MASTER

The Adaptive Resonance Theory network : (clusterings)behaviour in relation with Brainstem Auditory Evoked Potential patterns

Freriks, L.W.

Award date:
1993

5038

FACULTEIT DER ELEKTROTECHNIEK
TECHNISCHE UNIVERSITEIT EINDHOVEN
VAKGROEP MEDISCHE ELEKTROTECHNIEK

The Adaptive Resonance Theory network: (clusterings)behaviour in relation with Brainstem Auditory Evoked Potential patterns.

by: L.W. Freriks

Rapport van het afstudeerwerk
uitgevoerd van januari 1991 t/m oktober 1991
in opdracht van Prof. Dr. Ir. J.E.W. Beneken
onder leiding van Dr. Ir. P.J.M. Cluitmans en Ir. M.J. v. Gils

...toch namen zij zijn leven
en lieten slechts traan en woede.

Ter nagedachtenis aan
Cor Poort.

Contents

Samenvatting

Dit rapport gaat over neurale netten en hun mogelijke toepassing in patroon herkenning. Het beschrijft een studie van het *Adaptive Resonance Theory* (ART) netwerk in samenhang met een patroon herkennings probleem: het herkennen van karakteristieke eigenschappen in *Auditieve Evoked Potential* (AEP) patronen.

In de Vakgroep Medische Elektrotechniek van de Technische Universiteit Eindhoven probeert men automatisch karakteristieke eigenschappen in AEP patronen te herkennen omdat er een mogelijk verband bestaat tussen veranderingen die optreden in AEP patronen en variaties in de *anesthesie diepte*. Dit kan opgevat worden als een patroon herkennings probleem. Neurale netten kunnen een gereedschap zijn om dit soort problemen op te lossen. Daarom is er als eerste een literatuurstudie verricht om te onderzoeken welke netten eventueel een gereedschap kunnen zijn om het AEP patroon herkennings probleem op te lossen. Naar aanleiding van die studie kan worden geconcludeerd dat er 3 netwerken zijn die het AEP probleem mogelijk op kunnen lossen, te weten: het Multi Layer Perceptron, het Counter Propagation Netwerk en het Adaptive Resonance Theory netwerk. In 3 afzonderlijke projecten is geprobeerd het AEP probleem op te lossen met behelp van een van deze 3 netten. Dit rapport beschrijft het project waarin het ART netwerk gebruikt is. In dat project zijn twee verschillende ART netwerken behandeld: ART 1, het netwerk dat ontworpen is voor binaire patronen en ART 2, het netwerk dat ontworpen is voor analoge patronen. Dit laatste netwerk zal gebruikt worden om het AEP probleem op te lossen. Het algorithme dat de werking van dit netwerk beschrijft staat helaas nergens in de literatuur vermeld. Daarom is naar dit algorithme met behulp van "trial and error" gezocht. Daarbij werd gebruik gemaakt van resultaten van simulaties die wel in de literatuur beschreven zijn. Toen het juiste algorithme gevonden was, waren er nog een aantal onduidelijkheden met betrekking tot het ART 2 netwerk. Een zeer uitgebreide studie naar het gedrag van ART 2 heeft een aantal van deze onduidelijkheden uit de weg geruimd. Vervolgens is het netwerk gebruikt om zijn mogelijkheden in het patroon herkennings probleem te onderzoeken. Dit probleem werd in dit project niet opgelost. Wel is duidelijk geworden, dat het criterium dat ART 2 gebruikt om AEP patronen te clusteren heel duidelijk vast ligt. Daardoor valt te verwachten, dat het niet makkelijk zal zijn om het AEP patroon herkennings probleem op te lossen met behulp van het ART 2 netwerk.

Summary

This report discusses about neural networks and pattern recognition. It describes the study of the *Adaptive Resonance Theory* (ART) network in relation with a pattern recognition problem: extraction of features from *Auditory Evoked Potential* (AEP) patterns.

In the group Medical Electrical Engineering of the Eindhoven University of Technology it is tried to extract features from AEP patterns automatically because there is a possible correlation between AEP patterns and variations in *aneasthetic depth*. This can be regarded as a pattern recognition problem, for which neural networks can be a solution. First, a literature research was performed in order to investigate which neural networks can be used in the AEP pattern recognition problem. From this research was concluded that three networks can be used as a possible tool for solving the AEP problem: the Multi Layer Perceptron, the Counter Propagation network, and the Adaptive Resonance Theory network. These three networks have been used simultaneously to solve the AEP problem. This report handles about the ART network. Two networks are treated here: ART 1, which handles binary valued input patterns and ART 2, which handles analog input patterns. The last network was used to solve the pattern recognition problem. For this network unfortunately no straightforward algorithm can be found in literature. Therefore, this algorithm has to be found by trial and error making use of ART 1 knowledge and results of simulations which are described in literature. The investigations which have been done in order to find this algorithm are described in this report. After the algorithm was found, there were still a lot of questions about the network which could not be answered, therefore the network has been investigated very detailed. These investigations clarified a lot about ART networks. Next, a start was made to find a solution to the AEP problem. The problem has not been solved in this study described in this report but it will be shown that the criterion to which ART 2 clusters AEP patterns will be very clear. Due to this criterion it is expected, that it will not be easy to solve this pattern recognition problem with an ART 2 network.

1. Introduction

In the *Servo Aneasthesia Project* of the Division of Medical Electrical Engineering of the Eindhoven University of Technology one tries to find out whether it is possible to apply automation during the *aneasthesia* of a patient undergoing a surgical operation. With aneasthesia a complex combination of depression of the following functions of the nervous system is meant:

1. depression of motoric functions (relaxation)
2. depression of sensory input processing (analgesia)
3. depression of autonomic reflexes (e.g. respiration and circulation)
4. unconsciousness and amnesia

If a patient is going into surgery, an adequate level of depression of the above mentioned aspects is necessary to create suitable conditions to operate and to prevent the patient from physiological damage. Too much depression can cause unrecoverable damage to the patient. Too little depression leads to premature recovery of functions of the nervous system. It is possible that the patient regains consciousness during surgery. This can lead to recall of operative events (sometimes a very traumatic experience) or even to motoric reflexes. The latter can be very dangerous to the patient.

It will be clear, that one has to be able to monitor the level of depression of the functions of the nervous system during surgery before one is able to control the depression. By looking at physiological signals and signs such as the colour of the patient, the size of the pupil, the perspiration, the heart-beat or the blood-pressure, an experienced aneasthetist is capable to determine the level of depression of these functions and express it in an overall parameter which is expected to be an estimation of *aneasthetic depth*.
This method however is not accurate even if only a few drugs are used to reach the desired depression. Nowadays, one uses combinations of drugs to control the depression of different functions separately. Control then becomes more precise but the amount of injected drugs decreases.
Unfortunately it is more difficult to determine aneasthetic depth with the methods described above if these modern balanced aneasthetisia techniques are used because the reflexes of the nervous system do not only depend on aneasthetic depth but also on the drugs used. This causes the need for the research for a monitor of aneasthetic depth which is independent of the used drugs. In the *Aneasthetic Depth Project* one tries to develop such a monitor.

A possible monitor for aneasthetic depth may be the *auditory evoked potential* (AEP) [Cluitmans, 1990]. An AEP is the response of the central nervous system to an acoustic stimulus [Jansen, 1990]. It reflects the processes in the ear and the auditory nervous pathway. An AEP is superimposed on the *electroencephalogram* (EEG). It can be extracted from the EEG by averaging periods of EEG activity measured after each of many stimuli.
Research shows, that there is a possible correlation between changes in AEP patterns and variations in aneasthetic depth [Cluitmans, 1990]. Until now this relation is not known exactly. By studying AEP signals during surgery together with observations of an aneasthetist one tries to extract physical measurable parameters which correlate with aneasthetic depth. Even without knowing the exact relation between aneasthetic depth and AEP patterns, these patterns are very useful in a first step to estimate aneasthetic depth because characteristics in the AEP seem to change when aneasthetic depth changes. Much attention is paid to the minima and maxima in the AEP and to the *latency* of these extrema. With latency the time between the acoustic stimulus and the extrema is meant.

The first step in the direction of a monitor of aneasthetic depth is a device that recognizes the special characteristics of the AEP on-line. Until now this *Pattern Recognition* task has been done off-line by human experts. Devices which have been proven to perform well in the field of pattern recognition are *artificial neural nets*. Neural networks slightly resemble the structure of the human brain. They

consist of many simulated *neurons*: simple nonlinear processing elements (*nodes*), densely interconnected, w*eights* are attached to every connection.

The most important property of neural networks is, that they are able to *learn*: they adapt their internal structure to a set of combinations of input/output patterns or cluster input patterns by changing the values of the weights in such a way that this set will be remembered. When learning a set of combinations, the net does not learn every separate combination but tries to store the relation in the input/output combinations of this set. A *learning algorithm* is responsible for this task. In this way neural networks are capable in generating an output to an input which is in agreement with this relation even if this input/output combination has not been presented beforehand.

This report describes the investigations which have been done in order to get insight whether it is possible to extract a particular maximum in an AEP with a neural network: the *Adaptive Resonance Theory network* (ART). These investigations have been executed at the Division of Medical Electrical Engineering of the Eindhoven University of Technology in the Aneasthetic Depth project.
In chapter 2 general properties of neural networks are discussed. This discussion is followed in chapter 3 by an extend overview of 9 commonly known neural networks. From these networks 3 possible candidates are selected as a tool to solve the pattern recognition problem with. One of these networks is the ART network which is the main subject of this report. In chapter 4 ART 1 will be discussed; the ART version which handles only binary valued input patterns. There it is tried to get insight in the behaviour of some parts of ART networks. Chapter 5 and 6 treat ART 2, the ART version which handles analog valued input patterns. In these chapters it is tried to get insight in why ART networks ar so complicated. Simultaneously is tried to get insight in how the pattern recognition problem can be solved. Conclusions about both investigations are summarised in chapter 7. In that chapter also some general statements about neural networks are made as well as the conclusions which could be drawn after studying ART 1.

2 Introduction to Neural Networks

In this chapter some general properties of neural networks will be discussed. It will be made clear what the meaning is of the different parts of a neural network: the neuron, the network architecture, the training algorithm and the learning rule. After a short history overview, a number of possibilities will be discussed how these 4 parts can be realised. After that a plea is made to avoid grouping of neural networks according to a property of one of the four parts.

2.1 Neural networks background

If the architecture of a computer is compared to the architecture of the human brain, there are a few properties which attract attention [Denker, 1986].

The human brain consists of approximately 10^{10} neurons. Every neuron can be seen as a very simple processing element which can perform only simple tasks [McCulloch & Pitts, 1943]. A neuron operates very slow. Neurons are interconnected via a dense network of connections. Hundreds of thousands connections per neuron are no exception. Every connection has its own strength. The way neurons are interconnected is subject of research but will probably never be completely understood. The human brain is very robust and fault tolerant: every day neurons die but the brain continues to function.

A computer has one or a few very complicated processors which consist of 10^8 to 10^{10} transistors for memory and logic functions. Each transistor can also be regarded as a very simple computing (switching) element. In contrary to the neurons in the brain it switches very quickly (10 ns). Computers are designed in a very hierarchical way and are in no sense fault tolerant. A defect in a few transistors is enough to make the machine useless.

A computer is superior in calculations and processing data. To do this well and efficient it has to be programmed. Humans, in contrary, are less efficient in doing these things. They perform well in tasks like association, evaluation and pattern recognition. They manage to do this because they learn how to do these tasks. Computers are very bad in these tasks. Even the development of quicker processors only causes marginal improvement of this.

The differences between computer and brain contributed a lot to the research for a new type of networks which are able to solve problems in which humans operate well, better than conventional machines. These networks are called *Neural Networks*. Other names used are *Connectionist models*, *Parallel Distributed Processing models* and *Neuromorphic systems* [Lippmann, 1987a]. Researchers operating in this field, believe that not the processing elements, but the connections between them are responsible for the different behaviour of computer and brain. They believe the information in the brain is stored in the connections between the neurons. A change in the interconnection structure will cause a change in stored information. If machines have to be developed which perform well in "human"-tasks, the architecture of those machines must resemble the architecture of the brain, therefore these neural nets have to meet the following demands:

1. They exist of numerous simple processing elements
2. The nodes are densely interconnected; a weight is attached to every connection
3. Connections have to be variable therefor weights must be allowed to change

Implementing these 3 demands in a network immediately gives rise to 4 very important questions:

1. How to model a neuron?
2. How to model the interconnection?

3. How to choose which weights have to change in order to store desired information in the network?
4. How to change the selected weights?

The solving of each question concentrates on one essential part of a neural network namely:

1. The node characteristics
2. The network architecture
3. The training-algorithm
4. The learning-rule

The remainder of this section discusses these 4 parts. Every part is subject of much current research. It is important to note, that together they define one neural network.

2.1.1 Modelling the neuron

A neural network consists of artificial neurons. A simple highly idealized neuron is shown in Fig. 2.1 (a). It consists of a *cell-body* (B), *dendrites (D) and an axon* (A).

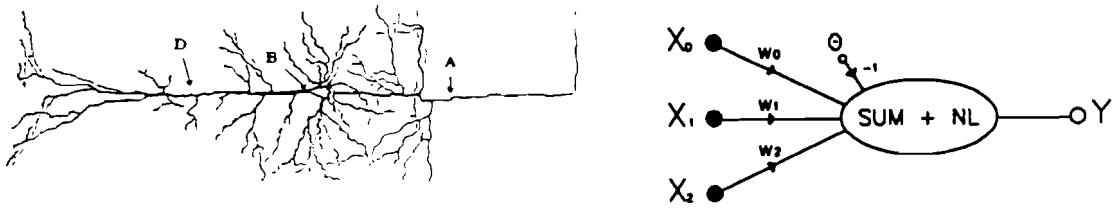

Fig. 2.1: A typical neuron (a) and an artificial neuron (b)

The dendrites are the inputs of the neuron. They are connected to the axons of other cells which are the outputs of these cells. The connection between two neurons is called a *synapse*. A synapse can be either *stimulatory*, which means that an incoming signal raises the activity level of the neuron, or *inhibitory*, which means that the incoming signal lowers the activity level of the neuron. A neuron collects all input signals. If the total input exceeds a certain *threshold level* the neuron *fires*, that is it generates an output signal. The threshold level governs the frequency at which the neuron fires.

A biological neuron can easily be modeled by an artificial neuron, in literature also called *node, unit, neurode or processing element*. The simplest most often used model can be seen in Fig. 2.1 (b). Again a cell body can be distinguished from inputs $\mathbf{x}_i$ and output **y**. The synapses can be expressed as weights $\mathbf{w}_i$ at the input. The node adds the N weighted inputs and passes the result through a nonlinearity. The node is characterised by an internal threshold θ and the type of nonlinearity **f**(.). Examples of nonlinearities are the hard-limiter and the sigmoid function, which can be seen in Fig. 2.2.

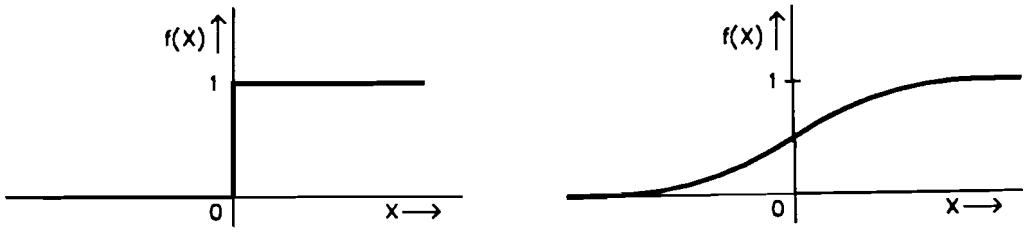

Fig. 2.2: Nonlinearities: hard limiter (a) and sigmoid function (b)

If a type of nonlinearity is chosen, the output **y** can be calculated according to eq. (1).

$$y = f\left(\sum_{i=0}^{N-1} w_i x_i - \theta\right) \quad (1)$$

With the hard limiter as nonlinearity, the node fires only if the value of summed inputs exceeds the activity level θ.

2.1.2 The network architecture

Now the neurons have been modelled, it is time to discuss the network architecture of a neural net. In a neural net many nodes are interconnected. The neural net has to be used to communicate with the outside world so *input* nodes and *output* nodes have to be discriminated. All other nodes are called *internal* nodes. The network architecture determines how nodes are interconnected. As will be discussed in section 2.3.3. this can be done in different manners. The network architecture also governs the type of connections which are used within a neural network: mono- or bi-directional.

