}.

From the analysis of wave equations (Courant and Hilbert, 1953) we know that spatial frequency and temporal frequency are equivalent. Hence, in order to obtain an indication of the bandwidth of $\hat{u}(t)$ for a 50×25 MCNN, the lowest and the highest frequency are measured by presenting a low-

and a high-spatial-frequency pattern to the network, respectively. In Figures 4.8(a) and 4.8(b) two low-spatial-frequency patterns are depicted with their corresponding signals. Figure 4.8(a) corresponds to 0.04 Hz, which is the lowest frequency in $\hat{u}(t)$. In Figure 4.8(c) a high spatial-frequency pattern, i.e. a checker-board, is presented, resulting in a frequency of 0.90 Hz, i.e. the highest frequency in $\hat{u}(t)$. We note that these frequencies can be scaled by multiplying $\mathbf{A}$ with a scalar, i.e. changing the wave velocity a in Equation 4.9.

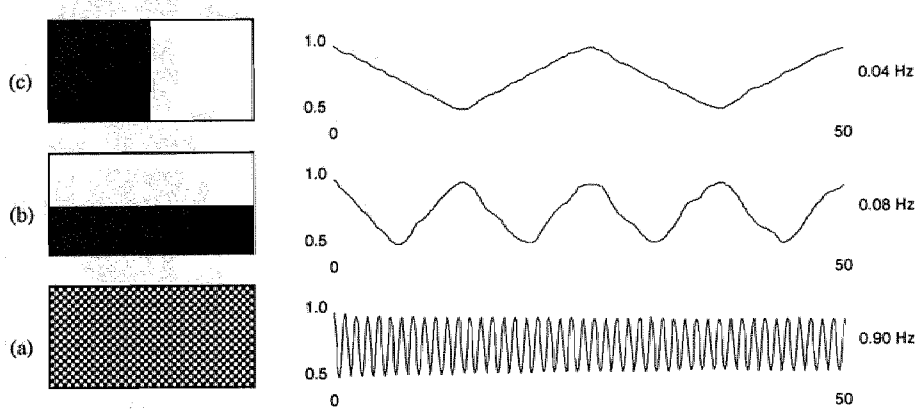


Figure 4.8: *Three typical frequencies in $\hat{u}(t)$ of a 50×25 MCNN. Spatial frequency is related to temporal frequency: (a) lowest-spatial-frequency pattern in horizontal direction corresponds to lowest frequency 0.04 Hz, (b) lowest-spatial-frequency pattern in vertical direction corresponds to frequency 0.08 Hz, and (c) highest-spatial-frequency pattern corresponds to highest frequency 0.90 Hz.*

Network simulations for the signature images were performed for $t \in [0, 50]$, resulting in the curves $\hat{u}_1(t), \ldots, \hat{u}_{12}(t)$ shown in Figure 4.9(a)-(l) corresponding to Figure 4.7(a)-(l), respectively. In order to process the output signals for further classification, $\hat{u}_i(t)$ will be sampled. Since the highest frequency in $\hat{u}_i(t)$ is 0.90 Hz, the sampling-time interval should be smaller than $1/2f_{max} = 0.56$ seconds. Furthermore, the performance of the system depends on the number of samples used for classification. Ideally,

this number is determined by the distribution of the harmonic components of $\hat{u}(t)$ in the frequency spectrum. When fewer samples are used, small frequency differences cannot be distinguished, i.e. it seems as if the network has smaller dimensions. When more samples are taken, classification performance will not increase significantly since the number of harmonic components is limited.

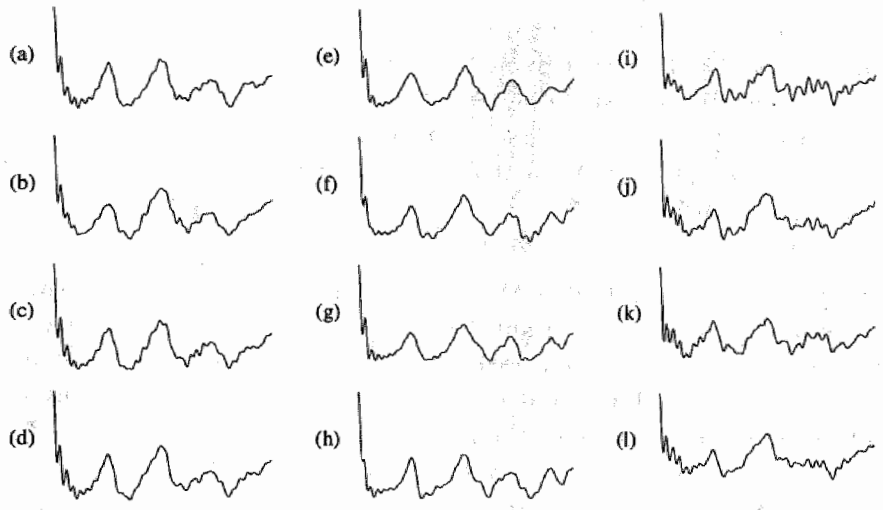


Figure 4.9: *Output signals corresponding to $\hat{u}(t)$ of the wave pattern for each initial deflection pattern shown in Figure 4.7 (a)-(l), respectively, from $t = 0$ to $t = 50$ seconds.*

We have roughly estimated the number of required samples for the signals presented in Figure 4.7 at 250 by counting the number of distinct spatial frequencies that can exist without aliasing in the network. Hence, each signal is sampled using $\Delta t = 0.2$s for $t \in [0, 50]$ resulting in 12 vectors $\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{12}$ with $\hat{u}_{ij} = \hat{u}_i(j\Delta t)$. We have analyzed the vectors using two different approaches.

Firstly, it can be shown that two signals corresponding to the same person have a higher correlation than any two signals corresponding to different persons. Let $\mathbf{X}$ be the correlation matrix of the normalized vectors

$\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{12}$, i.e.

$$x_{ij} = \frac{\hat{\mathbf{u}}_i^T \hat{\mathbf{u}}_j}{||\hat{\mathbf{u}}_i|| ||\hat{\mathbf{u}}_j||} \tag{4.27}$$

The columns of $\mathbf{X}$, i.e. $\mathbf{x}_1, \ldots, \mathbf{x}_{12}$, are reduced representations of $\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{12}$, respectively. We have clustered $\mathbf{x}_1, \ldots, \mathbf{x}_{12}$ using a two-dimensional Self-Organizing Feature Map (Kohonen, 1984) (see also Chapter 3) consisting of 10×10 neurons. The resulting two-dimensional representation of the clusters is depicted in Figure 4.10. This representation shows that signals corresponding to the same person, e.g. (a)–(d), have a stronger correlation than signals corresponding to different persons, e.g. (a) and (k). We note in passing that pixel correlation of the images for the purpose of classification would be *quite meaningless* owing to translation of the signature within different images.

Secondly, we will show that a MLNN using Back Propagation is capable of extracting translation-invariant features characterizing a signature. The vectors $\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{12}$ have been presented to a MLNN with 250 input neurons, 20 hidden neurons and 3 output neurons. The output neurons are used for classification of the input into three different classes, viz. person 1, 2 and 3. In order to examine the generality of the extracted features, a learning set and a test set were constructed. The signatures shown in Figure 4.7(a), 4.7(b), 4.7(e), 4.7(f), 4.7(i) and 4.7(j) are used as examples for the test set to see how well the network has generalized from the examples in the learning set, i.e. Figure 4.7(c), 4.7(d), 4.7(g), 4.7(h), 4.7(k) and 4.7(l). After approximately 8,000 cycles (learning rate and momentum are both 0.1) the average of the squared errors on the learning set is approximately 0.0003 and on the test set it is approximately 0.007 (cf. Figure 4.11). The errors of the individual examples in the test set and the learning set are shown in Table 4.1 A and 4.1 B, respectively. We note that the average of the squared errors (cf. Figure 4.11) has been calculated by averaging the squared differences between the output and target values for all output neurons and for all examples in either the learning set (solid curve) or the test set (dashed curve).

Although the network has not previously "seen" the examples in the test set, it can distinguish sufficiently between signatures of different persons

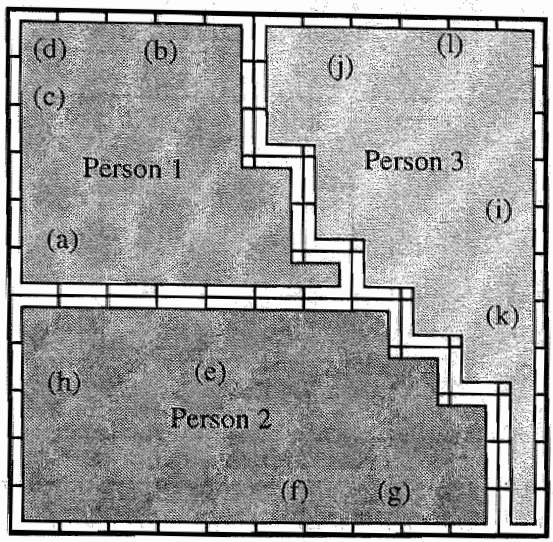


Figure 4.10: *Clustering of the correlations between the output signals depicted in Figure 4.9 (a)-(l) using a two-dimensional* 10×10 *Kohonen Self-Organizing Feature Map (neighbourhood size is 4, learning rate is 0.3).*

while generalizing over the slight variations in a signature if the samen person (cf. Table 4.1). Apparently, the network has learned the essential translation-invariant features which distinguish the signature of one person from any other person. We note that normally translation invariance is difficult to learn with a back-propagation scheme.

4.4 Chapter Summary

The neurons in a CNN are characterized by their relation with the heat equation, known from physics, combined with a piece-wise linear sigmoid output function. When the heat equation is replaced by a wave equation, the heat diffusion in the network becomes wave propagation. This modified CNN model is called the Membrain model since its dynamics resemble the

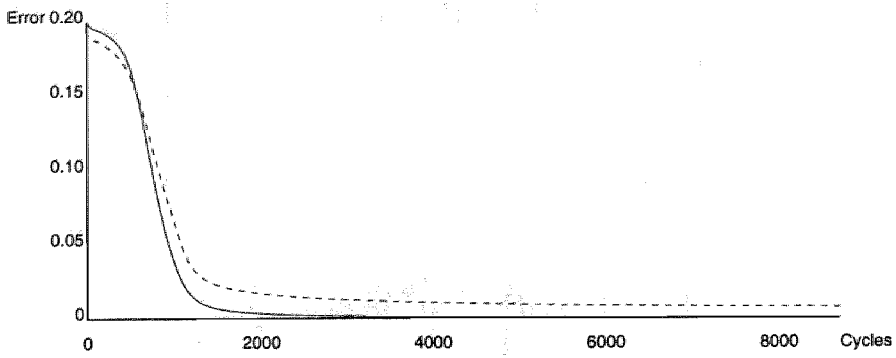


Figure 4.11: *Curves for the average of squared errors for the learning set (solid curve) and the test set (dashed curve). Back-propagation learning was applied to a MLNN with 250 input neurons, 20 hidden neurons and 3 output neurons (learning rate and momentum were both set to 0.1).*

vibrations in a vibrating membrane. The vibrations in a freely vibrating Membrain neural network are subject to the law of energy conservation since the neurons resemble harmonical oscillators. Hence, the neuron states in a Membrain neural network are *bounded* and the piece-wise linear output function of Membrain neurons can be chosen in such a way that it always operates in the linear regime.