2.1.3 The training algorithm

As mentioned before in the introduction, a neural network can be used to store a set of combinations of input/output patterns in such a way, that the relation between these patterns will be stored in the neural network. In other applications it must be able to cluster data into classes. To succeed in this, the net has to be *trained*. This means, that the connection weights must be adjusted in such a way that the relations are stored or the data is clustered correctly. A *training algorithm* is responsible for this task. From this can be concluded, that the task of this training algorithm is twofold:

1. determine which weights have to be changed in order to reach the desired property.
2. change the weights.

The set which is used to train the network is called the *training set*. This training set must contain a good representation of the relation which has to be stored into the network.
The training algorithm depends very heavily on the network architecture. In literature thousands of training algorithms can be found, unfortunately very often with minimal differences. The most important step in the training algorithm is the adjustment of the selected weights.

2.1.4 The learning rule

Adjustment of weights is considered to be *learning* in a neural net. It is very important to distinguish training and learning. In literature these two terms are unfortunately very often confused, therefore it is important to state here, that:

In this report, learning is considered to be the adjustment of the weights. Learning is governed by a learning rule which states how selected weights have to be adjusted. With the training algorithm the selection of the weights together with their adjustment is meant. Therefore learning is a part of the training algorithm.

If one succeeds in storing a relation between a set of input/output combinations, this is very useful because then one is able to generate an output to an input which satisfies that relation, even if the input has never been applied to the network before. The network can thus be seen as a network that generalizes. In the following chapters will be made clear what generalisation has to do with association and pattern recognition. Then it also will become clear why neural networks are very useful tools in the field of *speechrecognition, vision, robotics* and *Artificial Intelligence*.

In neural network research one is inclined to adjudge all kinds of human qualities to the networks. It is very dangerous to do this because this creates expectations which can not be met. Moreover it does not clarify the subject but often hampers easy understanding because it is not clear what is exactly meant by these qualities. Unfortunately it sometimes is difficult to avoid this nomenclature. Take account for this when human like qualities are discussed in this report.

2.2 History

The history of neural network research starts in 1943 with the work of neuropsychologist W. McCulloch and logician W. Pitts [McCulloch & Pitts, 1943]. Before 1943 one considered the brain as a collection of densely interconnected neurons. Connections could be placed or removed but not changed. McCulloch and Pitts introduced a *McCulloch and Pitts model* (MCP) in which connections could vary continuously. This MCP model has been discussed in section 2.1.1. Although this model is not capable of learning, it provided a lot of insight in this new field.
In 1949 D.O. Hebb discussed an important rule concerning the adaptation of weights in neural nets. In *The organization of behaviour* [Hebb, 1949] he stated: "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 places in one or both cells such that A's efficiency, as one of the cells firing B, is increased". In other words if: two neurons are both active at the same time, the synaptic connection between them has to be reinforced. Synapses of this kind are called *Hebb synapses*. The learning rule which results from this statement has become one of the two mostly used learning rules and is known as the *Hebb rule*.
In 1958 the psychologist F. Rosenblatt introduced the first precisely specified computational oriented neural network: *the perceptron* [Rosenblatt, 1958]. In this simple model Rosenblatt used one layer of MCP nodes as a pattern recognizer. An important step in history was, that Rosenblatt was the first one who decided to compute neural networks and was the first one who introduced a brain model which could "do" something. Rosenblatt showed that the perceptron was capable to perform associations but that there was a limit on the capacity. He also showed that the perceptron had an ability to generalize and that because of the distributed memory, the perceptron was insensitive to (a limited amount of) damage.
M. Minsky and S. Papert criticized this perceptron in their book *Perceptrons* [Minsky & Papert, 1969]. In this book they discuss the limitations of Perceptrons. They prove, that there are a lot of functions which can not be constructed in a one layer structure like a perceptron. One of these functions is the XOR function. It was commonly known, that these problems could be overcome if multi layer structures were used. Kolmogorov introduced a theorem in the late 1950's in which he stated that any continuous function can be implemented exactly by a 2 layer (2 layers of perceptron nodes!) feedforward network if every layer consists of enough nodes [Kolmogorov, 1957]. The problem at that moment was, that this theorem couldn't be successfully translated into an algorithm which was able to train neural nets. After 10 to 20 years of very successful developments the research in this field seemed to be stuck. Many projects were cancelled. The future of neural networks seemed to be very bad.
Despite the loss of support a few researchers continued their research. S. Grossberg was one of the most productive among them. He studied the so called *unsupervised models* and introduced a lot of new models like for example the INSTAR, the OUTSTAR and the Adaptive Resonance Theory (ART)[Grossberg, 1976]. T. Kohonen studied the self organizing feature maps [Kohonen, 1984]. K. Fukushima introduced his (Neo)Cognitron [Fukushima, 1975]. J. Hopfield developed his Hopfield net [Hopfield, 1982] which could be used as a Content Addressable Memory and which lead to the development of the Boltzman machine [Ackley & Hinton, 1985]. The big revival of neural network research came in the late 1980's.

In 1986 D. Rumelhart, G. Hinton and, R. Williams introduced an algorithm which is able to train multi layer perceptrons [Rumelhart et. al, 1986]. This algorithm is called the *Back Propagation Algorithm* (BPA). The learning rule used in the BPA is based on the *Delta rule* introduced by B. Widrow and M.E. Hoff [Widrow & Hoff, 1960]. Widrow and Hoff stated, that perceptrons under certain well

specified conditions can indeed classify items in an input set correctly but, that it takes a long time for the weights to converge. They introduced the ADALINE (a perceptronlike ADAptive LINear Element) together with the Delta rule in which they make use of the error signal between a generated output after applying an input and the desired output in order to adjust the weights. They were interested in minimizing the error function so the weights will be adjusted according to *gradient descent* methods.
In the BPA a generalization of this Delta rule is used. In this algorithm the error signal propagates back to the hidden layers in order to adapt the weights in those layers. The BPA showed to be very powerful. It seemed to operate especially well during simulations. Therefore it is not amazing, that it was this algorithm which started the revival in the field of neural networks.

Nowadays this field is not only wide, it is also a multidisciplinary one. Biologists, (neuro)psychologists, mathematicians, and engineers study neural networks. Some have the purpose to get insight in the behaviour and structure of the human brain. Others hope they can use neural networks as a tool to work with in order to solve their problems. Research is also done to develop hardware applications of neural networks.

It is doubtfully whether this will lead to "artificial brains", "thinking" machines or even "young Frankensteins" in the near future, but that it is an interesting field of research is beyond dispute.

2.3 Neural network taxonomy

Entering the field of neural networks, is entering a field in which nomenclature and grouping is rather confusing. In section 2.1 has been mentioned, that a neural network consists of 4 parts:

1. The neuron model
2. The network topology
3. The training algorithm
4. The learning rule

Profound research is done on every item very intensely but small changes in existing networks lead, if successfully applied, very often to new names for networks, topologies, algorithms and learning rules. In this way there seem to exist numerous "different" networks, topologies etc. described in literature. After literature study they bear more resemblance then in the first instance had been expected. Grouping of neural networks gives rise to the same problems; many categorisations exist. To avoid, that a new categorisation will be introduced in this report, the various criteria to which a neural net can be grouped are discussed in the remainder of this section. In chapter 3 the most important and interesting nets will be discussed.

2.3.1 Grouping criteria for neural networks

In order to group neural nets it is again important to distinguish properties which have something to do with the neuron model, the network architecture, the training algorithm and the learning rule. This enumeration is not complete. Lots of other network architectures are subject of current research. Some of them will be discussed in other parts of this report.

For the neuron model can be distinguished:

Binary input	vs.	Analog input

For the network architecture:

Feedforward	vs.	Feedback
Fully connected	vs.	Modular

For the training algorithm:

Constructed	vs.	Trained
Supervised	vs.	Unsupervised
Adaptive	vs.	Nonadaptive

And for the learning rule:

Hebbian learning	vs.	Delta learning

Each grouping criterium will be elucidated hereafter.

2.3.2 Neuron model grouping

A criterium of the network which can be used to come to some grouping of neural networks is whether the networks uses binary valued or analog input and output patterns. If a net is developed for binary patterns the nodes take values 0 and 1 or -1 and +1. If analog valued patterns are chosen the values of the nodes can be every real value within the range of the computer. Another criterium can be the node type: sigmoid, hardlimiter etc. This last criterium is not a very important criterium to distinguish between neural networks. First, because there are a lot of different types and second because the node type doesn't give any insight in the network "as a whole".

2.3.3 Network architecture grouping

Looking at the network topology, there are again some distinctions which can be made. A network can be a feedforward network, a feedback network or a combined feedforward/feedback network. In a feedforward network information always goes from the input-side of the network to the output-side. In a feedback network information can also be fed back in some parts of the neural network. Adjustment of the weights is not considered as a part of the network topology in these definitions. It is strictly speaking unnecessary to stress this, because learning is not a part of the network topology, but misconceptions can arise if for example the Delta rule is used in a network. Then it looks like there is always a kind of feedback from the output to the weights.
An important part of the architecture describes how the nodes in the neural network are connected. A possible neural net is a *fully interconnected* network. In this network every input node is connected to every internal node. Every internal node is also connected to every output node. The topology which exists in this way reflects indeed some elements of the human brain: many neurons which are densely connected.

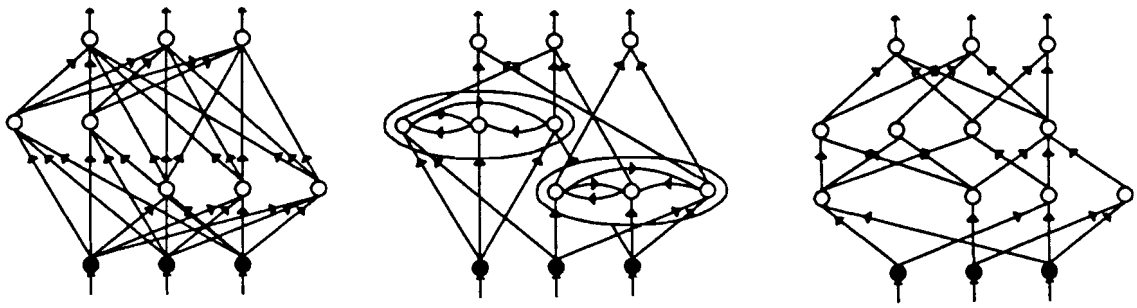

Fig. 2.3: Network topologies: fully connected (a), modular (b) and layered (c)

For simulations this can be a real problem because most calculations have to be executed sequentially in stead of parallel so this can take a lot of time. A possible solution to this is to group nodes together. Nodes within one group are densely connected. Compared to this density the connections between the different groups are sparse. These *modular networks* make use of the fact that there is a kind of

topological dependency in the brain: A neuron has more influence on the behaviour of a neuron in its neighbourhood then on a neuron "far away" in the brain. Another type of network is the *layered* network. In this network input- internal- and output-nodes are layered. The nodes in one layer are only connected to (all) nodes in the previous layer. Often more than one internal ore hidden layer exists. Models of the 3 mentioned networks are shown in fig. 2.3.

2.3.4 Training algorithm grouping

The most important distinctions between networks concern the training algorithm. This is sometimes very difficult because differences can be very small.
A simple distinction is the one between constructed and trained algorithms. In a constructed network, the weights are calculated from a set of input and output patterns. After these calculations the network is set and can be used. In a trained network the weights are adapted in subsequent iterations. This affects the information stored in the network continuously. Training can originally be divided into two groups of major importance: *supervised* and *unsupervised* training. Lately another type has been introduced: *reinforced* or *graded* training.

In supervised networks an input and output pattern are applied to the net simultaneously. Therefore in a supervised net a priori information about what the net should do is needed. The output pattern is called the *desired output, the target* or *the teacher.* During training the weights are adapted in such a way that the different presented input/output pairs are stored into the network.

In unsupervised neural networks there is no teacher. In those nets an input pattern only is applied thus it is not possible to store input/output relations. An unsupervised neural network is a net which clusters input patterns into categories. Every output node in an unsupervised network represents a category. The neural network has to "decide" to which category the input belongs most likely. Every category is related to features which can possibly occur in the input. The network has to extract features from the input in order to decide to which category the input belongs. Competition between possible features has to take place in order to decide this. If the right category has been found, the related node (often called *winning* node because the node is related to the feature which has won the competition) is pointed at. The weights between this node and the input are adapted to emphasise the relation between this node and the input. At this moment the input is associated with the winning node. During learning the network *self organises* in these unsupervised learning schemes. The strategy discussed above is called *competitive learning.* This name is in contradiction with what is considered to be learning in this report because in the competitive learning scheme *learning* is not mentioned. Thus competitive learning describes a training algorithm.

In graded trained networks the training strategy is similar to supervised training but instead of applying the desired output to the network during each learning trail, it gets a score that tells how well it has done after a sequence of trails. The score is calculated according to some *cost function* or *energy function.* Many different cost functions have been studied in the past. Although graded training is in between supervised and unsupervised training it is important, that this training algorithm is not confused with combined unsupervised/supervised networks in which unsupervised training is used to cluster the inputs and supervised training to label the output nodes of the network.

If a network is trained it can be used for the purpose it was trained for. Some networks never change after being trained once. These networks are called *nonadaptive* networks. Others are always able to change their weights if necessary. These are the *adaptive* neural networks.

2.3.5 Learning rule grouping

Learning is considered as the adjustment of weights. Taking this into account when studying learning rules, it becomes clear that the learning rules which are used are often derivates of one of the two

earlier mentioned learning rules: the Hebb rule or the Delta rule. Both rules were discussed in section 2.2.

For both rules a mathematical expression can be given. For the Hebb rule this expression is:

$$\Delta w_{ij} = k x_i y_j \tag{2}$$

This expression states that the weight change is proportional to the product of input and output. It expresses the functioning of Hebb synapses exactly: the weights change if both input and output are active. The amount of weight change is governed by parameter k, a proportional constant called the *learning rate.*

The mathematical expression for the Delta rule is:

$$\Delta w_{ij} = k(y_j - y_{j\cdot}) \tag{3}$$

where $y_{j\cdot}$ is the desired output.

From expression (3) can be concluded, that if the network converges (which means that $\Delta w_{ij} \rightarrow 0$) y_j converges to $y_{j\cdot}$ and therefore $\underline{y}$ converges to $\underline{y}_{j\cdot}$. Again, the convergence speed is governed by the learning rate.

2.3.6 Composing a neural network

The way the 4 parts of a neural net are described in this chapter suggests that development of neural networks is nothing but choosing a neuron model, a network architecture, a training algorithm, and a learning rule. This is not true because these choices depend on each other. If, for example the Delta rule is used in a network, this net has to use a supervised training algorithm. An unsupervised net excludes the Delta rule as its learning rule. If one chooses to use an unsupervised training algorithm one forces oneself to add a classifier to the net topology. From this can be concluded that, although the parts of neural nets can be distinguished very clearly, their functioning depends on each other. However, this doesn't mean, that if a choice is made for one particular part of the neural network this will always influence choices of the other parts. Taking this into account, this is the right moment to point out some criticism.

As mentioned before, nomenclature in the field of neural networks is rather confusing. On the one hand because nets which resemble very much can not be recognized quickly as such because their "inventors" assigned totally different name to the nets. On the other hand, networks which do not have much in common can be grouped together because for example they use the same learning rule or have the same node characteristics. In this way the grouping is rather confusing: it suggests to give a lot of insight in the nets which are grouped together but it only gives insight in one property of the neural nets.
To overcome these problems it is important that neural networks are treated in the same way in literature. A possible way to do this is:

1. Always mention choices made for the 4 parts when treating a neural network.
2. Avoid to come to some grouping of neural networks.

If these 2 aspects will be satisfied in neural network research it will make a quick literature research much easier. A start to do this is made in this report. In the next chapter 9 well known neural networks will be treated. For every node the properties of the 4 parts will be mentioned. There will be shown, that a Multi Layer Perceptron for example can be seen as a neural network with a layered combined feedforward/feedback architecture with hardlimiter or sigmoid nodes which will be trained

supervised and uses a Delta learning rule. The fact that it indeed belongs to the so called *hyperplane classifiers* [Lippmann, 1989] or that it is a *mapping network* will not be mentioned (there).

3 Neural Network models

In this chapter 9 neural networks will be discussed and compared. After that will be examined which net can be useful to solve the AEP pattern recognition problem.

3.1 The (Multi Layer) Perceptron and the decision tree

3.1.1 The Perceptron

The Perceptron was developed by F. Rosenblatt [Rosenblatt, 1958]. It has already been treated in section 2.1.1 implicitly from which could be concluded that the Perceptron equals an artificial neuron model. This artificial neuron model is shown again in Fig. 3.1.