The general Membrain model is described by a set of linear second-order differential equations and is not necessarily restricted to a cellular connection structure. An analytical solution is presented showing that the network behaviour is a multimode oscillation with the appearance of transversally travelling waves. The natural modes of a Membrain neural network are determined by the connectivity matrix $\mathbf{A}$. We have introduced a Membrain CNN (MCNN) having a cellular connection structure in which each neuron is influenced by a force equal to the sum of the four neighbouring activations minus four times its own activation. Moreover, the proposed MCNN has a toroidal geometry, i.e. neurons at the edges are connected

A: Results of the Test Set						
Image	Person 1		Person 2		Person 3	
(Fig. 4)	Output	Target	Output	Target	Output	Target
(a)	0.72	1.00	0.15	0.00	0.02	0.00
(b)	0.89	1.00	0.03	0.00	0.04	0.00
(e)	0.07	0.00	0.88	1.00	0.01	0.00
(f)	0.04	0.00	0.96	1.00	0.00	0.00
(i)	0.01	0.00	0.02	0.00	0.98	1.00
(j)	0.05	0.00	0.02	0.00	0.92	1.00

B: Results of the Learning Set						
Image	Person 1		Person 2		Person 3	
(Fig. 4)	Output	Target	Output	Target	Output	Target
(c)	0.97	1.00	0.02	0.00	0.01	0.00
(d)	0.98	1.00	0.02	0.00	0.01	0.00
(g)	0.01	0.00	0.97	1.00	0.03	0.00
(h)	0.02	0.00	0.99	1.00	0.00	0.00
(k)	0.02	0.00	0.01	0.00	0.97	1.00
(l)	0.01	0.00	0.02	0.00	0.98	1.00

Table 4.1: *Results of back-propagation learning for a MLNN consisting of three layers containing 250 input, 20 hidden and 3 output neurons. The labels in the first column correspond to the labelling of the images in Figure 4.9. Table A shows the errors for the examples in the test set after 8,000 cycles through the learning set. The errors for the examples in the learn set are presented in Table B. The learning rate and the momentum were set to 0.1.*

to neurons at the opposite edge. We have proven that linear subspaces of **A** corresponding to distinct eigenvalues are closed under image trans-

lation. Hence, the *power spectrum* of the temporal wave pattern can be interpreted as a translation invariant "fingerprint" of the deflection pattern (input pattern) of the network at $t = 0$. Through the symmetric connection structure a one-dimensional modulation of the power spectrum can easily be obtained by summing the squared neuron states, i.e. membrane deflections. Simulations were performed confirming that the temporal signal is useful for: (1) translation-invariant correlation of images, and, (2) extraction of translation-invariant features for image classification.

Summarizing, we can distinguish two different features of the organization of representation in the Membrain. Firstly, information is represented as a multimode oscillation, i.e. a superposition of natural modes in which each mode resonates at its natural frequency. Secondly, information is represented by the shape of natural modes that are determined by the weight matrix and the boundary conditions of the network. Strengthened by the research of Ingber and Nunez (1990) and Pribram (1991) we conclude that resonance and boundary conditions are the main elements in the temporal representation of an ANN with an oscillating activation pattern.

Chapter 5

Conclusion and Evaluation

In this chapter we evaluate our research and draw some conclusions on the research problems and the main problem as stated in Chapter 1. In Section 5.1 conclusions on the three research problems are presented. Additionally to each conclusion, the potential of the results for future ANN research and applications is analyzed. In Section 5.2, the main problem statement on the organization of representation in ANNs is addressed. The choices for the models described in Chapter 2, 3 and 4 are considered and the organization of representation in each ANN model is discussed. Finally, in Section 5.3 the main notions of this dissertation are evaluated.

5.1 Conclusion on the Problem Statements

In Chapter 1 three research problems were stated. They focus on MLNNs with complex-valued weights, a modular system containing multiple SOFMs and a network of harmonic oscillators. In this section the conclusions to the problems are presented. Each conclusion will be followed by a discussion on properties of the presented models and used methods in view of their potential for future ANN research and applications.

5.1.1 Complex-Valued MLNNs

The first research problem concerns the introduction and use of complex-valued weights in the connection architecture of MLNNs to *improve input-output mappings*. From the results in Chapter 2 we conclude that such an improvement is realized for the robot application by coding directions as complex values. The success of the complex-valued MLNN is because of the chosen complex-valued representation taking into account the correlation between $\sin(\alpha)$ and $\cos(\alpha)$. A real-valued MLNN does not contain this correlation.

The research has yielded two new techniques that are relevant for future ANN research. Firstly, by introducing a *vector notation* we have obtained a compact representation of the MLNN processing model and the equations underlying the BP algorithm. Secondly, for a not complex-differentiable neuron-output function we have derived a *new form of the generalized-delta rule* based on the partial derivatives of the output function. This form encompasses both complex-differentiable as well as multi-dimensional real-valued neuron-output functions. The application of both techniques in Chapter 2 may have an exemplary meaning in case other neuron-processing models are considered.

5.1.2 The Modular SOFM and its Visualization

The second research problem concerns the combination of multiple SOFMs. The research is inspired by the modular, two-dimensional organization of the primary cortex in the brain. The 2D SOFM (Kohonen, 1982) is an ANN approximation of a computational map as found in the primary cortex. Moreover, the SOFM model contains a self-organizing mechanism enabling the automatic construction of feature maps. Analogously to the primary cortex we introduce a *modular SOFM containing multiple 2D SOFMs*. Rather than presenting all input signals to one SOFM, we propose to separate non-correlated input signals presenting them as input to different maps. This strategy guarantees that the dimension reduction required from each map is feasible. Our conclusion is that a combination of multiple 2D SOFMs should be organized as a modular system in which each module corresponds to a map receiving correlated input signals only.

The performance of a MLNN is visualized by an error curve. Such a measure is not available for SOFMs and the self organization is difficult to observe. Hence, we designed a number of techniques to *visualize the representation of information in a 2D SOFM.* These techniques enable us to see easily how well the input examples are represented on the map and to study a three-dimensional projection of the internal organization. Inspecting the SOFM's internal organization provides a measure of accuracy. Moreover, it increases the experience for adjusting parameters, e.g. learning rate and neighbourhood size, in a particular application.

5.1.3 The Membrain Model

The third research problem deals with the nature of ANNs with an oscillating-activation pattern. In Chapter 4 this problem has been restricted to the Membrain model and experiments were performed with the MCNN. In contrast to the non-linear processing model of CNNs, the linear processing model of the MCNN can be solved analytically if the eigenvectors and eigenvalues of the connectivity matrix are known. From the analytical solution presented in Chapter 4 we conclude that the activation pattern of the MCNN is *a modulation of natural modes* determined by the connectivity matrix and the network input.

By calculating the *elastic energy of the activity pattern, a temporal coding of the natural-mode power spectrum* is obtained. This signal is applicable as a *compact representation of the network input.* The natural modes in the MCNN described in Chapter 4 are translation invariant. Hence, the elastic energy signal is a translation-invariant representation of the MCNN input.

The MCNN is a simple Membrain network. We briefly discuss here other possibilities for further research in this direction. In the experiments grey-level images were presented to the MCNN as *initial-deflection* patterns, setting the initial-deflection speed to zero. However, other methods of entering images into the system are possible. We describe two possibilities. Firstly, for signature classification, it is possible to use the spatio-temporal pattern of *writing velocity and pen pressure* as input. This is accomplished by applying a point force to the network with varying strength at a moving

position. Depending on the drawing speed and the pressure applied to the pen, a characteristic wave pattern emerges. As these properties are characteristic for a person's signature, this method will enhance the ability of the system to reject false signatures. We note that this application requires a different form of network *input*, viz., using the external force $\mathbf{f}(t)$ rather than using the initial-deflection pattern.

Secondly, input may be presented to the Membrain by *perturbing the connection matrix*. This idea is based on recent work of Pribram (1991) who describes a neural model in which neural activity is modeled as wave propagation similar to the propagation of waves in the MCNN. In Pribram's model, however, input is presented to the network as a perturbation to the system's stationary wave state. Analogously, input in the Membrain model may also be presented as a perturbation by entering $f_{ij}(t)u_{ij}(t)$ instead of $f_{ij}(t)$. We note that this actually results in a piece-wise addition of the image components of $f_{ij}(t)$ to the elements on the main diagonal of the connection matrix. Hence, a perturbation will change the eigenvalues of $\mathbf{A}$ and consequently $\hat{u}(t)$ will be a *frequency modulation* of the "fingerprint" rather than an amplitude modulation. An advantage of frequency modulation may be that the energy in the network is constant regardless of its input. Interestingly, Pribram's approach has a strong *formal* similarity with research on dilute metal alloys (see, e.g. Braspenning (1982); Stefanou *et al.* (1987)). In solid-state theory, dilute alloys are also modelled as small perturbations to an otherwise regular lattice of host atoms. However, in contrast to Pribram's zero-order perturbation approach, modern solid-state theory accounts for *any* order of perturbation by means of the multiple scattering theory.

The MCNN model is not restricted to translation-invariant pattern recognition. Other, non-homogeneous and non-cellular connection architectures may be constructed enforcing other types of invariancy corresponding to particular feature spaces. Ideally, such connection structures should be adapted automatically depending on the characteristics of the connection architecture. For instance, the learning method proposed by Zou *et al.* (1991) potentially applies to a MCNN connection architecture and the learning-associator rule (Kohonen, 1972; Anderson, 1972) potentially applies to a fully-interconnected Membrain connection architecture.

5.2 The Organization of Representation

This section evaluates the organization of representation in ANNs. In Chapter 1 we indicated that connections, neurons and activation entities are *media* in which representations can be stored. From the experiments in Chapter 2, 3 and 4 we conclude that the actual *format* of the representation in these media results from organization, i.e. the network behaviour and, if present, the learning procedure. We distinguish three types of organization: (1) supervised, (2) unsupervised and (3) cellular, corresponding to the behaviour of the MLNN, SOFM and MCNN respectively. From each one of these types of organization one or any combination of the following representation forms may emerge: distributed, pattern, modular, and temporal. Additionally we evaluate the feasibility of a massively parallel implementation of the MLNN, SOFM and MCNN.

5.2.1 Supervised Organization

The organization of the MLNN is *supervised*. This means that feedback from the environment, viz. an example, is received for creating internal representations. The feedback indicates to what degree the network output is in error. Repeated interaction with the environment allows adaptation of the MLNN using a gradient-descent method, i.e. the BP algorithm. In the MLNN, two kinds of representation can be distinguished. Firstly, the representations in the activation pattern which are organized by forward propagation of the input activities can be distinguished. In the input and output layers, this representation is localized, i.e. each activation represents a defined feature. In the hidden layers, this representation is distributed. For instance, one hidden-output activation may contribute to more than one representation which requires a combination of multiple hidden-output activations. Secondly, the representations in the connection weights are distinguished. This form of representation is organized by the BP algorithm. Similar to the hidden-activation representations, the representations in the connections weights are distributed.

The organization of the MLNN, i.e. forward propagation and back propagation, is difficult to implement on a MPC. The wiring for the fully interconnected layers (bidirectional) represents a problem since a MPC is not

likely to have such a richly-interconnected processor architecture. Hence, a considerable amount of communication overhead is introduced for routing of information in order to realize virtual interconnections.

5.2.2 Unsupervised Organization

The SOFM organizes in an *unsupervised* mode, i.e. the organization of an internal representation is not influenced by feedback from a teacher. This is also known as self organization since the construction of representations depends entirely on the system's input and dynamics. The self organization of a 2D SOFM results in a feature map, i.e. a two-dimensional (pattern) representation of the input distribution. This representation has two important characteristics. Firstly, the weight vector of a neuron is adapted to a typical input vector. Hence, in contrast to a weight vector in the MLNN, a SOFM weight vector refers to an interpretable environment state. Secondly, the arrangement of neurons reflects the structure of the input distribution, i.e. weight vectors of neighbouring neurons represent similar inputs. Moreover, the self organization in a modular SOFM will lead also to a modular representation of the input.

A SOFM can be implemented on a MPC with a sparse connection architecture, such as a matrix architecture, without communication overhead for routing of information. This is remarkable since neurons in a SOFM are fully interconnected. The importance of a local-interaction approach is underlined by results presented by Togneri and Attikouzel (1991). They implemented the Kohonen algorithm on a parallel transputer network that is configured as a master-slave system using a simple routing strategy for realizing global communication. From experimental results, they learned that the communication overhead caused the execution time to increase when too many transputers were used. In the following example we will show that self organization in the SOFM, i.e. finding the best matching neuron and adaptation of the weight vector, can be implemented by a simple *diffusion mechanism* in such a way that only neighbouring processors need to interact.

Example

The algorithm described here may be characterized as a diffusion model in

which the best-matching processor value (cf. Section 3.2.4) and its coordinates diffuse through the MPC. Let the MPC have a matrix architecture such that processors are indexed $(i,j) \in N \times N$. For instance, the ICL-DAP (see also Appendix A) may be used ($N = 8$). Let processor (i,j) simulate SOFM neuron (i,j) having output y_{ij}. The MPC operates on a single-instruction single-data stream. This means that each processor executes the same program on exactly the same input vector. The algorithm for this program is presented in Table 5.1.

```
        Processor (i,j) output y_ij.
        Max. communication distance is n.

1.      Y_ij ← y_ij
2.      p_ij ← (i,j)
3.      repeat n times
4.          for (a,b) in (i,j-1), (i-1,j), (i+1,j), (i,j+1)
5.              if Y_ij < Y_ab then
6.                  Y_ij = Y_ab
7.                  p_ij = p_ab
                endif
            endfor
        endrepeat
8.      Adapt weight vectors using distance ||p_ij - (i,j)||
```

Table 5.1: *Algorithm for SOFM implementation on a parallel computer.*

After each processor has calculated y_{ij}, the diffusion process begins. Without loss of generality we may assume that the value of y_{ij} is unique. The diffusion process operates on two local processor variables: Y_{ij} and $\mathbf{p}_{ij}$. Variable Y_{ij} represents the minimum activation value of the processor and the neighbouring processors. Variable $\mathbf{p}_{ij}$ represents the coordinates of the processor whose activation value is stored in Y_{ij}. In step one (cf. Table 5.1), Y_{ij} is initialized to y_{ij}, i.e. the local activation value. Hence, $\mathbf{p}_{ij}$

is initialized to (i, j) in step two. In step five the contents of Y_{ij} and $\mathbf{p}_{ij}$ are replaced by values Y_{ab} and $\mathbf{p}_{ab}$ of a neighbouring processor (a, b) if Y_{ab} corresponds to a smaller activation value, i.e. a better match. This comparison is repeated four times; one time for each neighbour (cf. step four). However, in order to receive data from non-neighbouring neurons the loop in step four must be executed n times allowing the smallest Y_{ij} and $\mathbf{p}_{ij}$ to disperse through the MPC (cf. step three). In this case, $n = 2N - 2$, i.e. the maximum communication distance (cf. Appendix A) in an $(N \times N)$ matrix connection architecture. Finally, in step eight, each processor can adapt its weight vector (cf. Equations 3.8 and 3.9) independently since the location of the best matching processor is represented by $\mathbf{p}_{ij}$. We emphasize the final *independent* adaptation meaning that the neighbourhood size used for adaptation does *not* influence the speed of self organization.

End of example

Using a hypercube connection architecture, the self-organization process runs even more efficiently since the maximum communication distance scales logarithmically with the number of processors. For instance, a 14-dimensional hypercube may be used to implement a 128×128 feature map ($2^{14} = 2^7 \times 2^7 = 128 \times 128$). Finding the best-matching processor and adapting the weight vectors takes 14 communication updates. We note that the diffusion process does not require routing of information. Apart from the minimum number of communication updates, there is no communication overhead.

5.2.3 Cellular Organization

CNNs organize representations by local interaction between cells. We call the behaviour that emerges from these interactions *cellular organization*. From the three types of organization, the cellular organization can be implemented with the least difficulty on a MPC since all cells are identical and there is no need for global communications. Cellular organization operates on representations formed in the activation pattern of the network and, unlike supervised and unsupervised organization, it does not affect the weight values of the network.

Representations in the MCNN are subject to a cellular organization. The

wave pattern that emerges from the interacting harmonic oscillators is a multi-mode oscillation, i.e. a superposition of natural-vibration modes. When the MCNN is in a natural-vibration mode, all cells oscillate with the same frequency as if they are operated by a central control mechanism. Each mode is characterized by a unique resonance frequency and both its amplitude and phase represent information. In case of the MCNN we may conclude that the introduction of a second-order neuron-processing model results in a *temporal* representation.

Two properties of the temporal representation in the MCNN can be distinguished. Firstly, the amplitudes of *the constituting features of the activity pattern* are represented separately in the elastic energy of the MCNN. We note that by calculating the elastic energy the phase information of the features is eliminated (cf. Equation 4.26). However, the phase also represents information which brings us to our second advantage. The wave pattern of the MCNN has a capacity for *representing* temporal patterns. We recall that a MCNN can also take into account the spatio-temporal pattern of writing velocity and applied pressure for signature classification (cf. Section 5.1).

5.3 Evaluation

In this section we evaluate the research topics and their results. Firstly, the representation in connections, neurons and activation is evaluated. Subsequently, the role of the complex-valued MLNN, the modular SOFM and the MCNN is evaluated in view of the conclusions of Section 5.1. Then we evaluate visualization techniques which have substantially contributed to all final results. Finally, we evaluate the organization of representation in ANNs.

Connections, Neurons and Activation

The basic entities of ANNs are connections, neurons and activation. Each entity represents information and can be subjected to organization. The connections entity refers to the information represented by connection weights determining the interactions between neurons. The weights in the MLNN and the SOFM participate in the representation of system inputs

and outputs. Weights in the MCNN are not adapted although they determine the natural-mode representation of the input. The neurons entity refers to the information represented by the arrangement of neurons, i.e. the network topology. In the MLNN this topology is not important provided that the input and output neurons are interpreted correctly. In the SOFM and MCNN the topology adds specific value to the information represented by the activity pattern. In the SOFM, the topology of the activity pattern represents the topology of the input distribution. In the MCNN, natural modes of the MCNN would just be eigenvectors of the connectivity matrix instead of wave patterns without the two-dimensional network topology. Finally, the activation entity refers to the information represented by the state of the network. This state is the combination of input, network dynamics and possibly also the history of the system input. The representation of a single neuron state can be localized, distributed or temporal. For instance, the state of the hidden neurons in a MLNN is a distributed representation. In contrast, the state of a SOFM is a localized representation, i.e. it represents a characteristic input state. The state of a MCNN is a temporal representation, i.e. the structure of the representation is coded in a time component.

MLNN and BP algorithm

A MLNN and a BP algorithm are a successful combination. Experiments have shown that the functional architecture of the MLNN and the equations underlying the BP algorithm may be adapted so that other processing functions and data types, e.g. complex values, can be used in the network. The choice of such functions and data types may be important for the success of the adaptation process. The research presented in Chapter 2 should be considered as a first introduction of complex values. More research is required to identify the exact potential of the complex-valued MLNN and BP algorithm.

The Modular SOFM

Feature maps can be used as a uniform representation medium for non-uniform input signals. In applications where different signals must be integrated such a uniform representation is an excellent preprocessing stage. The advantage lies in the dimension reduction of the input signals to two dimensions. A feature-map representation facilitates both the discrimination

between different inputs as well as the recognition of similar inputs.

Modularity is important. One of the many solutions that is employed by the brain to eliminate the wiring problem is the introduction of a modular architecture. ANNs also benefit from a modular approach and so will MPCs. In particular we studied a modular system of 2D SOFMs. Thanks to the modularization the input distribution was represented more accurately compared to a single-map system. This conclusion is based on a new type of visualization of the weight-point structure. The selection of modules can be aided heuristically by visualizing the correlation feature map on which correlated input sensors appear in the same cluster. In the DSM-experiment described in Chapter 3, the selection of modules required considerable heuristic involvement. Only three of the seven clusters were predicted accurately by the correlation feature map and the other four were found only after experimental verification. Hence, other heuristics should be tested to develop a more robust procedure for modular decomposition of the single SOFM.

The Membrain Model

The MCNN should be considered as a first step towards a combination of connectionist modelling and quantum-mechanical theory. Since a great deal of mathematical knowledge on such systems is available we believe that they represent powerful concepts for understanding connectionism. We expect the quantum-mechanical systems to be at least as valuable for modelling neural networks as the models from statistical physics, e.g. the Spin-Glass formalism (Hopfield, 1982), the Boltzmann machine (Aarts and Korst, 1989) and the lattice-gas model (Postma *et al.*, 1991). The MCNN is a purely linear dynamical system with important properties that can be analyzed in a straightforward manner. Either seen from an electrical, mathematical or biological perspective, the introduction of a non-linear MCNN is a desirable generalization of the linear MCNN. However, the analysis of such a non-linear dynamical system will require a considerable larger analytical effort. We can predict that some properties of the non-linear MCNN will be the same, e.g. translation invariance, but others may be quite different, e.g. stability and natural modes. Interestingly, Nikovski (1993) shows that a discrete-time version of the continuous-time MCNN possess almost identical properties without the need for numerical integra-

tion of the MCNN differential equations. For digital implementations this modification of the original MCNN should be considered.