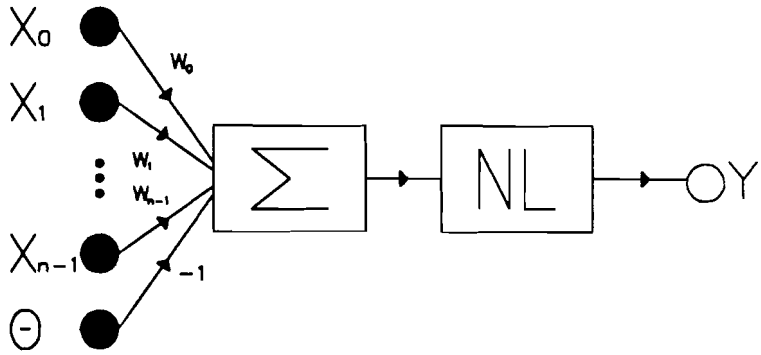

Fig. 3.1: A Perceptron

The Perceptron is a nonlinear summing element. It has N inputs x_i ($O \le i \le N-1$) and one output **y**. The N inputs are weighted and summed together with a threshold θ. This result is passed to a nonlinear element. This element usually performs a sigmoid function or a hard limiting function **f(.)**.

The Perceptron can be described with the following equation:

$$y = f\left(\sum_{i=0}^{N-1} w_i x_i - \theta\right) \tag{1}$$

If a hard limiter is chosen, this holds:

$$\begin{aligned} y &= +1, \quad if \sum_{i=0}^{N-1} w_i x_i \ge \theta \\ y &= -1, \quad if \sum_{i=0}^{N-1} w_i x_i < \theta \end{aligned} \tag{2}$$

From this can be concluded, that the Perceptron forms a decision boundary. This decision boundary can be described as:

$$\theta = \sum_{i=0}^{N-1} w_i x_i \tag{3}$$

which is a hyperplane in N dimensional space. This hyperplane divides the N dimensional space in two parts. In one part the inputs $\underline{x}$ for which $\underline{w}.\underline{x} > \theta$ are situated while $\underline{w}.\underline{x} < \theta$ holds for the inputs $\underline{x}$ in the other part. The product $\underline{w}.\underline{x}$ determines in which region an input $\underline{x}$ belongs and therefore a Perceptron is said to be able to create *decision regions*. (This property makes a Perceptron useful in the field of pattern recognition.) Imagine a 2 input Perceptron. The hyperplane then becomes a line in 2 dimensional space. This line divides the input space in 2 parts. Imagine 2 groups of patterns. If these 2 groups can be separated by a line, a Perceptron can be used to separate them by finding a proper $\underline{w}$.

The proper choice of $\underline{w}$ can be found by training the Perceptron. The algorithm used to train the Perceptron is described in Box 1.

Box 1: Perceptron Algorithm

Step 1:

Initialize the weights $w_i(0)$ to small initial values and threshold θ to a small value between 0 and 1.

Step 2:

Present an input $\underline{x}$ to the network together with the desired output y^*.

Step 3:

Calculate the output signal y according to (1).

Step 4:

Adapt the weights:

$$\Delta w_i = \eta(y^* - y)x_i \qquad (4)$$

according to the Delta rule.

Step 5:

Repeat by going to step 2.

In Fig. 3.2 is shown how the decision boundary develops into a boundary which indeed distinguishes two classes. In this example a pair of coordinates is applied to the system together with an output which expresses what kind of figure can be found at that place. In the X-Y plane two groups can be separated crosses (**X**) and circles (**O**).

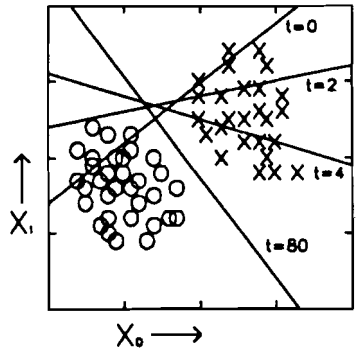

Fig. 3.2: Classification by a Perceptron

This example shows, that it is possible to train a neural network to separate between classes. In this case a Perceptron is used as a neural network. The Perceptron has to be trained supervised using the

Delta rule. The only parameter which influences the training algorithm is η. This parameter is called the *learning rate*. Rosenblatt stated, that a Perceptron is always able to separate between classes if the input classes are *separable*. If input classes can be separated by a hyperplane in N dimensional space, a Perceptron is able to do this. Unfortunately this shows immediately the major limitation of a Perceptron: there are a lot of problems in which the classes are not *linear separable* therefore these problems can not be solved with a Perceptron.
An example of such a function is the Exclusive-Or function. In Fig. 3.3 this function is sketched.

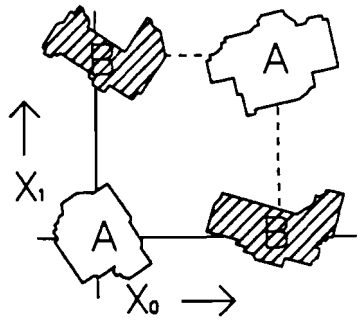

Fig. 3.3: Exclusive-Or problem

Two output classes can be separated: Class A and class B . Fig. 3.3 makes clear that A and B can not be separated by one decision boundary so the Perceptron is not able to store the XOR function.

Fortunately the Perceptron can be extended to a *Multi Layer Perceptron* (MLP) which is able to store more complex decision functions. This MLP will be treated in the next section.

3.1.2 The Multi Layer Perceptron

A MLP is a neural network which consists of an input layer $\underline{x}^0$, L-1 *hidden* or *internal* layers $\underline{x}^l$ ($1 \leq l \leq L-1$) of Perceptron nodes and one output layer $\underline{y}$ of Perceptron nodes (for simplicity this layer will be referred to as $\underline{x}^L$. Every node in the MLP is connected to all the nodes in the preceding layer. Every connection has a variable connection strength w_{ij}. The name of a MLP denotes the number of hidden and output layers. With a L-layer MLP a MLP with one input-, L-1 hidden- and one output-layer is meant. L denotes the number of Perceptron layers. Within a MLP the same formulas as in a Perceptron are valid. A 2-layer MLP is presented in Fig. 3.4.

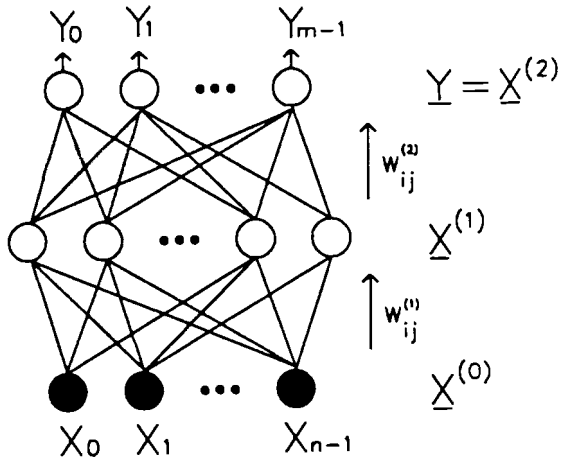

Fig. 3.4: 2-layer Multi Layer Perceptron

The MLP is related to the theorem of Kolmogorov which states, that: if $f:[0,1]^m \rightarrow R^n$, there exists a three-layer neural network that implements the function f exactly [Kolmogorov, 1957]. Kolmogorov even added demands for the layer dimensions to his theorem: m input nodes, 2m+1 hidden nodes and n output nodes. To store a function f into a MLP, the network has to be trained. In a Perceptron this is done by applying an input and a desired output to the network. The network calculates the actual output and adjusts the weights to an output node proportional to the difference between actual output and desired output of that node.
This does not work for a MLP because there is no direct path between an input node and an output node. If an output error is calculated, it is not clear which weights have to be adapted. A solution to this might be the calculation of the error function for each layer. However, it is not possible to calculate the errors directly because the targets for hidden layers are unknown. This problem gives an indication how to train the MLP: if the errors can't be calculated directly, calculate them indirectly. This can be done by the *Back Propagation Algorithm* (BPA).

Box 2: Back Propagation Algorithm

Step 1:

Initialise the weights w_{ij}^l and offsets θ_j to small random values.

Step 2:

Present input $\underline{x}$ and desired output $\underline{y}^*$ to the network.

Step 3:

Calculate the actual output by applying:

$$x_j^l = f\left(\sum_{i=0}^{N^{l-1}-1} w_{ij}^l x_i^{l-1} - \theta_j^l\right) \qquad (5)$$

in a feedforward calculation which means that the output values of the Perceptrons are calculated for every j ($0 \le j \le N^{l-1}-1$) from l=1 to l=L. l=1 denotes the first hidden layer and l=L the output layer. N^{l-1} the number of Perceptron nodes in layer l. f() is the sigmoid or hardlimiter nonlinearity.

Step 4:

Adapt the weights by calculating back the errors from output nodes to the first hidden layer according to:

$$\Delta w_{ij}^l = \eta \delta_j^l x_i^{l-1} \qquad (6)$$

w_{ij}^l is the connection strength from input- or hidden-node x_i^{l-1} in layer l-1 to hidden- or output-node x_j^l in layer l. η is the learning rate and δ_j^l is the error term for node x_j^l. The value of δ depends on whether node x_j^l is an hidden- or an output-node. If x_j^l is an output node:

$$\delta_j^L = y_j(1-y_j)(y_j^*-y_j) \qquad (7)$$

with $y_j(1 - y_j)$ as the derivative of the sigmoid function.

If x_j^l is a hidden node:

$$\delta_j^{l-1} = x_j^{l-1}(1-x_j^{l-1})\sum_{\forall k} \delta_k^l w_{jk}^l \qquad (8)$$

Step 5:

Repeat by going to step 2.

The BPA was rediscovered by Rumelhart, Werbos and others [Rumelhart et al, 1985], [Werbos, 1974]. Rumelhart and other members of the PDP-group developed the algorithm into a useful technique to

train the MLP for binary and discrete input patterns. In the BPA the problems mentioned are solved by calculating the output error between the actual output and the desired output and propagate this error in an appropriate way back to the hidden layers. In this way an error is known for every Perceptron in the MLP and the weight changes from the preceding layer to the Perceptrons can be calculated proportional to the error. The most important step in the BPA is step 4, in which the error functions are calculated. Note that the calculation of the output errors differs from the calculation of "hidden" errors.

With this training algorithm, which is a supervised training algorithm for a feed forward neural network, the following *energy function* or *cost function* is minimized:

$$E = \frac{1}{2}\sum_{\forall j}(y_j^* - y_j)^2 \tag{9}$$

in which E is the energy function for one presented pattern. It is possible to choose another cost function but this results in other learning rules. The training algorithm described in Box 2, is a consequence of the fact that a *gradient descent* method is used to minimize E [Hecht-Nielsen, 1989a]. In this method the adjustment of weights is such that it results in a *steepest descent* movement along the cost function. Due to the shape of the energy function, two disadvantages of this method occur when this method is used. The first disadvantage is that the minimizing process proceeds very slowly due to the very flat slope of the energy function at some places. As a consequence of this, the BPA is usually very slow. The second disadvantage is, that the minimizing process can get stuck in a local minimum instead of reaching the desired global minimum of the cost function E. Hecht Nielsen and others proved the existence of local minima [Hecht-Nielsen, 1989b].

In literature lots of solutions for these problems are described. A possibility to increase the convergence speed of the BPA is to add a *momentum* term α to the learning rule:

$$\Delta w_{ij}^l(t+1) = \eta \delta_j^l x_i^{l-1} + \alpha \Delta w_{ij}^l(t) \tag{10}$$

with $0 \leq \alpha \leq 1$.

If this learning rule is applied, convergence speed is sometimes higher because weight changes increase if subsequent weight changes are in the same direction. If not, weight changes decrease.
Another possibility to increase convergence speed (which results in reduced training time) is to minimize the size of the network. The network has to be big enough to solve the problem but not so big that too many parameters have to be estimated with limited training data.
The size of the network is determined by the number of input-, hidden- and output-nodes. If the number of input and output nodes is known, only the number of hidden nodes can be varied. A relation between the number of hidden nodes H, the number of inputs nodes d and the maximum number of linearly separable regions M was derived by Mirchandari [Mirchandari, 1989]:

$$M(H,d) = \sum_{k=0}^{d} \binom{H}{k} \tag{11}$$

This relation calculates the maximum number of linearly separable regions for given network dimensions. Although it is often not known how many separable regions are needed, this formula is a good start for dimensioning a MLP.

It can never be guaranteed that a global minimum will be reached with a MLP, although the chance that this will happen can be enlarged. To do this, one may start with a large learning rate and reduce this learning rate during training. In this way the weight adjustment becomes smaller during training. The chance that local minima will be skipped is large at the beginning of training.

The MLP has proven to be a very powerful network if trained with the BPA. Until now it is still the most used neural network architecture. It has been used in a lot of different applications e.g.:

- classification of speech sound
- transforming texture to phoneme rules (NETTALK) [Sejnowski & Rosenberg, 1986]
- discrimination of underwater sonar returns [Gorman & Sejnowski, 1988]

3.1.3 Decision tree classifiers

A totally different classifier which also classifies input patterns by dividing the input space in two parts by hyperplanes subsequently is the decision tree classifier. It has been developed during the last ten years. Decision tree classifiers form decision regions by performing simple operations on input features. They can handle both continuously valued and discrete input patterns. Their size can easily be adapted to the problem complexity. Unfortunately their training is complex.

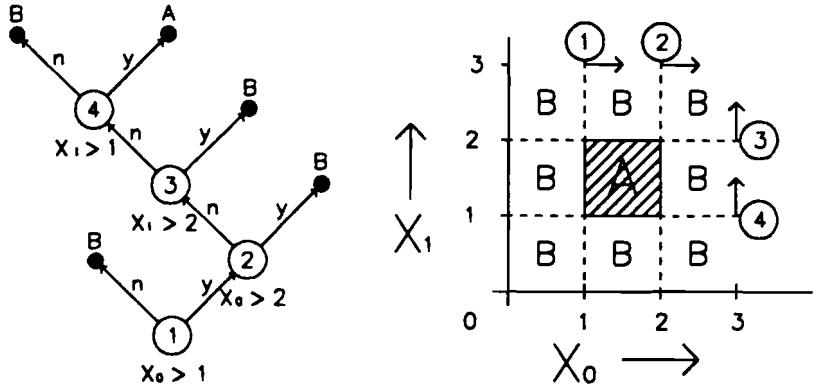

Fig. 3.5: (a) Decision tree classifier which (b) separates between classes

In Fig. 3.5 (a) a binary decision tree is sketched. This classifier consists of two types of nodes: *nonterminal-* or *decision-* nodes and *terminal-* or *output-* nodes. Every decision node divides the input space into two pieces. If a terminal node is reached the decision process ends. The class denoted by that terminal node is associated with the input vector. How the decision process works is shown in Fig. 3.5 (b). A point (x_0, x_1) is chosen in the (x,y) plane. The decision tree has to decide whether this point belongs to class A or B in Fig. 3.5 (b). One starts at the bottom of the decision tree and "answers" the questions in the tree. The direction of the right answer is followed until a terminal node is reached. If (a) and (b) are compared it becomes clear, that every question at a particular decision node corresponds to one decision region in the (x,y) plane. In this plane the number added to these regions correspond to the node numbers. The "yes direction" is pointed in by arrows. The decision regions in this example are very simple. More complex decision regions can be approximated by binary trees with many nodes and by using two or more variables at one decision node.

To make use of a decision tree for classifying inputs, the tree has to be formed or the net has to be trained. This training is complex but fast because a local cost function has to be minimized at every stage of training. Demand is, that data is sorted or ordered along each input dimension during training. In this way trees based on thousands of training examples can be trained very quickly. Comparison with a MLP yields small differences in behaviour but major improvement in training speed [Fisher & McKusich, 1988].

The training algorithm of the decision tree will not be treated in this report. The decision tree is mentioned here as another example to classify inputs.

3.2 The Hopfield network, Hamming network, and the Boltzmann machine

3.2.1 The Hopfield network

The Hopfield network was introduced by J. Hopfield in 1982 [Hopfield, 1982]. It is a one layer neural network which consists of N input nodes y_j with $0 \leq j \leq N-1$. Each node performs a hardlimiter nonlinearity. Its values can be either -1 or +1. The outputs of the hardlimiter nodes are fed back to the inputs by a fully interconnected network. The connections in this network have variable strength w_{ij}. In Fig. 3.6 a Hopfield network is sketched.