Visualization of Organization

Visualization of connections, neurons or activation provides a useful insight in the emerging representations. For instance, the speed of gradient descent in the BP algorithm is strongly influenced by the learning-rate parameter. Visual inspection of the learning-error curve allows interactive adjustment of the learning rate. We note that adaptive BP algorithms automatically adapt the learning rate but the criteria for decrease or increase are not always optimal. Also, visualization of the Membrain temporal wave pattern and single natural-vibration modes help to understand the dynamical behaviour of the system.

Owing to the self organization in the SOFM it is difficult to comprehend the forming of representations during adaptation. Hence, interactive visualization of the feature map during self organization is of great value and we paid special attention to several visualization techniques in Chapter 3. The visualization of the weight-point structure in SOFMs provides direct visual information regarding the quality of the topology-preserving map. Using such a visualization, system parameters such as neighbourhood size and learning rate can be decreased at proper moments. However, the presented visualization method is designed for visual interpretation and is therefore not suited for objectively comparing different weight projections. Therefore we have introduced feature-tension and in-depth statistics enabling an objective comparison, e.g. by using a t-test.

The Organization of Representation

The organization of representation in ANNs emerges from the collective computation of neurons in a network. The collective computation is characterized by the action of thousands of neurons operating in parallel on distributed, pattern, modular and temporal representations. In fact, neurons can be viewed as components of one large connectionist computer, the ANN. The collective computation of neurons in a connectionist computer is quite different from the computation in a traditional computer in which processor and memory are separated. Firstly, the connectionist computer operates neither on symbolical nor on numerical representations. Secondly, and even more surprising is the absence of a stored program in the connec-

tionist computer. That is, in the connectionist computer memory and processor are merged. Consequently, representations cannot be programmed but have to grow in patterns emerging from the network dynamics. In summary, our research has found that the integration of network dynamics and representation of information is the essence of a connectionist computer. The connections, neurons and activation that were the main themes of our research are in fact neural media that enable such an integration.

Appendix A

Massively Parallel Computers

This appendix elaborates the parallel programming problem mentioned in the introduction of Chapter 1. Essentially, this problem is encountered when programming *massively parallel computers* (MPCs). MPCs are computers containing a *scalable* number of processors operating in *parallel.* Scalable means that the number of processors in the MPC is adjustable to any size without consequences for the nature of the computation. In this way the MPC size may be adapted to the required processing power. State-of-the-art supercomputers may also be parallel computers but their system architecture still relies on a restricted number of super-fast processors, i.e. an architecture that is not scalable. For instance, a Cray-1 supercomputer is not a MPC but a parallel *pipeline computer.* This means that a computation is processed in parallel by dividing it into a number of subsequent stages in such a way that output of one stage is input to a subsequent stage, so that a processing pipeline is produced. This procedure can be illustrated as follows: suppose a table of values $y = 2x + 1$ must be computed for 1,000 different values x. Normally, this would require 2,000 time units, assuming that multiplication and addition require one time unit each. If the computation is partitioned into two concurrent stages $z = 2x$ and $y = z + 1$ only 1,001 time units are required. The first time unit is introduced by the start-up time, i.e. the time between the first input in the

pipeline and the first output result of the last stage.

The amount of pipeline parallelism is restricted by the nature of the computation. For instance, there is no straightforward solution for calculating $y = 2x + 1$ in three stages. Another example of this problem is illustrated by pipeline processing in the mass production of cars through a car-assembly line. Obviously, the number of stages in the assembly process must not exceed the number of parts to be assembled. Hence, the pipeline is not scalable. If we wish to parallelize the process beyond this number of stages, it is necessary to look for alternative solutions. One radical solution may be to design the same car in a different way using more parts. The inefficiency of such a design is compensated by the larger number of possible stages in the assembly line. Analogously, parallelizing sequential computer programs may require a new functional software design.

Sequential computers are characterized by their processor architecture. Parallel computers can be characterized by their overall architecture describing the communication channels between processors. Ideally, the processors are fully interconnected resulting a maximum communication distance of 1. By communication distance we mean the number of connections a message from a processor must pass in order to arrive at its destination. However, the number of interconnections in a fully interconnected system equals the number of processors squared. For instance, realizing a fully interconnected 1,000 processor system requires 1,000,000 interconnections. Such a number of interconnections represents a practical *wiring problem.* In order to find a good compromise between the number of interconnections and the maximal communication distance in the system, many different parallel computer architectures have been designed. To give an idea of some characteristic MPCs we present some examples.

ICL-DAP: The ICL-DAP computer is a Single-Instruction stream / Multiple-Data stream (SIMD) computer consisting of a matrix of 64 processors (8×8). In a SIMD computer each processing element executes the same program on different input data. Such a connection architecture is suited for computing numerical problems on two-dimensional surfaces, e.g. finite-element analysis. The maximal communication distance in the ICL-DAP is 14, e.g. from processor $(1, 1)$ to processor $(8, 8)$.

NCube: The NCube-II system consists of processors connected in a hypercube communication architecture (Mokhoff, 1986). In an N-dimensional hypercube architecture each processor is connected to N other processors by placing them on the vertices of the N-dimensional hypercube. For instance, a three-dimensional hypercube consists of eight processors placed on the vertices of a cube. The edges of the cube denote communication channels. Due to this structure the maximal communication distance is N. The NCube may be programmed in a MIMD fashion. However, most applications consist of SIMD solutions enabling a straightforward adaptation to an NCube configuration with a lower or higher dimension (Henseler *et al.*, 1987). Currently the largest NCube configuration is limited to $2^{13} = 8,192$ 64-bit processors (13-dimensional hypercube).

Connection Machine: The Connection Machine (CM) (Hillis, 1985) has a 12 dimensional hypercube architecture with 16 processors at each vertex (arranged in a 4×4 grid). Hence, the number of processors is $16 \times 2^{12} = 65,536$ which is more than the largest NCube II configuration. However, the CM processors are simple compared to the processors in the NCube II system and are called *cells* rather than processors[1]. The CM combines its parallel architecture with an efficient routing mechanism integrated in the CM LISP programming language (CmLisp). A Programmer working in CmLisp uses the cells in the CM as if they were cells in Lisp memory. Thus, effectively the CM is a *dynamic lisp memory*. Due to CmLisp the Memory/Processor split is absent in the CM.

Transputers: The transputer concept reflects the idea of designing a building block for parallel computers similar to a *trans*istor in a silicon chip but with the functionality of an entire com*puter* (May,

[1] J.L.C. Sanz of IBM Almaden Research Center was invited to speak about advances in massively parallel computing at the 11th IAPR International Conference on Pattern Recognition, held in 1992 in The Hague (Sanz, 1992). He noted a number of interesting changes in the latest CM design, called the CM-5. Firstly, the simple processing elements have been replaced by high-performance SPARC risc (reduced instruction set computer) processors. Secondly, the hypercube architecture is abandoned and instead a tree architecture is used. Finally, the CM-5 may also be programmed in Fortran to facilitate implementation of existing numerical software.

1986; Pountain, 1986). Current transputers, i.e. T414 and T800, have four communication channels making them suitable for constructing two-dimensional array computers. Such parallel computer systems can operate in a Multiple-Instruction stream / Multiple-Data stream (MIMD) fashion because each Transputer must be programmed separately. Processing elements in a MIMD computer may execute different programs at the same time on different input data.

In a MPC the parallelization of sequential programs into sub programs, i.e. a MIMD approach, is not possible since a parallel program consisting of thousands of functionally-different interacting modules is difficult to design and maintain. Ideally, the parallel power provided by massively parallel computers is used most effectively if such machines are programmed in a SIMD fashion. In that case each processor runs the same program but processes different data. It turns out that it is difficult to program a SIMD machine in such a way that all processors are equally loaded. Not every problem is suited for a SIMD approach because the work-load balance is determined by the level of separation that can be applied to the input data. Moreover, if much information needs to be shared by processors the balance depends heavily on the maximal communication distance in the system. Ideally, all computers should have access to the same memory. Therefore, the shared-memory approach is believed to be the most promising solution to the communication problem (Pool, 1992; Bell, 1992). For instance, the KSR-I MPC uses 1,088 64-bit microprocessors and has a special operating system that realizes a "virtual-shared memory". That is, the memory is physically located in hundreds of different locations connected to individual processors but owing to the computers' wiring and operating system the memory appears to be all in one place. However, any shared-memory approach becomes increasingly difficult when the number of processors continues to grow. Moreover, when thousands of processors are reading from and writing to the same memory the readers-writers problem (Van den Herik *et al.*, 1987) may still introduce long waiting times slowing down the performance.

Neither the MIMD nor the SIMD approach is particularly well suited for programming MPCs. On the one hand, a SIMD approach is preferred because all processors operate in the same way. On the other hand, a

MIMD approach may be necessary to ensure a proper work-load balance. In MPCs this problem often leads to a combination of both SIMD and MIMD. For example, the cells in the Connection Machine operate on LISP data structures. Each cell evaluates its data structures in the same way. However, since LISP data structures also represent functions the Multiple Data becomes Multiple Instruction. Consequently, the cells can display a non-uniform behaviour far different from the uniform behaviour of processors in, e.g. an array processor.

From this example, we can learn that, in order to understand massively parallel computing, a distinction between SIMD and MIMD is no longer adequate. The reason for this inadequacy is that their meanings are based directly on the assumption that *instruction* and *data* can be separated. To understand the interaction between instruction and data it is important to realize that, normally, computer data is restricted to a *symbolic representation* of physical objects. Hence, a computer program can be perceived as a sequence of instructions related to a physical system. In contrast, MPCs may use a *distributed representation* for physical objects in order to realize a balanced work load. Consequently, local information has no symbolic interpretation and the operation of a single processor in relation to the physical system is difficult to comprehend.

This new information-processing strategy is called the *connectionism framework* (Feldman, 1981) because information is coded as a pattern of connected elements. Connectionism is inspired on neural networks that process information in the brain. Within the research field of artificial neural networks (ANNs) many models have been developed (see, e.g. Rumelhart *et al.*, 1986). These models are either simulated on general-purpose computers or implemented on dedicated neural-network hardware. Some of these dedicated neural-network hardware solutions are in fact scalable MPCs containing digital signal processors (Roska *et al.*, 1991) or general-purpose microprocessors (Heemskerk *et al.*, 1991). Apparently, the gap between MPCs and ANNs is small on the level of hardware implementation. In contrast, on the operational level some major differences exist, for instance, the specification of system behaviour. Being aware of this contrast and of the massively parallel processing capabilities of the brain we are certain that ANN research represents a new approach to the parallel programming problem.