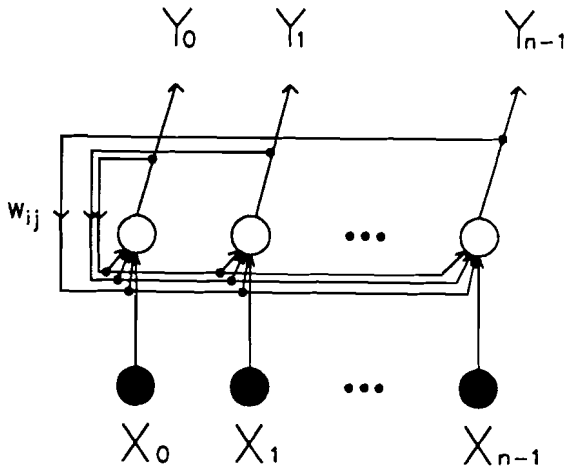

Fig. 3.6: A Hopfield network

The Hopfield network is an *associative memory*. First, M patterns are stored into the network. Then the network is developed in such a way, that its output will always converge to one of these M patterns if an input is applied. In this way the network *associates* an input pattern with one of the patterns stored in *memory*. The algorithm which is responsible for this task is described in Box 3. It is called the Hopfield algorithm.

From Box 3 (page 18) it becomes clear, that the Hopfield algorithm differs considerably from the earlier discussed algorithms. The algorithm consists of two parts. In the first part (step 1) the patterns are stored. In the second part (step 2 and 3) inputs are applied to the network and the network has to associate this input with one of its stored patterns. In Box 3 this task is only performed once. It will be clear that it may be repeated again and again. If this algorithm is compared with other algorithms it becomes clear that Part 1 can be seen as a training algorithm. Part 2 describes the use of the Hopfield network.

If Part 1 is compared to other training algorithms another difference becomes clear. In step 1 the weights are set. Instead of converging to a steady state, the weights are calculated directly "on line". Whether this can be called learning is discussable.

In Part 2 the use of the Hopfield network is described. Hopfield [Hopfield, 1982] proved that in the iteration process discussed in Box 3 the network converges to one of the M stored patterns if $w_{ij} = w_{ji}$ and $w_{ii} = 0$. These choices are included in the learning rule (step 1 in Box 3). Hopfield proved [Hopfield, 1984] that this is also true if graded response nonlinearities are used.

Box 3: The Hopfield Algorithm

Step 1:

Store the M patterns $\underline{x}^s$ ($0 \le s \le M-1$) in the network by assigning the weights to: with $0 \le i,j \le N-1$.

$$w_{ij} = \begin{cases} \sum_{s=0}^{M-1} x_i^s x_j^s & , \forall\, i \neq j \\ 0 & , \forall\, i = j \end{cases} \qquad (12)$$

Step 2:

Apply an input to the network and initialise the output nodes of the network to:

$$y_i(0) = x_i, \quad 0 \le i \le N-1 \qquad (13)$$

Step 3:

Repeat:

$$y_j(t+1) = f\left[\sum_{i=0}^{N-1} (w_{ij} y_i(t) - \theta_j)\right], \quad 0 \le j \le N-1 \qquad (14)$$

until the output of the network is stable.
In this equation f is the hard limiting nonlinearity which equals -1 or +1.

For the Hopfield network an energy function can be defined:

$$E = -\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} w_{ij} y_i y_j + 2\sum_{i=0}^{N-1} \theta_i y_i \qquad (15)$$

This function can be seen as a cost function. It can be proven, that this function always decreases when the state of a processing element changes and therefore will always lead to a lower energy. It is also possible to prove, that changes continue until a minimum energy is reached. This minimum energy state is a stable state of the Hopfield network. Because energy changes have a minimum value, this means that this stable state can be reached in a finite number of iteration steps. The stable state is one of the M stable states of the network. Every *local minimum* coincides with one of the M stored patterns.

To which local minimum the network converges, depends on the initial state of the iteration process that is, the applied input. This input determines to which minimum the network converges and which output will be generated. Very often this output is associated with the nearest local minimum (according to some distance measure; usually hamming distance) but this can not be guaranteed. Methods to improve this are unknown.
In spite of this, the Hopfield network is often used in pattern recognition. If M patterns are stored within the network and distorted, noisy or incomplete patterns are applied to the network, the output will be the stored pattern which resembles the input most, with high probability. This nearest pattern then is the undistorted or completed input pattern. In this way, the Hopfield network is then used to reconstruct patterns so it can be seen as a Content Addressable Memory (CAM). Another application of the Hopfield network is its use in a famous optimizing problem called the Travelling Salesman Problem in which a salesman has to visit N cities once ones while travelling a minimum number of miles [Hopfield & Tank, 1985].

Although applied often, the Hopfield network has of course its limitations. The two most important limitations will be mentioned in this report.

The first limitation of the Hopfield network is its limited capacity. If the network consists of N nodes, there is a maximum number of patterns which can be stored and accurately recalled. If this restriction is not obeyed it can not be guaranteed that the network reaches a stable state which coincides with one of the M local minima. The second limitation of the Hopfield network is, that patterns bear much resemblance can not be memorized correctly. This is due to the fact that their local minima are near. It is very well possible that the wrong minima will be reached if such patterns will be applied to the network. This last limitation can fortunately be overcome by *preprocessing* the inputs. When preprocessing data, one tries e.g. to *orthogonalize* the data. The first limitation can not be overcome. In this way the Hopfield network is not attractive when a large number of patterns has to be memorized. The results described in literature therefore always concern "smaller" problems.

3.2.2 The Hamming network

In the Hopfield network binary inputs have to be associated with one out of M stored patterns. This is done by minimizing some cost function. Problems like this occur very often in Communications Theory. Codes $\underline{x}$ of length N are sent over a noisy memoryless channel and received at the end of that channel as distorted noisy patterns $\underline{x}^*$. A decoder then has to decide which out of M possible codes was sent.

A decoder which is very often used if the a priori probabilities of the presented codes are equal, is the *Maximum Likelihood Decoder* (MLD). In a MLD first the *conditional probability* $P(\underline{x}|\underline{x}_j)$ is calculated for every pattern $\underline{x}_j$ ($0 \le j \le M-1$). P(.) denotes the probability of receiving $\underline{x}$ given that example $\underline{x}_j$ was presented at the input. The maximum likelihood decoder picks out that class j^* for which:

$$P(\underline{x} \mid \underline{x}_{j^*}) \ge P(\underline{x} \mid \underline{x}_j), \ \forall\, j \ne j^* \qquad (16)$$

This class will be assigned to the input pattern.

A neural net which uses this idea is the *Hamming network*. The net consists of one layer of N input nodes x_i. These input nodes are fully connected to a net of internal nodes z_j ($0 \le j \le M-1$). The connection strengths in this so called *lower subnet* are denoted as w_{ij}.
The lower subnet is followed by an *upper subnet*, the MAXNET. This net picks out the internal node with the maximum output. If z_k is the maximum in the internal layer, the MAXNET indicates this by making $y_k = 1$ and all other outputs y_j ($j \ne k$) equal to 0. The MAXNET is a Hopfield like network. It has M inputs z_j and M outputs y_j. Input layer $\underline{z}$ and output layer $\underline{y}$ are fully connected by connections with weight τ_{jk}.

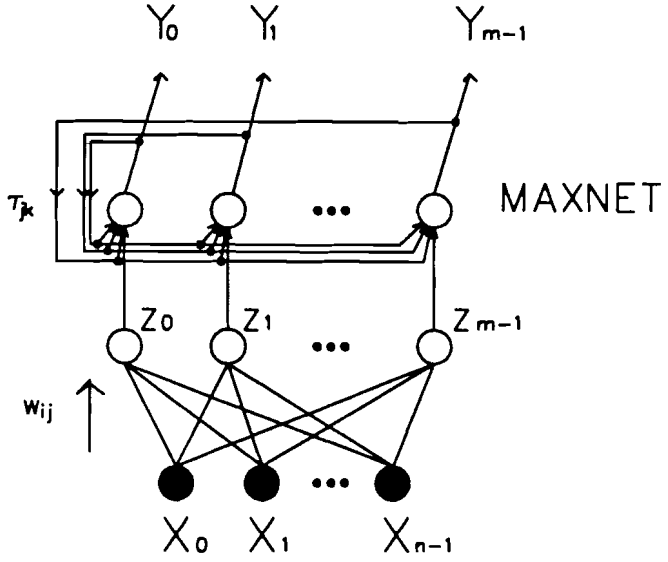

Fig. 3.7: The Hamming network

The algorithm which is used to classify within a Hamming net is described in Box 4. It consists (again) of two parts. In the first part (step 1) connections are set to their initial values. In the second part (step 2 and 3) an input is applied and the net iterates to a stable state. If that stable state is reached, only one out of M output nodes equals 1. The class associated with this winning node is associated with the input pattern.

Box 4: The Hamming network algorithm

Step 1:

Initialise the lower and upper net in the Hamming net:

$$w_{ij} = \frac{x_i^j}{2}, \quad \theta_j = \frac{N}{2}$$

$$\tau_{jk} = \begin{cases} 1, & j = k \\ -\varepsilon, & j \neq k \end{cases} \qquad (17)$$

for $0 \leq i \leq N-1$, $0 \leq j,k \leq M-1$ and $\varepsilon < 1/M$.

Step 2:

Apply an unknown input pattern to the network and calculate the initial internal node values:

$$z_j(0) = f\left(\sum_{i=0}^{N-1} (w_{ij} x_i - \theta_i) \right) \qquad (18)$$

for $0 \leq j \leq M-1$. $f(\_)$ is a hardlimiter nonlinearity.

Step 3:

Repeat:

$$z_j(t+1) = f\left(z_j(t) - \varepsilon \sum_{k \neq j} z_k(t) \right) \qquad (19)$$

for $0 \leq j,k \leq M-1$ until z_j stabilizes.

The initialisation of the Hamming net is the most essential step in this algorithm. By this initialisation, the internal nodes calculate a value which is proportional to N minus the Hamming distance N^j_{ham} between $\underline{x}$ and $\underline{x}^j$. In literature a relation is derived between $N-N^j_{ham}$ and $P(\underline{x} \mid \underline{x}^j)$ which states that [Lippman, 1987b]:

$$P(\underline{x} \mid \underline{x}^j) = \left(\frac{\varepsilon}{1-\varepsilon} \right)^{N^j_{ham}} (1 - \varepsilon)^N \qquad (20)$$

Because ε is chosen to be less then 0.5, $P(\underline{x} \mid \underline{x}^j)$ is maximal if the Hamming distance is minimal, so the maximum conditional probability can be found by selecting that pattern $\underline{x}^j$ with maximum $N-N^j_{ham}$. The MAXNET in Fig. 3.7 takes care of this task. The algorithm sketched here is a possible solution to find the maximum out of M nodes by iteration. If this algorithm is used, it is guaranteed that the MAXNET converges within 10 iterations [Lippmann, 1987b].

Compared to the Hopfield net, the Hamming net has a few major advantages. First, the number of connections is much smaller. Second, the performance is never worse [Lippmann, 1987a][Lippmann, 1987b] and third, a match will always occur. Whether this is an advantage or not is discussable.

When using the Hamming net, which can be seen as a supervised, feedforward Neural network which uses binary input and output patterns, one has to take into account that inputs have to be applied until

the neural network becomes stable. One also has to consider that the Hamming net implements the Maximum Likelihood Classifier which assumes equal a priori probabilities for the applied inputs.
For completeness it will be mentioned that the Hamming net algorithm is governed by only one parameter ε. If $\varepsilon < 1/M$ the network will always converge to one of the M stable states in finite time.

3.2.3 The Boltzmann machine

The *Boltzmann machine* is a neural network that was introduced by Hinton, Ackley and Sejnowski in 1985 [Achley & Hinton, 1985]. It has been developed to overcome the problem that a network gets stuck in a local minima of some cost function instead of reaching the global minimum. The Boltzmann machine is a neural network which is able to do this. When discussing the Boltzmann machine a different notation is used compared to the earlier used notation in this report.

The Boltzmann machine is a 3 layer network. It consists of NIN input nodes, H hidden nodes and NOUT output nodes. All these nodes are denoted as s_i or s_j respectively. Their values can be either 0 (inactive) or +1 (active). From layer to layer, the network is fully interconnected. The connections are denoted as w_{ij} for a connection from node s_i to node s_j. For simplicity the nodes are numbered from 1 to NTOT. NTOT = NIN + H + NOUT. Node s_1 is the first input node. Node s_{NOUT} the last output node.

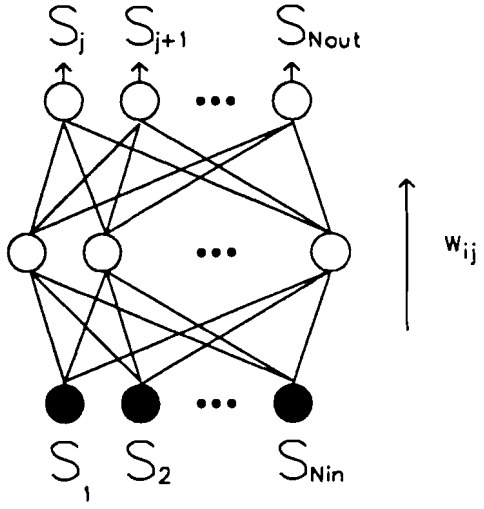

Fig. 3.8: The Boltzmann machine

Just as the Hopfield net, an energy function can be constructed for the Boltzmann machine. This energy function resembles the energy function of the Hopfield net with all $\theta_i = 0$, although there is a difference: the values of the nodes (-1 or +1 in the Hopfield net; 0 or +1 in the Boltzmann machine).

$$E = -\sum_{\forall i}\sum_{\forall j} w_{ij} s_i s_j \qquad (21)$$

The Hopfield network was designed as a network which associates an applied input with one out of M stored patterns. The weights were chosen in such a way that every pattern coincides with a local minimum in the Energy function.
The Boltzmann machine is designed for another task. It is developed for solving optimizing problems. This means that *the* optimal output has to be found if an input is applied. In terms of cost- or energy-functions this means that the global minimum has to be reached instead of one of the local minima.

To reach this, two things have to be realized as can be seen at the next page.

1. The weights have to be chosen such, that the global minimum of the energy function coincides with the optimum solution of some optimizing problem if input $\underline{x}$ is applied.

2. An algorithm has to be developed which is able to reach this global minimum.

The solution to 1 will be treated when the training algorithm is discussed. The solution to 2 is the *Statistical Cooling* or *Simulated Annealing* algorithm.

The Statistical Cooling Algorithm (SCA) can be seen as an extension to the Hopfield training algorithm. In the Hopfield algorithm an input is applied to the network. Then a node is switched and the change in energy which corresponds to this switch is calculated. If the switch causes an Energy decrease, this switch is accepted. If the energy increases, the node is switched back. This process is repeated until a local minimum is reached. Then the network has reached a stable state and the output of this state will be associated with the applied input pattern.

In the SCA almost the same happens: an input/output combination is applied and a hidden node is switched. Again the energy change ΔE which is a result of this switch is calculated. Then a probability is calculated according to:

$$P = \frac{1}{1 + e^{-\frac{\Delta E}{T}}} \tag{22}$$

This probability expresses the probability, that a switch is not accepted if the switch results in the calculated energy change and the system has a *temperature* T. The probability that a switch is accepted is 1-P. A random variable between 0 and 1 is generated to decide whether the node will be switched or not.

The process described above will be repeated. Temperature T, which has been set to a high initial value beforehand, decreases slowly during this process. At the end of the iteration process T equals 0. Then the global minimum will be reached with high probability.

The Simulated Annealing or Statistical Cooling Algorithm has been introduced by Metropolis, Rosenbluth, Rosenbluth, Teller & Teller [Metropolis et al., 1953] as an algorithm which was able to iterate to global minima in cost function. Kirkpatrick, Gelatt and Vecchi applied this algorithm and proved that the global minimum will be reached with high probability [Kirkpatrick et al., 1983]. The algorithm originates from the steelmaking industry in which steel is melted and cooled down very slowly in order to get very strong steel with a minimum energy.

By introducing probability P, it is possible to jump over local *energy hills* in the energy function. In this way one is able to skip local minima. The chosen probability function implements even more than this as can be seen in fig. 3.9:

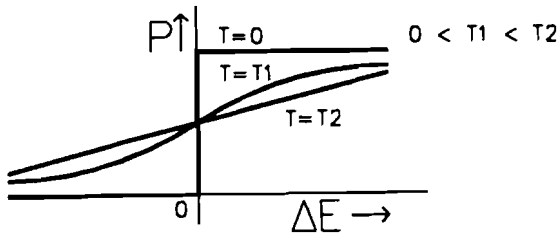

Fig. 3.9: Probability function

In this figure P is sketched as a function of ΔE with T as a parameter. From this figure can be concluded that with decreasing T the chance to switch to a higher energy becomes smaller so the chance to skip to a minimum increases. It also shows that the chance to skip is larger if the energy decreases much.

The SCA is shown in Box 5.

Box 5: The Statistical Cooling Algorithm

Step 1:

Apply an input to the network and set T.

Step 2:

Switch a hidden or output node s_j (j > NIN) and calculate

$$\Delta E_j = -(s_j^{new} - s_j^{old}) \sum_j w_{ij} s_i \qquad (23)$$

Step 3:

Calculate P according to (21), generate a random number between 0 and 1 and decide whether s_j has to be switched or not.

Step 4:

Decrease T according to some predefined decreasing function (linear stepwise; exponentional etc.) and repeat by going to step 2 until T=0.

The training algorithm of the Boltzmann machine can also be summarized very shortly:

Box 6: The Boltzmann machine training algorithm

```
begin
 for k = 1 to M
 begin
   apply input/output combination k
   minimize Energy with Statistical Cooling
   collect statistics P
 end
 for k = 1 to M
 begin
   randomize net
   minimize Energy with Statistical Cooling
   collect statistics P'
 end
 Adjust weights conform statistics
end
```

The algorithm is divided in two parts. In the first part (in literature sometimes called the *positive* or *clamped phase* of the algorithm) an input/output combination is applied to the network. Then the Statistical Cooling Algorithm is used to iterate the network to the global minimum. If this minimum is reached statistics are collected about the state of the network: a matrix P is generated in which p_{ij} increases with 1 if both s_i and s_j are active. After this collecting part a new input is applied and this positive phase is repeated till all the M examples are presented to the network

The second part of the training algorithm is known as the *negative phase* or *free run*. This phase resembles the positive phase. The only difference is that no inputs are applied to the network. The net is initialised randomly and after Statistical Cooling the statistics of the state are collected in matrix P'. This is also repeated M times.

After the free run, the weights are adapted according to equation 24.

$$\Delta w_{ij} = \eta(p_{ij} - p'_{ij}) \tag{24}$$

In this equation $0 \leq \eta \leq 1$.

At this point the learning cycle has been executed once. The training algorithm consist of repeating the learning cycle until P = P'. The network is trained now and can be used for the task it has been developed for.

From all this can be concluded that the Boltzmann machine is a neural net which is trained supervised. The type of learning is a kind of Hebbian learning: weights are adapted proportional to the number of times that its neighbouring nodes both were active. The learning and the converging process is only guided by η, the learning rate and a slowly descending temperature T. With the Boltzmann machine it is possible to reach a global minimum with high probability if T descends slowly during the Statistical Cooling Algorithm. Because T has to start at a high value and has to decrease until 0 this makes the Statistical Cooling Algorithm very slow. This algorithm has to be used both during the training as during the functioning of the Boltzmann machine. As a consequence the Boltzmann machine is very slow. It takes a lot of time to converge if an input is applied. The name *Boltzmann machine* originates from the distribution which is obtained if the quotient is taken from the probability of stable state and initial state:

$$\frac{P_{stab}}{P_{init}} = e^{-\frac{(E_{stab} - E_{init})}{T}} \tag{25}$$

This distribution is known as the *Boltzmann distribution.*

3.3 The Kohonen network and the Counter Propagation network

3.3.1 The Kohonen self-organising feature map

The networks discussed until now are all supervised neural networks. Their "opponents" are unsupervised neural networks. In these nets the desired output is unknown. The networks have to *classify* or *cluster* the input patterns by *self organising* their internal structure. The clustering is such that patterns which have features in common belong to the same classes. As mentioned in chapter 2, competition between output nodes is necessary in these networks.

An example of such an unsupervised neural network is the *Kohonen self organising feature net.* This network was developed from 1979-1982 by T. Kohonen [Kohonen, 1984]. It organises the weights in such a way, that topologically related nodes are sensitive to inputs that are physically similar. This idea is related to the fact, that placement of neurons in the brain is often ordered and reflects some physical relation between those neurons. The Kohonen network tries to map an input space on an output space. An example is the mapping of the position of a feeler mechanism which moves over for example a drawing board, on a 2 dimensional grid. In the beginning of the process there is no relation between both. After the learning process the position of the feeler point must be visible on the grid.

The feeler mechanism can be seen in Fig. 3.10. It moves over a 2 dimensional plane. The coordinates of the feeler point in this plane are denoted as (u,v). It consists of two movable arms. The angles of these arms are denoted as γ (shoulder joint) and ϕ (elbow joint). These two angles are the inputs to a Kohonen network which has two inputs and a two dimensional output grid of M by M nodes. This system has to be trained such, that after training the grid shows the position of the feeler point. In this way the Kohonen network acts as a two dimensional position indicator.

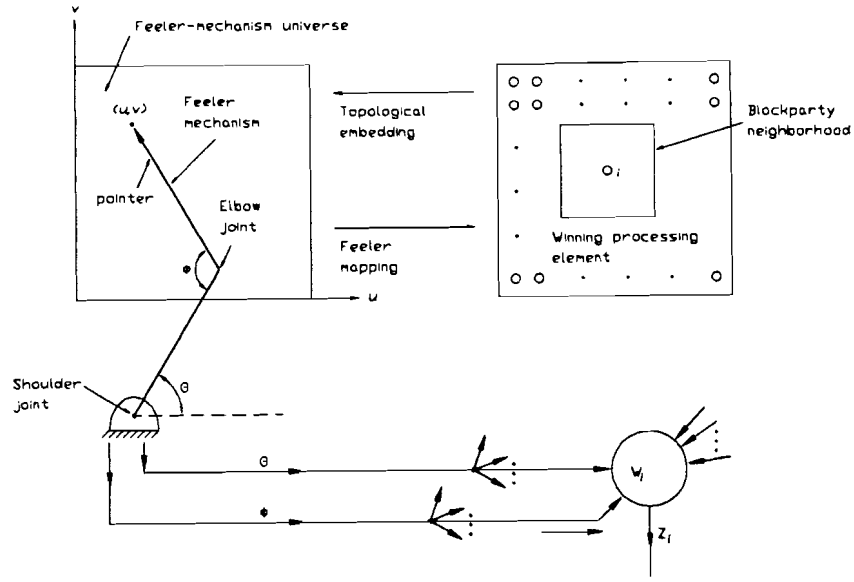

Fig. 3.10: 2 dimensional mapping

The algorithm which is able to do this task, is discussed in Box 7. The neural network which is related to this algorithm can be seen in Fig. 3.11.

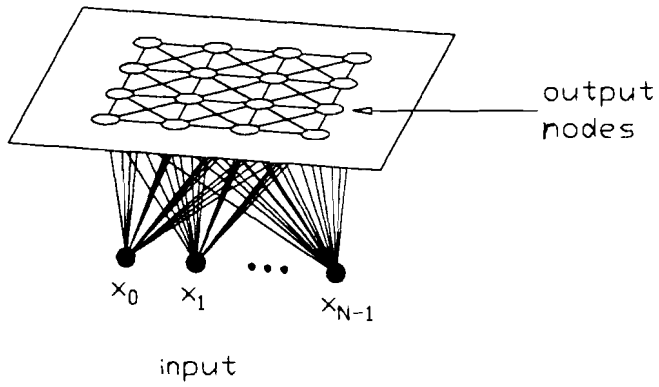

Fig. 3.11: A Kohonen network for 2 dimensional mapping

In this figure the inputs are denoted as x_i. For the feeler mechanism i equals 0 or 1. The output nodes y_j are arranged in a two dimensional grid. Input layer and output layer are fully connected. Weights are denoted as w_{ij}.

The Kohonen training algorithme is as follows. After initialising the weights an input is presented and the distances between input nodes and output nodes are calculated. Then the competition takes place, which selects the output node which is nearest to the input. This node will be associated with the input and in conformity with competitive learning strategies the weights to this node have to be adjusted. Kohonen arranges this adjustment a little different. He introduces a neighbourhood function NE_j, around this winning node and adapts all the nodes which belong to the neighbourhood of the winning node. He states that neurons which are topologically close to each other react more or less to the same inputs. The sequence discussed above is repeated. During this, the learning rate and the neighbourhood function slowly decrease. The latter can be seen in Fig. 3.12. In this way the network will be arranged globally initially. The weights become locally refined to local features. At the end of this process the weights are fixed and a topological map will have been realized.

Box 7: The Kohonen training algorithm

Step 1:

Initialise the weights to small random values and define a neighbourhood function NE_{j*} and set its radius to some initial value.

Step 2:

Present input pattern to the network.

Step 3:

Compute the distance between the input and every output node y_j according to some distance measure e.g.:

$$d_j = \sum_{i=0}^{N-1} (x_i(t) - w_{ij}(t))^2 , \quad \forall j: 1 \le j \le M.M \qquad (26)$$

Step 4:

Select the node y_{j*} with minimum distance d_j and adapt the weights to this node and the nodes which are in the neighbourhood of this winning node using:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t)) \qquad (27)$$

For $j \in NE_{j*}(t)$ and $0 \le i \le N-1$.
$\eta(t)$ is a gain term better known as the learning rate. Its initial value is between 0 and 1 but decreases in time.

Step 5:

Decrease the learning rate and the neighbourhood function $NE_{j*}(t)$ and repeat by going to step 2.

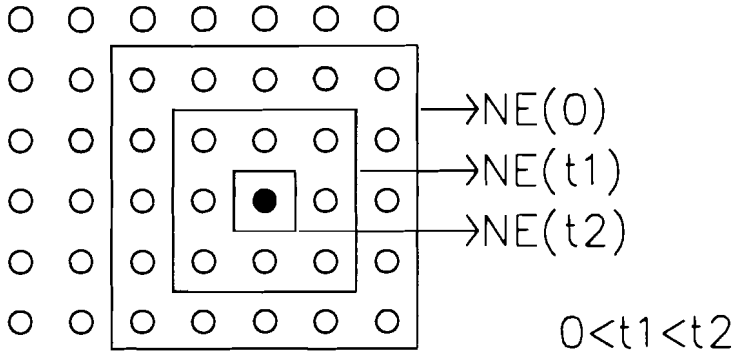

Fig. 3.12: Decreasing neighbourhood function

The architecture discussed in the example is a structure which maps 2 coordinates on a 2 dimensional grid. The Kohonen network can do any mapping from input space to output space. The dimensions of the network have to agree with these space dimensions. This also holds for the neighbourhood function which can be of any shape. If architectures are well chosen, Self Organising feature maps can be nice solutions in solving mapping problems. Unsupervised neural networks have proven to perform relatively well in noisy backgrounds if the amount of training data is large compared to the number of clusters.
The Self Organising feature maps are related to the Linear Vector Quantizer theory. The Kohonen training algorithm which implements this theory in a neural network is governed by just one parameter η the learning rate and one function NE_{j*}, the neighbourhood function.

3.3.2 The Counter Propagation network

The *Counter Propagation network* (CPN) is a network that was invented by Hecht-Nielsen in 1986 [Hecht-Nielsen, 1987], "while seeking a way to use self organising maps to learn explicit functions" [Hecht-Nielsen, 1989b]. The network is a combination of Kohonens self-organising map and the INSTAR/OUTSTAR structure of Grossberg [Grossberg, 1982]. It is developed to approximate continuous functions f: $\underline{y}_k = f(\underline{x}_k)$.

In Fig. 3.13 the "feedforward only CPN" is shown. It consists of three layers: one input layer in which inputs x_i and outputs y_i ($0 \le i \le N-1$) of the relation f(.) are presented to the network. The nodes x_i are fully connected to the second layer of the network by connections w_{ij}. This layer consists of L nodes z_j with $0 \le j \le L-1$ and is known as the *Kohonen layer*. This layer again is fully connected with the output or *Grossberg layer* by connections u_{ji}. The output nodes of the Grossberg layer are denoted as y'_i ($0 \le i \le N-1$). Besides the weights u_{ji} the function-outputs y_i are inputs to the output layer.

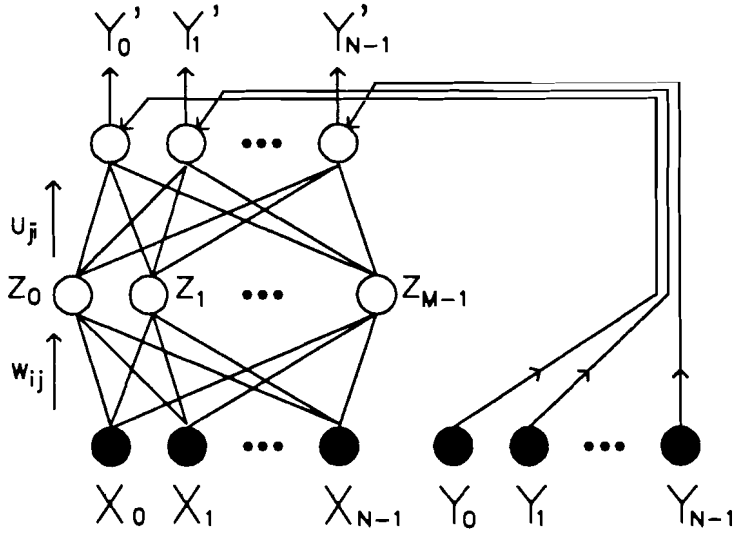

Fig. 3.13: A feedforward only Counter Propagation Network

The training of the CPN can be divided in two parts: training of the Kohonen layer and training of the Grossberg layer. Training of the Kohonen layer has already been discussed in section 3.3.1. After this training the inputs are clustered. Every node in the Kohonen layer represents one cluster. The clustering is such that the weight vectors of the Kohonen network distribute themselves in an almost equiprobable configuration. It is important to point out here, that this clustering is achieved unsupervised. Although outputs y_i are presented to the network, they have not been used till now. This changes when the Grossberg layer is added to the network. In this layer another process takes place. From literature can be concluded that the Grossberg layer stores average patterns [Grossberg, 1982]. In the CPN the input to the Grossberg layer is the output of the Kohonen layer. In this Kohonen layer only one node z_j, equals 1. All others equal 0. This so called winning node represents a class of output patterns $\underline{y}$. The Grossberg layer has to generate the pattern $\underline{y}'$ which is the statistical average of patterns $\underline{y}$ which belong to the class z_j, if this class wins the competition. This is reached by adjusting the weights from the winning node to the output layer using:

$$u_{j,i}(t+1) = u_{j,i}(t) + a(y_j - u_{j,i}(t)) \quad , \quad \forall i: 0 \le i \le N-1 \qquad (28)$$

In this formula a represents the learning rate which has a constant value between 0 and 1.

Training of the Grossberg layer starts simultaniously with the training of the Kohonen layer. Both layers execute two totally different tasks. The Kohonen layer is responsible for the clustering of inputs.

This is the neural network task which is still unsupervised. The parameters which guide this process are the same as mentioned in section 3.3.1. Training of the Grossberg layer can be seen as labelling the output nodes of the Kohonen layer. To every node the average of the patterns associated with that node is assigned.
Both parts together can thus be seen as a *look up table*. The Kohonen layer compares a vector $\underline{x}$ with all the vectors $\underline{w}$ and finds the closest match using some distance measure. The Grossberg layer looks up which pattern $\underline{y}$ has to be associated with the closest match. It can be proven, that if a sufficiently large network is chosen, the mapping approximation can be as accurate as desired. This means that a CPN is able to perform any continuous mapping function and is as effective as the MLP. Compared to the mapping of a MLP fewer calculations are needed if the network sizes are the same. Unfortunately, a CPN has to be large to achieve the same accuracy as a MLP. A trade off has to be made to determine which network is the quickest solution to some problem. In practice, CPN's are very often used as a first step to do rapid prototyping. After that a final prototyping is achieved with a MLP.

3.4 The Adaptive Resonance Theory network

The last neural network which will be discussed in this report is the net which has been developed by G. Carpenter and S. Grossberg: ART. It was designed after the development of the *Adaptive Resonance Theory (ART)*. It is said, that ART networks solve the *stability-plasticity dilemma*: "they are stable enough to preserve significant past learning but nevertheless they remain adaptable enough to incorporate new information". In other words: they are always able to learn new patterns without forgetting the past. ART networks implement a *clustering algorithm* [Hartigan, 1975]. An input is presented to the network and the algorithm checks whether it fits into one of the already stored clusters. If so, the input is added to this cluster. If not, a new cluster is formed.

Carpenter and Grossberg developed different ART architectures as a result of 20 years of very fundamental research in different fields of science. They introduced ART 1 [Carpenter & Grossberg, 1986], a neural network for binary input patterns. They developed and are still developing different ART 2 architectures [Carpenter & Grossberg, 1987a][Carpenter & Grossberg, 1987b] which can be used for both analog and binary input patterns. Recently they introduced ART 3 [Carpenter & Grossberg, 1990], hierarchical ART 2 networks in which they even incorporate chemical (pre)synaptic properties. Carpenter and Grossberg are not only very productive researchers they also publish many papers. Unfortunately their reports are very difficult to understand and only little literature can be found from other authors about this subject. Literature from other authors [Lippman, 1987a], [Maren, 1990] is often much easier to understand. This is the reason why the ART 1 network which will be discussed in this section concerns the "Lippman architecture". At first sight this net does not resemble the Carpenter and Grossberg architecture. In chapter 4 will be shown, that they are almost identical.