Appendix B

Solution of second-order linear differential equations

The system of uncoupled second-order linear differential equations described in Equation 4.17 may be solved analytically using a Laplace transform. Let $x_n(t)$ and $y_n(t)$ denote the n-th component of $\mathbf{x}(t)$ and $\mathbf{y}(t)$, respectively. The typical differential equation for $x_n(t)$ looks as follows:

$$\frac{d^2x_n(t)}{dt^2} = \gamma_n x_n(t) - \alpha\frac{dx_n(t)}{dt} + y_n(t) \tag{B.1}$$

Let $X(s)$ denote the Laplace transform of $x_n(t)$ and $Y(s)$ the Laplace transform of $y_n(t)$; then

$$s^2X(s) - sx_n(0) - \dot{x}_n(0) =$$

$$\gamma_n X(s) - \alpha(sX(s) - x_n(0)) + Y(s) \tag{B.2}$$

From this it follows that

$$X(s) = \frac{1}{s^2 - \gamma_n + \alpha s}Y(s) + \frac{1}{s^2 - \gamma_n + \alpha s}\dot{x}_n(0)$$

$$+\frac{s+\alpha}{s^2-\gamma_n+\alpha s}x_n(0) \tag{B.3}$$

The solution of Equation 4.17 can be determined using an inverse Laplace transform, yielding

$$\mathcal{L}^{-1}\left\{\frac{1}{s^2-\gamma_n+\alpha s}\right\} = \mathcal{L}^{-1}\left\{\frac{1}{(s+\frac{\alpha}{2})^2-\gamma_n-\frac{\alpha^2}{4}}\right\} =$$

$$\begin{cases} \frac{1}{\omega_n}e^{-\frac{\alpha}{2}t}\sin(\omega_n t), & \omega_n^2 = -\gamma_n - \frac{\alpha^2}{4} \quad if\ \gamma_n < -\frac{\alpha^2}{4} \\ te^{-\frac{\alpha}{2}t} & if\ \gamma_n = -\frac{\alpha^2}{4} \\ \frac{1}{\omega_n}e^{-\frac{\alpha}{2}t}\sinh(\omega_n t), & \omega_n^2 = \gamma_n + \frac{\alpha^2}{4} \quad if\ \gamma_n > -\frac{\alpha^2}{4} \end{cases} \tag{B.4}$$

and

$$\mathcal{L}^{-1}\left\{\frac{s+\alpha}{s^2-\gamma_n+\alpha s}\right\} =$$

$$\mathcal{L}^{-1}\left\{\frac{s+\frac{\alpha}{2}}{(s+\frac{\alpha}{2})^2-\gamma_n-\frac{\alpha^2}{4}}\right\} + \mathcal{L}^{-1}\left\{\frac{\frac{\alpha}{2}}{(s+\frac{\alpha}{2})^2-\gamma_n-\frac{s\alpha^2}{4}}\right\} =$$

$$\begin{cases} e^{-\frac{\alpha}{2}t}\cos(\omega_n t) + \frac{\alpha}{2\omega_n}e^{-\frac{\alpha}{2}t}\sin(\omega_n t), & \omega_n^2 = -\gamma_n - \frac{\alpha^2}{4} \\ & if\ \gamma_n < -\frac{\alpha^2}{4} \\ e^{-\frac{\alpha}{2}t} + \frac{\alpha}{2}te^{-\frac{\alpha}{2}t} & if\ \gamma_n = -\frac{\alpha^2}{4} \\ e^{-\frac{\alpha}{2}t}\cosh(\omega_n t) + \frac{\alpha}{2\omega_n}e^{-\frac{\alpha}{2}t}\sinh(\omega_n t), & \omega_n^2 = \gamma_n + \frac{\alpha^2}{4} \\ & if\ \gamma_n > -\frac{\alpha^2}{4} \end{cases} \tag{B.5}$$

Given these Laplace transforms it is possible to determine the solutions for $x(t)$.

I. $\gamma_n < -\frac{\alpha^2}{4}$ $(\omega_n^2 = -\gamma_n - \frac{\alpha^2}{4})$:

$$x_n(t) = \frac{1}{\omega_n}\int_0^t y(\tau)e^{-\frac{\alpha}{2}(t-\tau)}\sin((t-\tau)\omega_n)d\tau$$

$$+\frac{e^{-\frac{\alpha}{2}t}\sin(\omega_n t)}{\omega_n}\dot{x}_n(0) \qquad (B.6)$$
$$+\left(e^{-\frac{\alpha}{2}t}\cos(\omega_n t)+\frac{\alpha}{2\omega_n}e^{-\frac{\alpha}{2}t}\sin(\omega_n t)\right)x_n(0)$$

II. $\gamma_n = -\frac{\alpha^2}{4}$:

$$x_n(t) = \int_0^t y(\tau)(t-\tau)e^{-\frac{\alpha}{2}(t-\tau)}d\tau + te^{-\frac{\alpha}{2}t}\dot{x}_n(0)$$
$$+\left(e^{-\frac{\alpha}{2}t}+\frac{\alpha}{2}te^{-\frac{\alpha}{2}t}\right)x_n(0) \qquad (B.7)$$

III. $\gamma_n > -\frac{\alpha^2}{4}$ $(\omega_n^2 = \gamma_n + \frac{\alpha^2}{4})$:

$$x_n(t) = \frac{1}{\omega_n}\int_0^t y(\tau)e^{-\frac{\alpha}{2}(t-\tau)}\sinh((t-\tau)\omega_n)d\tau$$
$$+\frac{e^{-\frac{\alpha}{2}t}\sinh(\omega_n t)}{\omega_n}\dot{x}_n(0) \qquad (B.8)$$
$$+\left(e^{-\frac{\alpha}{2}t}\cosh(\omega_n t)+\frac{\alpha}{2\omega_n}e^{-\frac{\alpha}{2}t}\sinh(\omega_n t)\right)x_n(0)$$

Bibliography

Aarts E. and Korst J. (1989). *Simulated Annealing and Boltzmann Machines*. J. Wiley, New York, NY.

Adrian E.D. (1946). *The Physical Background of Perception*. Clarendon Press, Oxford, England.

Amit D.J. (1989). *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, Cambridge, England.

Anderson J.A. (1972). A simple neural network generating an interactive memory. *Mathematical Biosciences*, Vol. 14, pp. 197–220.

Anderson J.A. and Mozer A.C. (1981). Categorization and Selective Neurons. *Parallel Models of Associative Memory* (eds. G.E. Hinton and J.A. Anderson), pp. 213–236. Lawrence-Erlbaum Associates, Inc., Hillsdale, NJ.

Anderson J.A. and Rosenfeld E. (eds.) (1988). *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA.

Bak J. and Newman D.J. (1982). *Complex Analysis*. Springer-Verlag, Berlin, Germany.

Barlow H.B. (1972). Single units and sensations: A neuron doctrine for perceptual physiology? *Perception*, Vol. 1, pp. 371–394.

Bell G. (1992). Ultracomputers: A Teraflop Before its Time. *Science*, Vol. 256, No. April, pp. 64.

Benvenuto N., Marchesi M., Piazza F., and Uncini A. (1991). A Comparison between Real and Complex-Valued Neural Networks in Communication

Applications. *Artificial Neural Networks. Proceedings of the ICANN-91, Helsinki, Finland* (eds. T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas), pp. 1177–1180. Elsevier Science Publishers.

Block H.D. (1962). The Perceptron: a model for brain functioning I. *Reviews of Modern Physics*, Vol. 34, pp. 123–135. Reprinted in Anderson and Rosenfeld (1988), pp. 138-150.

Boyce W.E. and DiPrima R.C. (1965). *Elementary Differential Equations and Boundary Value Problems.* J. Wiley. 3rd edition (1977).

Braspenning P.J. (1982). *A Multiple Scattering Treatment of Dilute Metal Alloys.* Ph.D. thesis, Vrije Universiteit Amsterdam, The Netherlands.

Braspenning P.J. (1986). Reasoning for Interpreting Sensor Data. *Proceedings of the International Conference on Intelligent Autonomous Systems*, pp. 3–20. Amsterdam, The Netherlands.

Brodmann K. (1909). *Vergleichende Localisationslehre der Grosshirnrinde in Ihren Prinzipien Dargestellt auf Grund des Zellenbaues.* Barth Publishing, Leipzig, Germany.

Chua L.O. and Yang L. (1988a). Cellular Neural Networks: Theory. *IEEE Transactions on Circuits and Systems*, Vol. 35, pp. 1257–1272.

Chua L.O. and Yang L. (1988b). Cellular Neural Networks: Applications. *IEEE Transactions on Circuits and Systems*, Vol. 35, pp. 1273–1290.

Churchland P.S. (1986). *Neurophilosophy: Toward a Unified Science of the Mind-Brain.* MIT Press, Cambridge, MA.

Clarke T.L. (1990). Generalization of Neural Networks to the Complex Plane. *Proceedings of the International Joint Conference on Neural Networks*, pp. 435–440. San Diego, CA.

Courant R. and Hilbert D. (1953). *Methods of Mathematical Physics.* Interscience Publishers, Inc., New York, NY. English translation from German original (1937).

DARPA (1988). *Neural Network Study, October 1987 - February 1988.* Technical Report, Lincoln Laboratory, MIT, Cambridge, MA.

Daugman J.G. (1989). Entropy Reduction and Decorrelation in Visual Coding by Oriented Receptive Fields. *IEEE Transactions on Biomedical Engineering*, Vol. 36, No. 1, pp. 107–114.

Daugman J.G. (1990). *An Information-Theoretic View of Analog Representations in Striate-Cortex*. MIT Press, Cambridge.

Durbin R. and Mitchison G. (1990). A dimension reduction framework for understanding cortical maps. *Nature*, Vol. 343, pp. 644–647.

Feldman J.A. (1981). A Connectionist Model of Visual Memory. *Parallel Models of Associative Memory* (eds. G.E. Hinton and J.A. Anderson), pp. 49–212. Lawrence-Erlbaum Associates, Inc., Hillsdale, NJ.

Feldman J.A. and Ballard D.H. (1982). Connectionist models and their properties. *Cognitive Science*, Vol. 6, pp. 205–254. Reprinted in Anderson and Rosenfeld (1988), pp. 484-507.

Fodor J.A. (1983). *The Modularity of Mind*. MIT/Bradford Press, Cambridge, MA.

Gazzaniga M.S. (1985). *The Working Brain*. Basic Books, New York, NY.

Goldman-Rakic P.S. (1988). Topography of Cognition: Parallel Distributed Networks in Primate Association Cortex. *Ann.Rev. Neurosci.*, Vol. 11, pp. 137–156.