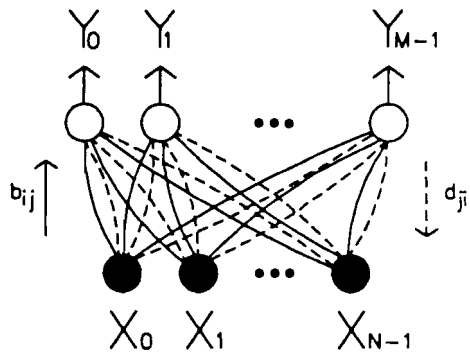

Fig. 3.14: ART 1 network

The ART 1 network used by Lippman can be seen in Fig. 3.14. It consist of an input layer of nodes x_i ($0 \leq i \leq N-1$) and an output layer with nodes y_j ($0 \leq j \leq M-1$). The values of al these nodes can be either 0 or +1. Between the input and output layer one finds two fully connected networks: a *bottom-up* net with connections b_{ij} from input nodes x_i to output nodes y_j and a *top-down* net with connections d_{ji} from output layer to input layer. These nets are also denoted as the bottom-up and top-down filter. A set of connections to or from one output node is called a vector.

The algorithm to train the ART 1 network is shown in Box 8. First the net is initialised by setting the bottom up and top-down weights. Thereafter an input is applied to the network. A bottom-up calculation to every output node takes place by calculating the weighted summation from the input vector to every output node. The output which reacts maximal on the presented input is selected as the winner. This output node is associated with the presented input. To check whether this input indeed belongs to the assigned output class, a *vigilance test* which is guided by ρ, the vigilance parameter, takes place in which the input vector will be compared with the vector from this winning node which is stored in the top-down filter. In this vector a *prototype* for the class assigned by the winning node is stored. This also holds for the bottom-up filter. Both filters are therefore called the *Long Term Memory* (LTM) of the ART network. If the vigilance test results in a fit, this LTM will be adjusted in order to incorporate the presented input in the assigned cluster. If the outcome of the vigilance test results in a non-fit, the present winning node will be switched off. The node which reacts maximal on the presented input now, will be denoted as the new winning node. The above operations are repeated with this winning node. This whole process will be repeated until a fit is found. If there are enough output nodes this is always possible.

Box 8: The ART 1 training algorithm (Lippmann)

Step 1:

Initialise the bottom-up and top-down net:

$$b_{ij} = \frac{1}{1+N}, \quad d_{ji} = 1 \tag{29}$$

for $0 \leq i \leq N-1$ and $0 \leq j \leq M-1$ and set ρ to a value between 0 and 1.

Step 2:

Present a (new) input $\underline{x}$ to the network.

Step 3:

Execute a bottom-up calculation by multiplying the input vector with the bottom-up filter:

$$y_j = \sum_{i=0}^{N-1} b_{ij}(t)x_i, \quad \forall j: 0 \leq j \leq M-1 \tag{30}$$

Step 4:

Select the output y_{j^*} with the highest output and test whether the pattern stored in the top-down filter and is associated with this node fits the input by calculating:

$$\|\underline{x}\| = \sum_{i=0}^{N-1} x_i, \quad \|\underline{d}_{j^*}.\underline{x}\| = \sum_{i=0}^{N-1} d_{j^*i}x_i \tag{31}$$

and testing whether

$$\frac{\|\underline{d}.\underline{x}\|}{\|\underline{x}\|} > \rho \qquad (32)$$

If this is not true then go to step 5 else go to step 6.

Step 5:

The input doesn't belong to the class assigned by y_{j^*}. Disable this class by setting y_{j^*} temporary to zero. Repeat by going to step 4.

Step 6:

The input belongs to the assigned cluster; the weights to and from y_{j^*} have to be adapted using

$$d_{j^*i}(t+1) = d_{j^*i}(t)x_i \;, \quad b_{ij^*}(t+1) = \frac{d_{j^*i}(t+1)x_i}{0.5 + \sum_{i=0}^{N-1} d_{j^*i}(t)x_i} \qquad (33)$$

for all i: $0 \le i \le N-1$.

Step 7:

Repeat by going to step 2.

This training algorithm looks rather complicated but it can be concluded, that the clustering algorithm is implemented. The most important steps in which this is realised are step 3 and 4. In these steps a winner is selected and a calculation performed to test, whether the input pattern belongs to the class associated with the winning node. The first criterion is governed by the bottom-up filter to the winning node; the second criterion by the top-down filter from that node. At first glance both criteria seem to be totally different. If step 6 is taken into account, it can be seen that this is not true. In step 6 can be seen that if y_{j^*} is associated with the input pattern, only the vectors connected with this winning node will be adjusted. A nonzero value at position i in b_{ij} and d_{ji} denotes that a connection is present between input node i and output node j. Equation (33) shows that this connection remains present only if $x_i = 1$. If $x_i = 0$, this connections disappears. This counts for both the bottom-up and the top-down connections. The difference can be found in the values of the connection weights. In the top-down filter these weights always equal 1 if a connection is present. In the bottom-up filter these values are inverse proportional to (0.5 + the number of connections to output node y_{j^*}). From this can be concluded, that information in the bottom-up vector to an output node is exactly the same as the information in the top-down vector from that node. Due to these learning rules the resemblance between step 3 and step 4 becomes clear. In both steps the "overlap" between the input pattern and the filter vectors is calculated. In step 3 the number of connections within a vector is taken into account. In this way it is prevented that nodes which are densely connected with the input layer will always win the competition. In step 4 the number of connections is not taken into account. Because the strength of connections always equals 1 the test criterion becomes clear: it is a Hamming distance comparison between $\underline{x}$ and $\underline{d}.\underline{x}$. The cooperation between these two criteria together with the value of the vigilance parameter governs the clustering. This cooperation makes the ART 1 network difficult to understand but also very interesting, because the network is "critical" to its own clustering. Where competitive learning schemes only calculate winners, ART architectures also take into account whether the winning pattern really matches the input pattern.

3.5 The networks compared

This chapter is a result of an extensive literature research in the field of neural networks. It gives a review of 9 networks studied during that research. Although this review is not complete it gives an

impression of the properties and possibilities of some important neural networks which are known nowadays.

The reason for studying neural networks is a pattern recognition problem: recognizing features in Auditive Evoked Potential (AEP) patterns. Why neural networks can offer a solution to problems of this kind has been discussed in chapter 1. The exact pattern recognition problem will be discussed in chapter 5. In order to make a decision which neural network is a possible tool to solve the problem, it is important to know the *dimensions* of the problem. J. Habraken [Habraken, 1991] was the first one who studied this pattern recognition problem. In this study simplified AEP patterns which consisted of 64 real input values were used. Every pattern contained 5 or 6 minima and maxima. The neural network has to determine the latencies of these extremities. The first feature studied was top V, the fifth maximum in the AEP. For the network dimensions this implies that a network must have at least 64 input nodes and at most 64 output nodes. The input nodes must be able to deal with real inputs. Taking this into account, it is possible to decide which networks are possible candidates to solve this problem.

A Perceptron can only separate between two input classes. Latencies of top V vary in a wide range [Grundy et al., 1982]. A Perceptron can never separate between all these latencies. The MLP is a better solution to this problem because it can do any continuous mapping if certain demands are satisfied. The MLP can handle both binary and real valued input patterns. These properties together with the fact that the MLP is still the most used neural network, make the MLP the first candidate for further research.

The Hopfield network is not very useful in this case. This is due to a number of reasons: its limited capacity, its binary operation and its auto associativity. The first problem is not present in the Hamming net. Unfortunately this net uses also only binary inputs. The BAEP patterns consist of real values so this Hamming net is not able to solve the BAEP problem. A network which has the same problem is the Boltzmann machine. Compared to the MLP trained with the BPA its performance is better [Prager & Fallside, 1987]. Its convergence speed is less, but can be improved [Barna & Kaski, 1989]. Whether the Boltzmann machine will be quicker than for example the MLP or Kohonens Vector Quantizer is doubtful. The binary input restriction makes the Boltzmann machine useless for our pattern recognition problem.

The Kohonen network (especially when it is combined with a Grossberg layer) can do any mapping as accurate as desired if the network is large enough. This so called Counter Propagation Network handles both binary and real valued input patterns. Therefore it is selected as a candidate for further research. Compared to the MLP this CPN has the advantage, that it is a much quicker network to train and develop.

This advantage of simplicity is not present in the last network which will be discussed: ART. This network is very complicated especially if ART 2, the network which handles both binary and analog or real valued input patterns is used. The learning and clustering process of this network is governed by 7 parameters and an exact algorithm which implements this network is not known yet. In spite of these difficulties the ART 2 network is selected as the third and last candidate for further research. This is mainly because of the very interesting fact that the network is self organising and that it is critical to its own clustering.

Summarized, 3 networks have been selected for further research:

- the Multi Layer Perceptron
- the Counter Propagation Network
- the Adaptive Resonance Theory network

Every network will be developed and trained to detect the fifth maximum in an BAEP patterns by simulation. After this, the results will be compared.
The MLP has already been studied. In [Habraken, 1991] the results of this research are described: the 5^{th} maximum can be detected with an accuracy between 80 and 90 percent. Expectations are propound which state that this accuracy can be even increased.
The CPN is studied by M.J. v. Gils. Results have not been published yet but from presentations can be concluded, that the first results are comparable to MLP results.

This report concentrates on the ART networks: ART 1 and ART 2. As opposed to the other two networks, straightforward algorithms do not exist for ART 2. Literature does not give a solution to this problem. Therefore an important part of this report will discuss the simulations and tests to develop such an algorithm. After succeeding in finding this algorithm the ART 2 network and its behaviour are investigated making use of the AEP pattern recognition problem.

4 The Adaptive Resonance Theory (ART 1) network

This chapter handles about ART 1, the binary input ART network. First the architecture and training algorithm which Carpenter and Grossberg introduced will be discussed. Next this interpretation will be compared to the Lippmann interpretation which has been shown in chapter 3 and which looks much more easy to understand. With the comparison will be made acceptable that both interpretations implement the same ideas. Then the Lippmann interpretation will be used in order to simulate ART 1. These simulations will explain a lot about ART networks.

4.1 The Carpenter/Grossberg ART 1 network

The ART 1 network was been introduced by Carpenter & Grossberg in 1987 [Carpenter & Grossberg, 1987a]. It is a neural network which was designed to classify or cluster a set of binary valued input patterns by self organisation according to the clustering algorithm described in chapter 3. If an input is applied to the network, this input will be associated with a class or cluster to which the input possibly belongs. Whether this is justified is tested. If the outcome of this test is true, the input is incorporated in the assigned cluster. If the outcome of the test is not true, search for another cluster will be continued. It is said that ART networks solve the *stability-plasticity dilemma*: "How can a learning system be designed to remain plastic, or adaptable, in response to significant events and yet remain stable in response to irrelevant events?" [Carpenter & Grossberg, 1988].

4.1.1 The network

The ART 1 network, as introduced by Carpenter and Grossberg, can be seen in Fig. 4.1. It consists of 3 parts: *a gain controller, an attentional subsystem* and *an orienting subsystem.*

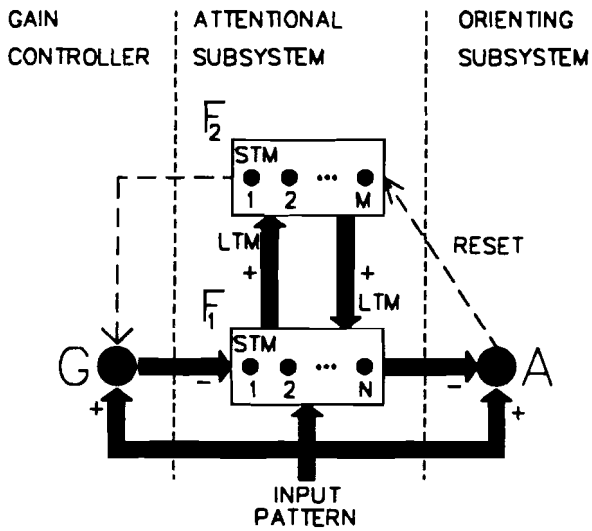

Fig. 4.1: The Carpenter/Grossberg ART 1 network

The gain controller (G) is nothing but a data controller. It controls the attentional subsystem by sending data or not.
The attentional subsystem is the central part of the ART 1 network. It consists of two layers, a bottom layer F1 and a top layer F2. F1 consists of N binary valued nodes. The output $\underline{X}$ of these nodes is controlled by a so called *2/3 rule*, which states that a node i in F1 outputs a signal if it receives at least 2 out of 3 inputs equal to 1. The F2 layer has M binary valued nodes. It is the output layer $\underline{Y}$ of the ART 1 network. Every node in F2 can represent a cluster.
F1 and F2 are also denoted as the *Short Term Memory* (STM) of the ART 1 network. $\underline{X}$ and $\underline{Y}$ are therefore called the *F1 STM pattern* and the *F2 STM pattern.* F1 and F2 are connected by two fully

connected filters: a bottom-up filter BOT with connections BOT_{ij} from F1-nodes X_i to F2 nodes Y_j and a top-down filter TOP with connections TOP_{ji} from F2 to F1. These filters are also indicated as the *Long Term Memory* (LTM). BOT_{ij} and TOP_{ji} have variable weights. Although F1 and F2 each consist of only 1 layer, a few values can be related to these vectors. This will be expressed by associating a few arrays to the two STM layers.
For the F1 layer these associated arrays are **S** and **B**. For the F2 layer they are **T** and **U**. Including these arrays in the network leads to the following representation of the attentional subsystem:

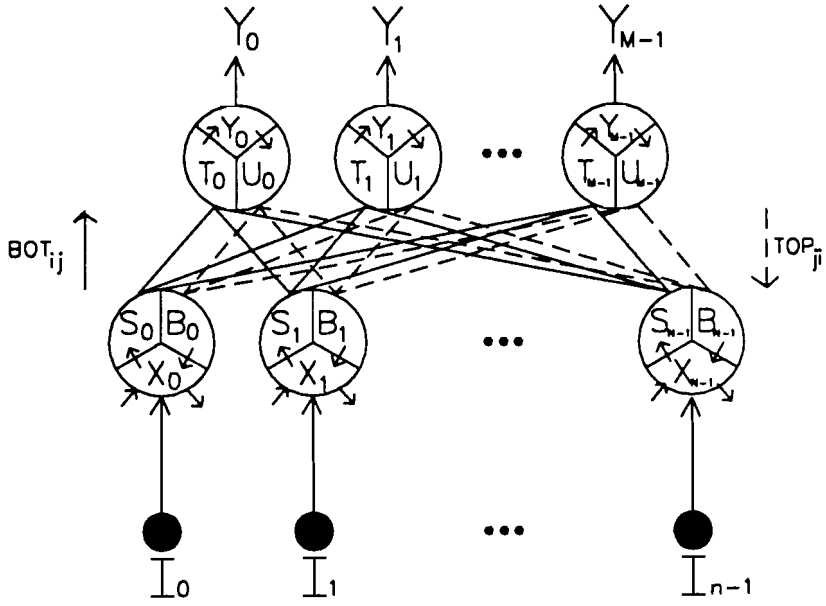

Fig. 4.2: ART 1 attentional subsystem

The third part of the ART 1 network is the orienting subsystem (A). This part is responsible for testing whether the input fits the assigned cluster. The orienting subsystem tests this by comparing **I** and **X**. If these vectors look alike nothing happens. If they differ enough, A generates a RESET. The test which has to check this is guided by a parameter ρ, the *vigilance*.

4.1.2 The ART 1 training algorithm

The training algorithm of a neural network selects the weights which have to be changed in order to reach some association, generalisation etc. How this is reached within the ART 1 network will be discussed here. The different stages in this algorithm are shown in Fig. 4.3.

First the LTM will be initialised by assigning small random values to the BOT_{ij} weights and setting the TOP_{ji} weights equal to 1. At this stage ρ, is set to a value between 0 and 1. Then an input will be applied to both the gain controller, the attentional subsystem and the orienting subsystem. The gain controller immediately feeds this input **I** to F1. The nodes in F1 now receive 2 inputs both equal to **I**. Because of the 2/3 rule this pattern will be copied into **X**, the F1 *output signal*. This signal will be applied to the orienting subsystem A very quickly. Now A also receives two input signals **I**. Because these signals are the same, a RESET at A will be suppressed.
The F1 STM output signal is not only used to suppress a RESET. It will also be passed through the bottom-up filter. Before this happens, it can be transformed to a pattern **S**, the *F1 activation pattern*. This possibility enables the network to do some pattern processing. Very often, **X** is just copied into **S**. After that, a bottom-up calculation takes place in which **S** will be multiplied with the bottom-up filter by a weighted summation. This results in an *initial F2 pattern* **T** at F2:

$$T_j = \sum_{i=0}^{N-1} BOT_{ij} S_i \qquad \forall j,\ 0 \le j \le M-1 \tag{1}$$

As stated before, every F2 node T_j represents a class or cluster. In the vectors to and from these nodes the prototypes of the input patterns of these clusters are stored. This means that eq. (1) is nothing but a series of vector multiplications of a pattern **S** with the prototypes of each clusters. To find the cluster to which the input belongs, the output node which represents this cluster has to be selected. This is reached by letting the values T_j undergo a *competitive* process. During such a process a criterion is defined; the output node which satisfies this criterion most is selected. This node is called the *winning node*. A criterion which is often used in neural networks is selection of the output node with the maximum response to the bottom-up calculation. This winning node T_{j^*} will be stored in **Y** by making $Y_{j^*} = 1$ and all other Y_j nodes = 0. The cluster assigned or represented by this node is assumed to be the best match to the applied input (fig. 4.3 (a)).

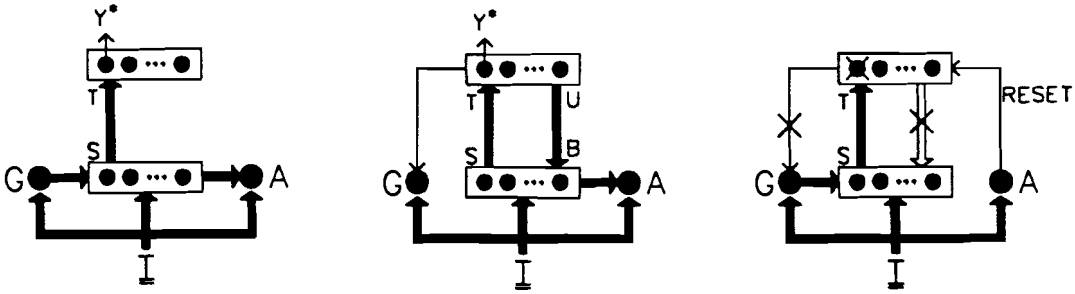

Fig. 4.3: 3 stages of an ART 1 training cycle

To test whether this match is valid, the prototype of this cluster has to be compared with the applied input. This is done by a top-down calculation from the F2 layer to the F1 layer. The F2 STM pattern **Y** eventually can, similar to **X**, be transformed into a *F2 activation pattern* **U**. Often **U** = **Y**. This pattern is sent to the top-down LTM. Simultaneously a signal is sent to the gain controller (G). This signal shuts off the feed through of pattern **I** at G. Now a new pattern enters F1 as a result of the weighted summation of pattern **S** and the top-down LTM. This pattern **B** can be calculated as follows:

$$B_i = \sum_{j=0}^{M-1} TOP_{ji} U_j \qquad \forall i,\ 0 \le i \le N-1 \tag{2}$$

B is called a *top town template* or *top-down learned expectation.* It represents the prototype of the input of the cluster assigned by the winning node. F1 now receives two signals **I** and **B**. Those nodes X_i in F1, which receive two input values elicit an output signal equal to 1. Output vector **X** will be sent to the orienting subsystem A. There, two things can happen now.

1. If **I** and **B** look alike (the way this is checked, will be discussed in section 4.1.3), **X** will resemble **I** very much. Therefore **X** is able to suppress a RESET in the orienting system (fig. 4.3 (b)). The input is said to match the assigned cluster.

2. If **I** and **B** differ much, **X** does not resemble **I**. Now **X** is not able to prevent A from generating a RESET, so this RESET will be generated.

In the first case the match cycle ends. It is known to which cluster the input belongs. The learning phase can start now to incorporate **I** in that cluster. The way this is done will be discussed in section 4.1.4. In the second case the search process or training algorithm has to continue because the input is not accepted as a member of the assigned class. The RESET signal disables the winning node in **Y**. As a consequence of this, the top-down control signal and the F2 activation pattern are removed

(fig. 4.3 (c)). This results in a new feed through of $\underline{I}$ at G, suppression of the RESET signal and a new bottom-up calculation. The outcome of this calculation is the same as after the first bottom-up calculation. When the competitive process starts the preceding winning node is excluded from competition. This results in a new winning node. Now again this winning node is stored into $\underline{Y}$, transformed into $\underline{U}$ and calculated back to $\underline{B}$. At F1 will be tested then, whether the prototype of the cluster assigned by this winning node matches $\underline{I}$. If so, the training algorithm has come to an end. If not, the above described process of bottom-up calculation and top-down matching has to be repeated until a match is found. This will always happen if there still remains at least one so called *uncommitted node*. This is an output node which is not assigned to any cluster yet. If an uncommitted node wins the competition, due to the initialisation of the top-down LTM this node will always be assigned to input pattern $\underline{I}$. Only if there are no uncommitted nodes left it can happen that an input pattern $\underline{I}$ doesn't fit to any cluster. In that case this has to be made clear although it is always possible within the ART network to add new nodes without influencing the behaviour of the net.

4.1.3 The vigilance test

The orienting subsystem has to test whether the F1 STM pattern $\underline{X}$ and input pattern $\underline{I}$ resemble enough in order to prevent a RESET. What "enough" means can be expressed with a parameter ρ, the *vigilance*. The test which compares both vectors is therefore called the *vigilance test.* ρ is defined as follows:

Consider $\underline{I}$ as the input to the ART 1 network, then A receives $\underline{I}$ and computes $\delta|\underline{I}|$. With $\delta|\underline{I}|$ is meant a value proportional to the number of active (positive valued) nodes in $\underline{I}$. Because this value is positive, it is called the *excitatory* input to A. A receives also an *inhibitory* value from F1. This value is proportional to the number of active elements in $\underline{X}$: $\gamma|\underline{X}|$ and is subtracted from the excitatory value. The outcome of this substraction now is very important, because a RESET signal is suppressed if this substraction is negative. This means, that $\underline{X}$ and $\underline{I}$ match if:

$$\delta|\underline{I}| - \gamma|\underline{X}| > 0 \qquad (3)$$

so that no RESET has to be generated if:

$$\frac{|\underline{X}|}{|\underline{I}|} > \frac{\delta}{\gamma} = \rho \qquad (4)$$

In this equation the vigilance parameter is defined. The vigilance test used to check wether a RESET has to be generated is then as follows:

if $|\underline{X}| / |\underline{I}| < \rho$ then RESET
else ADAPT WEIGHTS

During training this test will be executed at least 2 times. The first time is when an input is applied to the network, just before a bottom-up calculation takes place. In 4.1.2 has been concluded, that $\underline{X} = \underline{I}$ then. From eq. (4) can be concluded, that this does not result in a RESET because $|\underline{X}|/|\underline{I}| = 1$.
The second time at which the vigilance test will be executed is, if a top-down calculation has taken place. Patterns $\underline{B}$ and $\underline{I}$ are inputs to F1. Because of the 2/3 rule $\underline{X} = \underline{I} \cap \underline{B}$ which means that:

$$\begin{array}{ll} X_i = 1 & \text{if } I_i = 1 \wedge B_i = 1 \\ X_i = 0 & \text{otherwise} \end{array} \quad \Rightarrow \quad X_i = I_i B_i \qquad (5)$$

Although the notation $\underline{I} \cap \underline{B}$ is matmematically incorrect, it is used in this report because it is the notation which Carpenter and Grossberg used when they introduced the network.

The vigilance test now becomes:

$$\frac{|\underline{I} \cap \underline{B}|}{|\underline{I}|} \leq \rho \qquad (6)$$

With binary inputs this is nothing but comparing the "overlapping" active nodes in $\underline{I}$ and $\underline{B}$ with the number of active nodes in $\underline{I}$. This shows why a so called *uncommitted node* always results in a fit. For an uncommitted node vector $\underline{B}$ equals (1,1,1...1). Therefore $\underline{I} \cap \underline{B} = \underline{I}$ and $|\underline{I}|/|\underline{I}| = 1$; a 100% fit.

4.1.4 The learning rule

If the training algorithm has come to an end, an output node is associated with the input vector. This output node represents a cluster. To stress this relation between input vector and cluster, this vector has to be incorporated into the cluster. The way this is done in ART 1 is a *winner take all* method, which means, that only weights to and from the winning node j^* are allowed to change. Now the name *Adaptive Resonance* becomes clear because weights only change if an association is made or in other words weights will only be *adapted* if input patterns and LTM-prototypes *resonate*.

During learning, bottom-up weights and top-down weights have to be distinguished.
The bottom-up vector in LTM to the winning node becomes:

$$\underline{BOT}_{j^*}^{(new)} = \frac{\underline{I} \cap \underline{BOT}_{j^*}^{(old)}}{\beta + |\underline{I} \cap \underline{BOT}_{j^*}^{(old)}|} \qquad (7)$$

in which β can be seen as a normalisation parameter. Its meaning will be treated in section 4.2.

Adjustment of the top-down vector from the winning node is easier:

$$\underline{TOP}_{j^*}^{(new)} = \underline{I} \cap \underline{TOP}_{j^*}^{(old)} \qquad (8)$$

From these equations becomes clear, that the number of branches can only decrease during learning. The prototypes in LTM become smaller during learning. The type of learning used here is so called *fast learning*, which means that an input is stored into LTM in only a few learning cycles. Learning in this ART network is even very fast. After adjustment of the weights, the input is totally incorporated into LTM. At first sight, the combination of fast learning and "decreasing prototypes" looks rather strange because this will lead to changing prototypes very often. Later on in this chapter will be discussed, how these learning rules can lead to a stable clustering of input vectors.

The opposite of fast learning is *slow learning*. When this type of learning is used, the LTM is not totally adapted to the present input pattern during one learning cycle but only partially. Only if the same input pattern is applied to the network and assigned to the same cluster repeatedly it can be totally stored into LTM. In this way prototypes can be a mixture of the patterns which are stored into the same cluster, but it can take a long time before LTM stabilises.

4.2 The Lippmann ART 1 network and the Carpenter & Grossberg network compared

At this point, two ART 1 architectures have been discussed: the relatively simple Lippmann interpretation in section 3 and the more complex looking Carpenter & Grossberg version in section 4.1. It is said, that both algorithms implement the clustering algorithm. Lippmann states that his implementation is identical to Carpenter and Grossbergs network.

At first sight, this sounds a little unbelievable, because Carpenter and Grossbergs network looks much more complex due to the gain controller, the 2/3 rule etc. which control the dataflow in the network. Lippmanns interpretation of this network does not have this controllers and looks very straight forward. In this section, both interpretations will be compared and it will be shown, that they are identical. To make this comparison less difficult, the Carpenter and Grossberg ART 1 algorithm as discussed in section 4.1 is summarized in Box 9.

Box 9: The ART 1 training algorithm (Carpenter & Grossberg)

Step 1:

Initialise the bottom-up and top-down net according to:

$$BOT_{ij} = small\ initial\ values, \qquad TOP_{ji} = 1 \tag{9}$$

for $0 \le i \le N-1$ and $0 \le j \le M-1$ and set ρ between 0 and 1.

Step 2:

a. Present a (new) input $\underline{I}$ to the network.
Because of the 2/3 rule: $\underline{X} = \underline{I}$ and a RESET is suppressed immediately.
b. Copy $\underline{X}$ into $\underline{S}$.

Step 3:

Execute a bottom-up calculation by multiplying $\underline{S}$ with the bottom-up filter according to:

$$T_j = \sum_{i=0}^{N-1} BOT_{ij} S_i, \qquad \forall j:\ 0 \le j \le M-1 \tag{10}$$

Step 4:

a. Select the output T_{j^*} with the highest output value. Store this output in $\underline{Y}$ by making $Y_{j^*} = 1$ and $Y_j\ (j \neq j^*) = 0$.
b. Copy $\underline{Y}$ into $\underline{U}$ and execute the top-down calculation:

$$B_i = \sum_{j=0}^{M-1} TOP_{ji} U_j, \qquad \forall i:\ 0 \le i \le N-1 \tag{11}$$

c. According to the 2/3 rule: $\underline{X}$ becomes $\underline{I} \cap \underline{B}$.
d. Calculate whether a RESET will be suppressed or not according to:

$$\frac{|\underline{X}|}{|\underline{I}|} > \rho \tag{12}$$

If this is not true, a RESET can not be suppressed so go to step 5.
Else a RESET can be suppressed so go to step 6.

Step 5:

The input does not belong to the class assigned by Y_{j^*}. Disable this class by setting Y_{j^*} to 0. Repeat by going to step 4.

Step 6:

The input belongs to the assigned class. The weights to and from the winning node have to be adjusted in order to incorporate the input in the assigned class:

$$\underline{TOP}_{j\bullet}^{(new)} = \underline{I} \cap \underline{TOP}_{j\bullet}^{(old)}$$

$$\underline{BOT}_{j\bullet}^{(new)} = \frac{\underline{I} \cap \underline{BOT}_{j\bullet}^{(old)}}{\beta + |\underline{I} \cap \underline{BOT}_{j\bullet}^{(old)}|} \qquad \textbf{(13)}$$

Step 7:

Repeat by going to step 2.

The (set of) actions which take place in each step in this Box are comparable to the actions in the same steps in Box 8. In this way both algorithms can be compared step by step.
Step 1 describes the initialisation of the network. In both algorithms the top-down connections get values equal to 1, the bottom-up connections become small initial values and the vigilance parameter is set to a value between 0 and 1. Step 1 in Box 8 and 9 are therefore identical.
In step 2 of the Lippmann algorithm, an input $\underline{x}$ is presented to the network. Nothing else happens. This is also true in Box 9 if a RESET is suppressed. This is always true, due to the 2/3 rule. Therefore $\underline{I} = \underline{X} = \underline{S}$ which means that also in this interpretation only an input is presented to the network in step 2.
Knowing this, it is clear that for both interpretations step 3 is nothing but a multiplication of this input and the bottom-up filter. As will be shown in step 6, the weights of this filter are the same for both interpretations and therefore also this step is the same for both interpretations.
In step 4 both algorithms look rather different. It is easy to show, that this is not true. Eq. (11) (Box 9) shows the top-down calculation. Due to $\underline{U} = \underline{Y}$ and the fact that only node $Y_{j\bullet}$ has a nonzero value ($Y_{j\bullet} = 1$), B_i equals $TOP_{j\bullet i}$ after applying eq. (11). With eq. (5) it is now possible to rewrite step 4c in which $\underline{X}$ has to be calculated.

$$|\underline{X}| = |\underline{I} \cdot \underline{B}| = \sum_{i=0}^{N-1} I_i \cdot B_i = \sum_{i=0}^{N-1} TOP_{j\bullet i} \cdot I_i \qquad \textbf{(14)}$$

This equation equals the second part of eq. (30) (Box 8). Because $\underline{I}$ and $\underline{X}$ both denote the applied input $|\underline{I}| = |\underline{X}|$, it can be concluded, that both vigilance tests are exactly the same. This means that step 4 in Box 8 is the same as step 4 in Box 9. As a consequence of this, this also counts for step 5. This leaves only step 6, which describes the weight adjustment to and from the winning node to be discussed.
Because of eq. (5) (section 4.1), eq. (13) in Box 9 can be rewritten as follows:

$$TOP_{j\bullet i}^{(new)} = TOP_{j\bullet i}^{(old)} \cdot X_i$$

$$BOT_{ij\bullet}^{(new)} = \frac{BOT_{ij\bullet}^{(old)} \cdot X_i}{\beta + \sum_{i=0}^{N-1} BOT_{ij\bullet}^{(old)} \cdot X_i} \qquad \textbf{(15)}$$

From this equation can be concluded, that it resembles eq. (33) (Box 8) partially: the top-down weight adjustments are the same in both equations while the bottom-up weight adjustments seem to differ. It is not difficult to prove that these adjustments also resemble. To prove this, it is important to understand the top-down weight adjustment.
From eq. (33) it becomes clear, that the top-down weights $d_{j\bullet i}(t+1)$ can only remain nonzero if $d_{j\bullet i}(t)$ and x_i are both nonzero. Due to the initialisation and the top-down learning rule, nonzero always yields 1 here. This means that the vectors stored into the top-down filter are also binary valued vectors. The learning rule also yields that only top-down weights connected to the winning node change. The vector from this node can thus be seen as the prototype of the class presented by that winning node. What goes for the top-down filter also goes partially for the bottom-up filter. According to the second equation in eq. (33), a bottom-up weight b_{ij} can only remain nonzero if the coinciding top-down weight d_{ji} is nonzero. The difference between these coinciding weights is, that the top-down weight

is equal to 1. The value of the bottom-up weight can be calculated according to the second equation in eq. (33) which reduces to

$$b_{ij^*} = \frac{1}{0.5 + Z_1} \tag{16}$$

because of the above mentioned properties. In this equation Z_1 denotes the number of coinciding 1's in the top-down vector from the winning node $\underline{d}_{j^*}$ and input $\underline{x}$.