Goodale M.A. and Milner A.D. (1992). Separate visual pathways for perception and action. *Trends in Neurosciences*, Vol. 15, pp. 20–25.

Gross C.G., Rocha-Miranda C.E., and Bender D.B. (1972). Visual properties of neurons in inferotemporal cortex of the macaque. *Journal of Neurophysiology*, Vol. 35, pp. 96–111.

Grossberg S. (1973). Contour Enhancement, Short Term Memory, and Constancies in Reverberating Neural Networks. *Studies in Applied Mathematics*, Vol. LII, pp. 213–257. Reprinted in Grossberg (1982), pp. 334-378.

Grossberg S. (ed.) (1982). *Studies of Mind and Brain*. Reidel-Publishing Company, Dordrecht, The Netherlands.

Halonen K., Porra V., Roska T., and Chua L. (1991). VLSI Implementation of a Reconficurable Cellular Neural Network Containing Local Logic (CNNL). *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, CNNA-90* (ed. T. Roska). IEEE Catalog No. 990TH0312-9, Budapest, Hungary.

Hebb D.O. (1949). *The Organization of Behavior, pp. xi-xix and 60-78.* J. Wiley, New York, NY. Reprinted in Anderson and Rosenfeld (1988), pp. 45-56.

Heemskerk J.N.H., Murre J.M.J., Hoekstra J., Kemma L.H.J.G., and Hudson P.T.W. (1991). BSP 400: A modular neurocomputer assembled from 400 low-cost microprocessors. *Artificial Neural Networks. Proceedings of the ICANN-91, Helsinki, Finland* (eds. T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas), pp. 709–714. Elsevier Science Publishers.

Henseler J. and Braspenning P.J. (1990a). Training Complex Multi-Layer Neural Networks. *Proceedings of the Latvian Signal Processing International Conference, LSPIC'90*, pp. 301–305. Riga, Latvia. Also as Technical Report CS 90-02, University of Limburg, The Netherlands.

Henseler J. and Braspenning P.J. (1990b). Back Propgation with Complex-Valued Weights. *Proceedings of the IEEE Symposium on Neural Networks* (eds. H. Koppelaar, M.G. Voorzanger, G.R. den Heijer, F.J. Bernard, and P. ter Wee), pp. 41–52. IEEE Student Branch, Delft, The Netherlands.

Henseler J. and Braspenning P.J. (1991). A Cortex-like Architecture of a Cellular Neural Network. *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, CNNA-90* (ed. T. Roska), pp. 244–253. IEEE Catalog No. 990TH0312-9, Budapest, Hungary.

Henseler J. and Braspenning P.J. (1992). Membrain: A Cellular Neural Network based on a Vibrating Membrane. *Int. Journ. of Circuit Theory and Applications*, Vol. 20, No. 5, pp. 483–496. Special Issue on Cellular Neural Networks. Reprinted in Roska and Vandewalle (1993).

Henseler J., Scholtes J.C., and Verhoest C.R.J. (1987). *The Design of a Parallel Knowledge-based Optical Character-Recognition System.* M.Sc. thesis, Dept. of Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands.

Hillis D. (1985). *The Connection Machine.* MIT Press, Cambridge, MA.

Hirsch M. (1989). Convergent Activation Dynamics in Continuous Time Networks. *Neural Networks*, Vol. 2, pp. 331–349.

Holdaway R.M. (1989). Enhancing Supervised Learning Algorithms Via Self-Organization. *Proceedings of the IJCNN, Washington 1989*, pp. 523–530. IEEE Service Center, Piscataway, NJ.

Hopfield J.J. (1982). Neural networks and physical systems with emergent collective behavior. *Proc. Natl. Acad. sci. USA*, Vol. 79, pp. 2554–2558. Reprinted in Anderson and Rosenfeld (1988), pp. 460-464.

Hopfield J.J. and Tank D.W. (1985). "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, Vol. 52, pp. 141–152.

Hubel D.H. and Wiesel T.N. (1959). Receptive fields of single neurons in the cat's striate cortex. *J. Physiol.*, Vol. 148, pp. 574–591.

Hubel D.H. and Wiesel T.N. (1963). Shape and arrangements of columns in cat's striate cortex. *J. Physiol.*, Vol. 165, pp. 559–568.

Ingber L. and Nunez P.L. (1990). Multiple Scales of Statistical Physics of the Neocortex: Application to Electronencephalography. *Mathl. Comput. Modelling*, Vol. 13, No. 7, pp. 83–95.

Kaas J.H. (1987). The Organization of Neocortex in Mammals: Implications for Theories of Brain Function. *Ann. Rev. Psychol.*, Vol. 38, pp. 129–151.

Kaas J.H., Nelson R.J., Sur M., Lin C.-L., and Merzenich M.M. (1979). Multiplle Representations of the body within the primary somatosensory cortex of primates. *Science*, Vol. 204, pp. 521–523.

Kandel E.R. and Schwartz J.H. (eds.) (1985). *Principles of Neural Science.* Elsevier, New York, NY. 2nd edition (1985).

Kim M.S. and Guest C.C. (1990). Modification of Back Propagation Networks for Complex-Valued Signal Processing in Frequency Domain. *Proc. Int. Joint Conf. on Neural Networks*, pp. 27–31. San Diego, CA.

Klaassen A.J. (1992). *Computing with cables: towards massively parallel neuro computers.* Ph.D. thesis, Dept. Electrical Engineering, Delft University Press, Delft, The Netherlands.

Knapp A.G. and Anderson J.A. (1984). Theory of Categorization Based on Distributed Memory Storage. *Journ. of Exp. Psych.: Learning, Memory and Cognition*, Vol. 10, pp. 616–637. Reprinted in Anderson and Rosenfeld (1988), pp. 588-609.

Knudsen E.I., Du Lac S., and Esterly S.D. (1987). Computational Maps in the Brain. *Ann.Rev. Neurosci.*, Vol. 10, pp. 41–65.

Kohonen T. (1972). Correlation Matrix Memories. *IEEE Transactions on Computers*, Vol. C-21, pp. 353–359.

Kohonen T. (1977). *Associative Memory: A system-Theoretical Approach.* Springer-Verlag, Berlin, Germany.

Kohonen T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, Vol. 43, pp. 59–69.

Kohonen T. (1984). *Self-Organisation and Associative Memory.* Springer-Verlag, Berlin, Germany. 2nd revised edition (1988).

Koikkalainen P. and Oja E. (1990). Self-Organizing Hierarchical Feature Maps. *Proc. Int. Joint Conf. on Neural Networks*, pp. 279–284. San Diego, CA.

Kolb B. and Whishaw I.Q. (1990). *Fundamentals of Human Neuropsychology.* W.H. Freeman and Company, New York. 3rd edition (1990).

Kraaijveld M.A., Mao J., and Jain A.K. (1992). A non-linear projection method based on Kohonen's Topology Preserving Maps. *Proceedings of the 11th IAPR International Conference on Pattern Recognition. Conference B: Pattern Recognition Methodology and Systems. The Hague, The Netherlands.*, pp. 41–45. IEEE Computer Society Press, Los Alamitos, CA.

Lashley K.S. (1950). In search of the engram. *Proceedings of the Society of Experimental Biology Symposium, No. 4: Psychological Mechanisms in Animal Behavior*, pp. 454–455, 468–473, 477–480. Cambridge: Cambridge University Press. Reprinted in Anderson and Rosenfeld (1988), pp. 59-63.

Le Cun Y. (1985). A Learning Procedure for Assymetric Threshold Network. *Proceedings of Cognitiva '85*, pp. 599–604. Paris, France.

Leung H. and Haykin S. (1991). The Complex Backpropagation Algorithm. *IEEE Transactions on Signal Processing*, Vol. 39, No. 9, pp. 2101–2104.

Lipschutz S. (1991). *Theory and Problems of Linear Algebra*. McGraw-Hill, London, England.

Luria A.R. (1973). *The Working Brain*. Penguin, New York, NY.

Martin J.H. (1985). Anatomical Substrates of Somatic Sensation. *Principles of Neural Science* (eds. E.R. Kandel and J.H. Schwartz). Elsevier, New York, NY.

Matsumoto T., Yokohama T., Suzuki H., Furukawa R., Oshimoto A., Shimmi T., Matsushita Y., Seo T., and Chua L.O. (1990). Several Image Processing Examples by CNN. *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, CNNA-90* (ed. T. Roska). IEEE Catalog No. 990TH0312-9, Budapest, Hungary.

May D. (1986). *Occam Product Definition (Preliminary)*. Technical Report, Inmos Inc., England.

McCulloch W.S. and Pitts W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133.

Minsky M. and Papert S. (1969). *Perceptrons: An introduction to Computational Geometry, expanded edition*. MIT Press, Cambridge, MA. 2nd revised edition (1988).

Mishkin M., Ungerleider L.G., and Macko K.A. (1983). Object vision and spatial vision: two cortical pathways. *Trends in Neurosciences*, Vol. 6, pp. 414–417.

Mokhoff N. (1986). Hypercube Architecture leads the Way for Commercial Supercomputers in Scientific Applications. *Computer Design*, Vol. May, pp. 28–30.

Mood A.M., Graybilll F.A., and Boes D.C. (1950). *Introduction to the Theory of Statistics.* McGraw-Hill, Tokyo, Japan. 3rd edition (1973).

Mountcastle V.B. (1978). An Organizing Principle for Cerebral Function: The Unit Module and the Distributed System. *The Mindful Brain* (ed. V.B. Mountcastle). MIT Press, Cambridge, MA.

Murre J.M.J., Phaf R.H., and Wolters G. (1989). CALM Networks: a modular approach to supervised and unsupervised learning. *Proceedings of the IJCNN, Washington 1989*, pp. 649–656. IEEE Service Center, Piscataway, NJ.

Nauta W.J.H. and Feirtag M. (1979). The Organization of the Brain. *Scientific American*, Vol. 241, pp. 102.

Nikovski D. (1993). *Notes on Transforming the Membrain Model into Discrete Time.* Technical Report unpublished, Department of Computerscience, University of Limburg, Maastricht, The Netherlands.

Noest A.J. (1988). *Informatie Verwerking In Neurale Netwerken.* Technical Report, Dutch Institute for Brain Research, Amsterdam, The Netherlands.

Nunez P.L. (1988). Global Contributions to Cortical Dynamics: Theoretical and Experimental Evidence for Standing Wave Phenomena. *Dynamics of Sensory and Cognitive Processing by the Brain* (ed. E. Basar), pp. 173–182. Springer, New York, NY.

Oppenheim A.V., Willsky A.S., and Young I.T. (1983). *Signals and Systems.* Prentice-Hall, Inc., Englewood Cliffs, NJ.

Palm G. (1982). *Studies of Brain Function: Neural Assemblies, an Alternative Approach to Artificial Intelligence.* Springer-Verlag, Berlin, Germany.

Parker D.B. (1985). *Learning-Logic.* Technical Report TR-47, MIT, Cambridge, MA.

Penfield W. and Jasper H.H. (1954). *Epilepsy and the Functional Anatomy of the Human Brain.* Little and Brown, Boston, NY.

Penfield W. and Rasmussen T. (1950). *The cerebral cortex of man.* Macmillan, New York, NY.

Pool R. (1992). Massively Parallel Machines Usher In Next Level of Computing Power. *Science*, Vol. 256, No. April 3, pp. 50–52.

Posner M.I. (1978). *Chronometric Explorations of Mind*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Postma E.O., Van den Herik H.J., and Hudson P.T.W. (1991). Autonomous Attentional Selection in SCAN. *Proceedings of the IJCNN*, pp. 448–453. Singapore.

Pountain D. (1986). Personal Supercomputers (Transputers). *Byte*. No. July, pp. 363-368

Pribram K.H. (1991). *Brain and Perception: Holonomy and Structure in Figural Processing. The 1986 MacEachran Lectures*. Lawrence-Erlbaum Associates, Inc., Hillsdale, NJ.

Ritter H. (1989). Combining Self-Organizing Maps. *Proceedings of the IJCNN, Washington 1989*, pp. 499–502. IEEE Service Center, Piscataway, NJ.

Rojer A. and Schwartz E. (1989). A Multiple-Map model for Pattern Classification. *Neural Computation*, Vol. 1, No. 1, pp. 104–115.

Rosenberg C.R. (1987). Analysis of NETtalk's Internal Structure. *Proceedings of the 9th Annual Cognitive Science Conference held in Seattle WA, August 1987.*

Rosenblatt F. (1959). The perceptron: a probablistic model for information storage and organisation in the brain. *Psychological Review*, Vol. 65, pp. 386–408. Reprinted in Anderson and Rosenfeld (1988), pp. 92-114.

Rosenblatt F. (1962). *Principles of Neurodynamics*. Spartan, New York, NY.

Roska T. and Chua L.O. (1991). Cellular Neural Networks with Nonlinear and Delay-Type Template Elements. *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, CNNA-90* (ed. T. Roska). IEEE Catalog No. 990TH0312-9, Budapest, Hungary.

Roska T. and Vandewalle J. (eds.) (1993). *Cellular Neural Networks.* John Wiley & Sons, Ltd., England.

Roska T., Bártfai G., Sziráni T., Radványi A., Kozek T., and Ugray Zs. (1991). A Hardware Accelerator Board for Cellular Neural Networks: CNN-HAC. *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, CNNA-90* (ed. T. Roska). IEEE Catalog No. 990TH0312-9, Budapest, Hungary.

Rumelhart D.E. (1988). Learning and Generalization. *Proceedings of the ICNN'88, San Diego.* Plenary Address.

Rumelhart D.E., Hinton G.E., and Williams R.J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing: Foundations, Volume 1 (Rumelhart et al., 1986)* (eds. D.E. Rumelhart, J.L. McClelland, and the PDP Research Group), pp. 318–362. MIT Press, Cambridge, MA.

Rumelhart D.E., McClelland J.L., and the PDP Research Group (eds.) (1986). *Parallel Distributed Processing: Exaplorations in the Microstructure of Cognition. Volume 1: Foundations.* MIT Press, Cambridge, MA.

Sanz J.L.C. (1992). Advances in Massively Parallel Computing. *Proceedings of the 11th IAPR International Conference on Pattern Recognition. Conference D: Architectures for Vision and Pattern Recognition. The Hague, The Netherlands.*, pp. 95–106. IEEE Computer Society Press, Los Alamitos, CA.

Schadé J.P. (1983). *De Functie van het Zenuwstelsel.* Het Spectrum, Utrecht, Antwerpen.

Schadé J.P. (1984). *Onze Hersenen.* Het Spectrum, Utrecht, Antwerpen.

Scheibel M.E. and Scheibel A.B. (1970). Elementary processes in selected thalamic and cortical subsystems - the structural substrates. *The Neurosciences second study program* (ed. F.O. Schmitt), pp. 443–457. Rockefeller University Press, New York, NY.

Scholtes J.C. (1993). *Neural Networks in Natural Language Processing and Information Retrieval.* Ph.D. thesis, Universiteit van Amsterdam, The Netherlands.

Sejnowski T.J. (1981). Skeleton Filters in the Brain. *Parallel Models of Associative Memory* (eds. G.E. Hinton and J.A. Anderson), pp. 189–212. Lawrence-Erlbaum Associates, Inc., Hillsdale, NJ.

Stark L.W. (1993). Neural Nets, Random Design and Reverse Engineering. *Proceedings of the IJCNN, San Fransisco 1993*, pp. 1313–1320. IEEE Service Center, Piscataway, NJ.

Stefanou N., Braspenning P.J., Zeller R., and Dederichs P.H. (1987). Treatment of lattice relaxations in dilute alloys within the Korringa-Kohn-Rostoker Green's-function method. *Physical Review B*, Vol. 36, No. 12, pp. 6372–6382.

Suga N. (1990). Cortical Computational Maps for Auditory Imaging. *Neural Networks*, Vol. 3.

Togneri R. and Attikouzel Y. (1991). Parallell implementation of the Kohonen algorithm on Transputer. *Proceedings of the IJCNN '91, Singapore*, pp. 1717–1722.

Ullman S. (1986). Artificial Intelligence and the Brain: Computational Studies of the Visual System. *Ann. Rev. Neurosci.*, Vol. 9, pp. 1–26.

Van den Herik H.J., Stoop J.C., and Varkevisser P.R. (1987). Heuristics in the Abbot-Monk Problem. *Expert Systems and Artificial Intelligence in Decision Support Systems* (eds. H.G. Sol, C.A.Th. Takkenberg, and P.F. de Vries-Robbé), pp. 175–195. Reidel Publishing Company, Dordrecht, The Netherlands.

Van der Smagt P.P. and Kröse B.J.A (1991). A real-time learning neural robot controller. *Artificial Neural Networks. Proceedings of the ICANN-91, Helsinki, Finland* (eds. T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas), pp. 351–356. Elsevier Science Publishers.

Von der Malsburg Ch. and Willshaw D.J. (1977). How to label nerve cells so that they can interconnect in an ordered fashion. *Proc. Natl. Acad. Sci. USA*, Vol. 74, No. 11, pp. 5176–5178.

Von Neumann J. (1966). Theory and Organization of Complex Automata. *Theory of Self-Reproducing Automata*. University of Illinois Press. lecture delivered at the University of Illinois, December, 1949.

Weigend A.S., Rumelhart D.E., and Huberman B.A. (1990). Back-Propagation, Weight-Elimination and Time Series Prediction. *Connectionist Models. Proceedings of the 1990 Summer School.* Morgan Kaufmann Publishers, Inc., San Mateo, Ca.

Weijters A.J.M.M. (1991). Self-Organizing Feature Maps. *Applications of Neural Networks: Day 1: Theory*, pp. 45–55. Centrum voor Kennistechnologie, Utrecht, The Netherlands.

Werbos P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* M.Sc. thesis, Applied Mathematics, Harvard University, Boston, MA.

Young M. (1977). *Optics and Lasers.* Springer-Verlag, Berlin, Heidelberg, New York.

Zou F., Schwarz S., and Nossek J.A. (1991). Cellular Neural Network Design Using a Learning Algorithm. *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, CNNA-90* (ed. T. Roska), pp. 73–81. IEEE Catalog No. 990TH0312-9, Budapest, Hungary.

Glossary

ACMM: Auto-Correlation Matrix Memory (Kohonen, 1977). Linear model for associative memory. Described in subsection 1.2.2.

ANN: Artificial Neural Network. Structure modelled on the biological neural networks in the brain. Introduced in section 1.2.

BP: Back Propagation (Rumelhart *et al.*, 1986). Supervised learning algorithm for MLNNs. Described in Chapter 2.

BSB: Brain-State-in-a-Box model (Anderson and Mozer, 1981). ANN used for classification. Described in subsection 1.2.2.

CM: Connection Machine (Hillis, 1985). A massively parallel computer containing 65,536 processing elements. Described in Appendix A.

CNN: Cellular Neural Network (Chua and Yang, 1988a). ANN with a two-dimensional network topology containing identical neurons that have local interaction. Described in Chapter 4.

CPU: Central Processing Unit. Central processing unit in a digital computer. Described in Chapter 1.

EPSP: Exciting Post-Synaptic Potential (Kandel and Schwartz, 1985). Electric potential generated by a synapse increasing the cell-membrane potential thereby facilitating the depolarization of a neuron. Described in subsection 1.1.3.

FINN: Fully-Interconnected Neural Network. ANN connection architecture in which every pair of neurons has a bidirectional connection. Described in subsection 1.2.2.

HTNN: Hopfield-Tank Neural Network (Hopfield and Tank, 1985). ANN-model in which neurons are arranged in rows and columns so that one winner exists in every row and column. Described in subsection 1.2.2.

IPSP: Inhibiting Post-Synaptic Potential (Kandel and Schwartz, 1985). Electric potential generated by a synapse decreasing the cell-membrane potential thereby inhibiting the depolarization of a neuron. Described in subsection 1.1.3.

MCNN: Membrain CNN (Henseler and Braspenning, 1992). CNN with an oscillating activation pattern resembling the wave pattern on a vibrating membrane. Described in Chapter 4.

MIMD: Multiple-Instruction stream / Multiple-Data stream. Mode of operation in a parallel computer where processing elements each execute a different program on different input data. Described in Appendix A.

MLNN: Multi-Layer Neural Network (Rumelhart *et al.*, 1986). Neurons in a MLNN are placed in a number of consecutive layers. Described in subsection 2.1.1.

MPC: Massively Parallel Computer. A parallel computer containing a scalable number of processing elements. Described in Appendix A.

SIMD: Single-Instruction stream / Multiple-Data stream. Mode of operation in a parallel computer where processing elements each execute the same program on different input data. Described in Appendix A.

SOFM: Self-Organizing Feature Map (Kohonen, 1982; Kohonen, 1984). ANN that forms a topological feature map of an input distribution by self organization. Described in Chapter 3.

WTANN: Winner-Take-All Neural Network (Grossberg, 1973). A FINN that activates a single neuron in favour of all other neurons in the network if it has the largest response to the network input. Described in subsection 1.2.2.

Summary

This dissertation describes research on Artificial Neural Networks (ANNs), structures that are modelled on the biological neural networks in the brain. Chapter 1 starts with a description of neural networks identifying them as networks of nerve cells, called neurons. ANNs are computing models which simplify the neuron-processing model and neural-network structure in order to understand the basic principles of neural processing. The main theme of this dissertation is the organization of representation in ANNs. The research is divided into three parts addressing three different ANN models, i.e. Multi-Layer Neural Networks, Self-Organizing Feature Maps and oscillating Cellular Neural Networks.

The Multi-Layer Neural Network (MLNN) is studied in Chapter 2. The neurons in a MLNN are organized in one or more layers, starting at an input layer and finishing with an output layer. The network can be adapted to establish a desired input-output behaviour by using a supervised-learning algorithm called Back Propagation. The input-output behaviour is determined by the weights assigned to the neurons' inputs. Normally, these weights are real-valued. However, we have studied the impact of complex-valued weights. Hence, we have developed a complex-valued back-propagation algorithm as well as a complex-valued neuron-processing model. A robot-arm experiment comparing the complex-valued MLNN with the real-valued MLNN demonstrates that in this case the complex-valued MLNN outperforms the real-valued MLNN. Hence, a complex-valued representation should be preferred to a real-valued representation.

In Chapter 3 the representation of information in Kohonen's Self-Organizing Feature Maps (SOFMs) is studied. In many ways the SOFM resembles the primary-cortical areas in the brain representing input to and output from the cortex, e.g. vision, feeling, hearing, smelling, movement and speech.

A two-dimensional SOFM can be interpreted as a map of features. The representation of features by neurons in the SOFM is created by self organization. In an organized SOFM similar features are located in the same neighbourhood. Moreover, in the primary cortex, uncorrelated signals are represented in separate areas. The research shows that likewise a system of multiple two-dimensional SOFMs can be built. The organization of the representation can be visualized and its accuracy can be tested statistically. The performance of a modular SOFM is demonstrated by 3540 measurements of 28 sensors of an oil refinery of Dutch State Mines (DSM).

The final part of the research is described in Chapter 4 and addresses the organization of representation in a new ANN model in which neurons have been modelled as harmonic oscillators. The model is called the Membrain model since waves in the activation pattern strongly resemble waves on a vibrating membrane. It is shown analytically that the Membrain modulates every input as a pattern of interfering activation waves. The input is projected on the linear space of natural vibration patterns of the network, each resonating with a unique frequency. These patterns are coded in the weight matrix and can be adjusted to features for identification. The projection of the input is a feature vector that is represented in the frequency spectrum of the elastic-energy signal. This property has been tested in a Membrain Cellular Neural Network (MCNN). In this network neurons are organized in a two-dimensional grid and only neighbouring neurons are connected. Moreover, neurons on an edge are connected to neurons at the opposite edge so that the Membrain surface is virtually unlimited. Thus, waves are not reflected by boundaries and, since neurons in a CNN are identical cells, the natural vibration patterns are translation invariant. A simulation is performed to demonstrate translation-invariant signature-image recognition.

Finally, in Chapter 5 the research results are evaluated. In this evaluation we focus on the organization of representation in ANNs. Three different types of organization have been studied leading to four different representation forms, i.e. a distributed, a pattern-based, a modular and a temporal representation or to a combination of any of these forms. We conclude that the strength of ANNs lies in their ability to integrate efficiently the representation forms through the connectionist ensemble of connections, neurons and activation.

Samenvatting

Dit proefschrift beschrijft onderzoek naar Artificiële Neurale Netwerken (ANNen). Het onderzoek is geïnspireerd door de biologische neurale netwerken in de hersenen. Hoofdstuk 1 begint met een beschrijving van neurale netwerken bestaande uit neuronen die in een netwerk van zenuwen met elkaar verbonden zijn. In de ANN-modellen zijn het verwerkingsmodel van de neuronen en de netwerkstructuur vereenvoudigd om hun mogelijkheden te doorgronden. In dit proefschrift staat de organisatie van representatie in ANNen centraal. Het onderzoek bestaat uit drie delen waarin drie verschillende ANN modellen gebruikt worden, te weten Multi-Layer Neurale Netwerken, Self-Organizing Feature Maps en oscillerende Cellulaire Neurale Netwerken.

In hoofdstuk 2 wordt het Multi-Layer Neurale Netwerk (MLNN) onderzocht. De neuronen in een MLNN zijn georganiseerd in één of meerdere lagen beginnend met een invoerlaag en eindigend met een uitvoerlaag. Met behulp van het zogenaamde Backpropagation-algoritme kan een MLNN aan de hand van voorbeelden een gewenst invoer-uitvoer gedrag aangeleerd worden. Dit gedrag wordt mede bepaald door de gewichten die de neuronen aan de invoer toekennen. Normaal gesproken zijn deze gewichten reëel; in het onderzoek is de invloed van complexe gewichten onderzocht. Hiertoe is een complexe versie van zowel het Backpropagation-algoritme als het verwerkingsmodel ontwikkeld. De prestaties van een complex MLNN zijn vergeleken met die van een reëel MLNN in een robotarmbesturing. Hieruit blijkt dat het complexe MLNN, althans in dit geval, betere resultaten oplevert. Dit betekent dat een complexe representatie de voorkeur verdient boven een reële representatie.

In hoofdstuk 3 wordt de representatie van informatie in Kohonen Self-Organizing Feature Maps (SOFMs) onderzocht. De SOFM lijkt in veel opzichten op de gebieden in de primaire hersenschors waarbinnen belangrijke in- en uitvoerfuncties voor de hersenen gerepresenteerd worden, zoals zien, voelen, horen, ruiken, bewegen en praten. Een twee-dimensionale SOFM kan beschouwd worden als een kenmerkkaart. Een zelforganiserend proces bepaalt welke kenmerken van de invoer door welke neuronen gerepresenteerd moeten worden. In een georganiseerde SOFM liggen invoerkenmerken die op elkaar lijken dicht bij elkaar. In de primaire hersenschors is dit ook het geval en worden bovendien invoersignalen, die niet samenhangen, in verschillende gebieden gerepresenteerd. Het onderzoek laat zien dat op die manier ook met meerdere twee-dimensionale SOFMs een modulaire representatie opgebouwd kan worden. Om een indruk te krijgen van de kwaliteit van de representatie is een maat ontworpen die zowel visueel als statistisch geïnterpreteerd kan worden. De modulaire SOFM en de visualisatie worden gedemonstreerd aan de hand van 3540 metingen afkomstig van 28 sensoren in een raffinaderij van DSM.

Het laatste deel van het onderzoek wordt beschreven in hoofdstuk 4 en gaat over de organisatie van representatie in een nieuw ANN-model waarin neuronen gemodelleerd zijn als harmonische oscillatoren. Dit model wordt het Membrain-model genoemd omdat de golfbeweging in het activatiepatroon overeenkomsten vertoont met de golfbeweging in een trillend membraan. Uit analyse blijkt dat het netwerk elke invoer moduleert tot een interfererend patroon van activatiegolven. De invoer wordt geprojecteerd op de ruimte van natuurlijke golfpatronen van het netwerk die ieder met een unieke frequentie trillen. Door een bepaalde gewichtenmatrix te kiezen kunnen de natuurlijke patronen afgestemd worden op identificerende kenmerken. De projectie van de invoer wordt hierdoor een kenmerkvector die direct gerepresenteerd wordt door het frequentiespectrum van het trillingsenergiesignaal. Deze mogelijkheid is getoetst in een Membrain Cellulair Neuraal Netwerk (MCNN). Hierin zijn de neuronen georganiseerd in een twee-dimensionaal rooster waarin alleen naburige neuronen met elkaar verbonden zijn. Bovendien zijn de neuronen aan de rand doorverbonden met de neuronen van de tegenoverliggende rand, zodat het oppervlak feitelijk onbegrensd is. Hierdoor en door het cellulaire karakter van de neuronen zijn de natuurlijke golfpatronen van het netwerk translatie-invariant. Met

behulp van een simulatie wordt bij wijze van voorbeeld gedemonstreerd dat het mogelijk is om handtekeningen te herkennen ongeacht een translatie van het beeld.

In hoofdstuk 5, tenslotte, worden de resultaten van het onderzoek geëvalueerd. Hierbij staat de organisatie van representatie in ANNen centraal. In het beschreven onderzoek is sprake geweest van drie organisatietypen die hebben geleid tot vier verschillende representatievormen, namelijk een gedistribueerde, een op patronen gebaseerde, een modulaire en een temporele representatievorm, of tot een mengvorm die bestaat uit een combinatie van deze vormen. De conclusie van dit proefschrift luidt dat de kracht van ANNen gelegen is in een efficiënte integratie van genoemde representatievormen door connectionistisch samenspel van verbindingen, neuronen en activatie.

Curriculum Vitae

Hans Henseler werd op 26 juli 1964 geboren in Den Haag. Van 1976 tot 1982 volgde hij het Athneum B aan het Norbertus College in Roosendaal. Van 1982 tot 1987 studeerde hij informatica aan de Technische Universiteit in Delft. Hier werd hij in 1985 studentassistent van Dr. H.J. van den Herik bij het college "Inleiding Artificiële Intelligentie Technieken". Door de aard van deze werkzaamheden was hij nauw betrokken bij het onderzoek naar parallelle expertsystemen. In 1986 was hij gedurende 1 jaar werkzaam in Delft bij Westmount Technology B.V., waar hij cursussen Prolog, C en Unix heeft gegeven. In datzelfde jaar was hij tevens op de TUD werkzaam als studentassistent bij het Prolog-practicum. In 1987 verrichtte hij met twee medestudenten onderzoek op een parallelle NCube/4+ computer. Dit onderzoek resulteerde in een gezamenlijk afstudeerverslag getiteld "The Design of a Parallel Knowledge-based Optical Character-Recognition System" op basis waarvan hij in november 1987 zijn ingenieursexamen behaalde.

In 1988 werd hij toegevoegd onderzoeker bij de vakgroep Informatica aan de Rijksuniversiteit Limburg in Maastricht. Hier heeft hij tot 1992 onderzoek verricht aan Artificiële Neurale Netwerken onder supervisie van Prof.Dr. H.J. van den Herik en Dr. P.J. Braspenning. De belangrijkste resultaten van dit onderzoek zijn in dit proefschrift beschreven. Naast het onderzoek heeft hij onder andere meegewerkt aan een cursus Neurale Netwerken voor het bedrijfsleven en een probleem-gestuurde cursus Neurale Netwerken voor studenten van de Faculteit der Economie. Ook is hij enige tijd redactielid geweest van de Nieuwsbrief van de Nederlandse Vereniging voor Kunstmatige Intelligentie.

Direct aansluitend op zijn werkzaamheden in Maastricht is hij op 1 maart 1992 gaan werken als wetenschappelijk medewerker bij het Gerechtelijk Laboratorium van het Ministerie van Justitie in Rijswijk. Daar is hij verantwoordelijk voor de ontwikkeling van de sectie Forensisch Computeronderzoek die Politie en Justitie ondersteunt bij de opsporing van en bewijsvoering in misdaden waarbij geautomatiseerde systemen een belangrijke rol spelen.