From this, 2 important conclusions can be drawn:

1. All nonzero values in a bottom-up vector to the same output node are equal.
2. Because these nonzero values coincide with the 1s in the top-down vector, the bottom-up vector to a winning node stores the same vector as the top-down vector from this winning node. They only have different magnitudes.

These two conclusions are very important if the resemblance between eq. (13) (Box 9) and eq. (33) (Box 8) has to be explained.

As stated before the top-down weight adjustment is the same in both equation sets. This means that in the Carpenter/Grossberg ART 1 network top-down weights from the winning node also remain one, only if the input node connected to that weight is active ($X_i = 1$). From eq. (13) (Box 9) can also be concluded, that a bottom-up weight b_{ij} can only remain nonzero if the coinciding top-down weight $d_{ji} = 1$. With induction it can be proven, that in this interpretation also all nonzero bottom-up weights to the same output node are equal. This means that BOT_{ij^*} can be simplified to BOT_{j^*}, for $j = j^*$ and $0 \leq i \leq N-1$. All together this means that only the bottom-up weights between active input nodes and the winning output node change. Their new value becomes:

$$BOT_{j^*}^{(new)} = \frac{BOT_{j^*}^{(old)}}{\beta + \sum_{i=0}^{N-1} BOT_{j^*}^{(old)} \cdot X_i} = \frac{BOT_{j^*}^{(old)}}{\beta + BOT_{j^*}^{(old)} \cdot Z_2} \tag{17}$$

In this equation Z_2 denotes the number of coinciding nonzeros in the input pattern and the bottom-up vector to the winning node. As stated before the vectors to a winning node store the same patterns as vectors from a winning node, so $Z_2 = Z_1$. Knowing this, eq. (17) can be rewritten one more time which results in:

$$BOT_{j^*}^{(new)} = \frac{BOT_{j^*}^{(old)}}{\beta + BOT_{j^*}^{(old)} \cdot Z_1} = \frac{1}{(\beta / BOT_{j^*}^{(old)}) + Z_1} \tag{18}$$

Lippmann now chose $\beta/BOT_{j^*}^{(old)} = 0.5$. By doing this, he excludes past learning by which eq. (18) indeed reduces to eq. (16) from which can be expected that the Carpenter and Grossberg interpretation indeed includes Lippmann's interpretation. Question is, whether Lippmann's choice is valid. By choosing $\beta/BOT_{j^*}^{(old)} = 0.5$ Lippmann implies, that $\beta/BOT_{j^*}^{(old)}$ is constant. This can be true, if β is chosen equal to $0.5.BOT_{j^*}^{(old)}$ but in that case β has to be adapted after every learning trial. This is highly unlikely, because BOT_{j^*} changes after every learning trial according to eq.(18). Lippmann's choice also implies, that the nonzero bottom-up weights to different output nodes are the same. This is not true. It looks therefore, that Lippmanns choice is not valid; it is a simplification which is too strong. On the other hand it can be proven, that Lippmann's choice results in a clustering which satisfies the same criteria. This is due to the fact, that the bottom-up weights are only responsible for the order of search. If the way in which they will be adjusted changes, this will perhaps change the order in which output nodes win the competitive process. In ART networks these output nodes have

to undergo the vigilance test in order to test whether their prototype fits the input. This vigilance test is guided by the top-down weights. The top-down weight adjustment has not changed so the clustering will be guided by exactly the same vigilance test. This means, that the clustering in the Lippmann interpretation can be different from the Carpenter and Grossberg interpretation but, that it satisfies the same criteria. At this point we thus have two interpretations which cluster a dataset according to the same strategy and criteria and therefore, they can considered to be similar.

4.3 Simulating the ART 1 network

Now it has been proven that the algorithms described in Box 8 and Box 9 implement the same ideas and cluster a dataset according to the same criteria, it is time to start simulations of the ART 1 network. With these simulations it is possible to get insight in the clustering process of ART 1 networks. It is also a means to get costumed to the nomenclature which accompanies ART literature.

To simulate the ART 1 network, a program has been written in the C programming language. This program is a straightforward implementation of the algorithm described in Box 8 in section 4.2. The program can be used to study every ART 1 clustering process.
The first simulation that is done is a reproduction of a simulation which Carpenter and Grossberg describe in literature [Carpenter & Grossberg, 1986]. In this simulation they use a dataset which consists of 4 input patterns which are shown and visualised in Fig. 4.4.

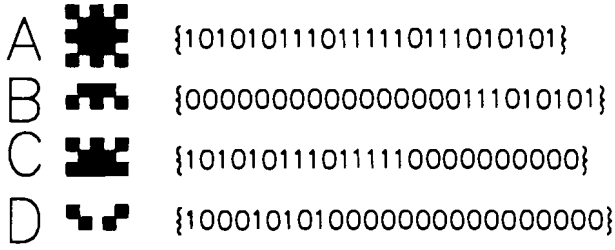

Fig. 4.4: Visualisation of the dataset

These patterns are presented to the network in order: A-B-C-A-D. This is repeated 2 times. With some trial and error a vigilance parameter was found for which the final clustering of this dataset equals the clustering which Carpenter and Grossberg describe. Both clustering processes are shown in table 4.1, in which (a) is the simulated Lippmann interpretation and (b) is the simulation by Carpenter and Grossberg.The fact that they of course used their own interpretation, makes it interesting to compare both clustering processes.

		search process through prototypes			
ex.	input	1	2	3	4
1	A	A_1			
2	B	B_1			
3	C	B	C_1		
4	A	B_2	C_1	A_3	
5	D	B	D_1	A	
6	A	B	D	A_1	
7	B	B_1	D	A	
8	C	B	D_1	C_2	
9	A	B_2	D_3	C_1	A_4
10	D	B	D_1	C	A
11	A	B	D	C	A_1
12	B	B_1	D	C	A
13	C	B	D	C_1	A
14	A	B	D	C	A_1
15	D	B	D_1	C	A
		stable coding			

		search process through prototypes			
ex.	input	1	2	3	4
1	A	A_1			
2	B	B_1			
3	C	B	C_1		
4	A	B_2	C_1	A_3	
5	D	B	D_1	A	
6	A	B	D	A_1	
7	B	B_1	D	A	
8	C	B	D	C_1	
9	A	B_2	D_3	C_1	A_4
10	D	B	D_1	C	A
11	A	B	D	C	A_1
12	B	B_1	D	C	A
13	C	B	D	C_1	A
14	A	B	D	C	A_1
15	D	B	D_1	C	A
		stable coding			

Table 4.1: ART 1 clustering process: (a) Lippmann and (b) Carpenter and Grossberg

The clustering processes in Table 4.1 show 15 so called *match cycles*. Every row in this table shows the clustering in one match cycle, in which an input is applied to the network and the algorithm is repeated until a fit is found and the weights have been adapted. In every row, the prototypes which are stored in the vectors to output nodes 1 to 4 after learning, are shown. The indices below these prototypes show the order in which the output nodes have been searched. The node which is pointed at at last, is associated with the input.

Let us take a look at the clustering process in Table 4.1 (a). It shows, that the first pattern which is applied to the network is associated directly with output node 1 and, due to fast learning, becomes the prototype of this node. Then pattern B is applied which *resonates* with output node 1. After this match cycle B becomes the prototype for this output node. The next pattern which is presented to the network is pattern C. This pattern resonates with uncommitted output node 2 and therefore becomes the prototype. In match cycle 4 pattern A is applied again. First it is assigned to output node 1 and 2. Pattern A does not resemble the prototypes of these output nodes enough so both assignments are RESET. Then output node 3 is assigned. Because this node is uncommitted, this assignment is accepted and A is stored as the prototype. Next pattern D is applied and output node 2 reacts maximal. The prototype of this node fits pattern D so D will be associated to this output node. D even becomes the prototype.
At this point, the set A-B-C-A-D has been applied one time to the network. This will be repeated two more times.

From Table 4.1 (a) and (b) a few properties attract attention:

1. If patterns are recoded, this will always lead to smaller patterns in ART's top-down filter. With smaller patterns is meant, that the number of ones in the top-down filter decreases after learning. Due to this, it can happen that patterns which are assigned to an output node do not fit anymore the next time they are applied to the network, because the prototype of that particular output node has changed. This happens for example with pattern A in match cycle 4. In this cycle, it looks like pattern A has been forgotten.

2. During some match cycles a search is started. Sometimes this search ends at an uncommitted node; other times at an earlier committed node. In first instance, this looks rather strange because this means that a node which does not react maximal on the input fits the input while the node which reacts maximal on the input does not fit.

3. In match cycle 9 type clustering becomes *stable*, which means that the prototypes of the output nodes do not change anymore. In match cycle 10 the output nodes become *direct accessable*. From now on the output node with the applied input as its prototype wins the competitive process and is immediately associated with the applied input.

4. If the clustering in 4.1 (a) is compared to the clustering in 4.1 (b) it can be concluded that the final clustering is exactly the same in both simulations although the clustering process develops slightly different (match cycle 8).

These properties give rise to a number of questions and conclusions.

4.3.1 Decreasing prototypes and stable clustering

The first question is, whether the forgetting due to decreasing prototypes, discussed in property 1, is not in disagreement with the statement that ART networks offer a solution to the stability-plasticity dilemma in which Carpenter and Grossberg state, that their network is plastic enough to incorporate new inputs but remain stable enough to remember past learning. If this is indeed implemented, this is a nice property because the network does not forget then. Unfortunately , the recoding of pattern

A in match cycle 4 and 9 looks like forgetting which is in contradiction with the statement that ART 1 implements a solution to the dilemma!

A second question which arises while looking at this property of ART 1, is whether stable clustering will always be reached. To study this, a second experiment has been done. In this experiment a set of 5 patterns is generated: A = 111111, B = 011111, C = 001111, D = 000111 and E = 000011. During the first part of the experiment, the patterns are applied to the network in order A-B-C-D-E repeatedly. The vigilance parameter is chosen such that all patterns are assigned to the same output node the first time they are applied. As can be seen in Table 4.2 (a), this leads to decreasing LTM prototypes and to recoding if the patterns are applied a second, third etc. time. After 5 times a stable clustering arises in which every output node is direct accesable which shows, that stable clustering can be reached even if the order in which patterns are applied is chosen as unfavourable as possible. Of course this is no proof, but it shows the possibility. The question now is, whether the order of input patterns does not influence the clustering at all.

		search process through prototypes				
ex.	input	1	2	3	4	5
1	A	A_1				
2	B	B_1				
3	C	C_1				
4	D	D_1				
5	E	E_1				
6	A	E	A_1			
7	B	E_1	B_2			
8	C	E_1	C_2			
9	D	E_1	D_2			
10	E	E_1				
11	A	E	D_1	A_3		
12	B	E_2	D_1	B_3		
13	C	E_2	D_1	C_3		
14	D	E	D_1	C		
15	E	E_1	D	C		
16	A	E	D_2	C_1	A_3	
17	B	E_3	D_2	C_1	B_4	
18	C	E	D	C_1	B	
19	D	E	D_1	C	B	
20	E	E_1	D	C	B	
21	A	E	D_3	C_2	B_1	A_4
22	B	E	D	C	B_1	A
23	C	E	D	C_1	B	A
24	D	E	D_1	C	B	A
25	E	E_1	D	C	B	A
		stable coding				

		search process through prototypes				
ex.	input	1	2	3	4	5
1	A	A_1				
2	D	D_1				
3	E	E_1				
4	C	E_1	C_2			
5	B	E_2	C_1	B_3		
6	B	E	C	B_1		
7	A	E	C_2	B_1	A_3	
8	D	E_1	D_2	B	A	
9	C	E_2	D_1	C_3	A	
10	E	E_1	D	C	A	
11	D	E	D_1	C	A	
12	B	E_3	D_2	C_1	B_4	
13	E	E_1	D	C	B	
14	A	E	D_3	C_2	B_1	A_4
15	C	E	D	C_1	B	A
16	B	E	D	C	B_1	A
17	E	E_1	D	C	B	A
18	D	E	D_1	C	B	A
19	A	E	D	C	B	A_1
20	C	E	D	C_1	B	A
		stable coding				

		search process through prototypes				
ex.	input	1	2	3	4	5
1	E					
2	D	E_1	D_2			
3	C	E_2	D_1	C_3		
4	B	E_3	D_2	C_1	B_4	
5	A	E	D_3	C_2	B_1	A_4
6	E	E_1	D	C	B	A
7	D	E	D_1	C	B	A
8	C	E	D	C_1	B	A
9	B	E	D	C	B_1	A
10	A	E	D	C	B	A_1
		stable coding				

Table 4.2: ART 1 stability test: (a) repeated, (b) random and (c) ideal order

To examine that, two subsequent simulations have been done. During these simulations the dataset A-B-C-D-E has been applied to the network again but in the first simulation the order in which the patterns are applied is randomly chosen (Table 4.2 (b)). During the second simulation the order becomes E-D-C-B-A (Table 4.2 (c)). In both simulations the dataset is applied repeatedly.
If the 3 simulations are compared, the influence of pattern order becomes clear: during all simulations stable clustering is reached; how quick this is reached depends of the order in which the patterns are applied.

4.3.2 First fit versus best fit first model

Another question which arises while looking at the properties of the ART network, is the question why ART networks assign nodes which do not resemble the input, before nodes which do resemble the

input. This is due to the fact that ART 1 networks do not search for the best fit but for the first fit. To understand why this is the case a few things need to be explained.

While searching for a fit, ART networks assign output nodes to input patterns by choosing winning nodes after a bottom-up calculation. To check whether that winning node fits the input, a vigilance test governed by a top-down calculation takes place. The result of that top-down calculation is a kind of distance measure. Question now is, why ART networks do not calculate this distance immediately for every combination of input pattern and output node and then select the best match. This idea is easy to implement and looks much less complicated then the combination of bottom-up and top-down calculations as can be concluded from Box 10. This Box describes an algorithm which searches for the best match. In this algorithm $\underline{x}$ denotes the input, $\underline{d}$ denotes the LTM of the network; because its resemblance with ART's top-down filter, it will also be called a top-down vector here. Res is the *resonance factor*, which can be seen as a distance measure, N denotes the number of input nodes, MAXOUT the maximum number of output nodes and M the number of committed output nodes.

Box 10: Search for best fit first algorithm

Step 1:

Initialise the network:

$$d_{ji} = 1$$

for $0 \le i \le N$ *and* j: $0 \le j \le MAXOUT$.
Set M *to* 0 *and* ρ *to a value between* 0 *and* 1.

Step 2:

Apply input $\underline{x}$ *to the network.*

Step 3:

Calculate

$$res_j = \frac{|\underline{d}_j \cdot \underline{x}|}{|\underline{x}|} \quad \forall j,\ 0 \le j \le M \tag{19}$$

Step 4:

Select the best match: $res_{j^*} = max\{res_j\}$

Step 5:

If ($res_{j^*} \le \rho$)
then { $j^* = M + 1$; $M = M + 1$}

adapt weights according to:

$$d_{j^*i}^{(new)} = d_{j^*i}^{(old)} \cdot x_i \quad \forall i,\ 0 \le i \le N-1 \tag{20}$$

Step 6:

Repeat by going to step 2.

It is not difficult to understand, that this algorithm indeed selects the output node whose prototype fits the input best first. If this prototype satisfies the vigilance test, the input is assigned to that prototype and incorporated into its top-down filter. If the test is not satisfied, none of the prototypes fits the input. The input is then assigned to an uncommitted output node. This algorithm has two advantages. First, bottom-up calculations are not necessary and second, the network does never have to start a search, which makes the network much quicker.

In spite of these advantages this algorithm is not used in the ART 1 network. The reason for this is, that even with a simple example it is possible to show that it can happen that a stable clustering will

never be reached. To show this, a dataset is chosen that consists of 3 patterns: 111, 010 and 110. ρ is set to 0.65 and M is initially set to 0. The 3 patterns are repeatedly applied in order 111, 010, 110. It can easily be verified, that after 4 match cycles 2 output nodes are committed to inputs. Then M = 2; $\underline{\mathbf{d}}_1$ = 010 and is associated with pattern 010. $\underline{\mathbf{d}}_2$ = 110 and is associated with 110 and 111. In the 5th match cycle now 010 is applied to the network. Because M = 2 now res_1 and res_2 have to be calculated in step 2. This results in $res_1 = res_2 = 1$. The algorithm now has to choose the maximum out of these two values. Because they are equal, the clustering can go wrong now. If output node 1 is selected nothing is wrong, because node 1 indeed represents 010. If output node 2 is selected something strange happens. Because $res_2 = 1$ this choice results in a fit and therefore $\underline{\mathbf{d}}_2$ becomes 010 too. At this point the two committed nodes store exactly the same prototypes. This whole process can be continued if the input patterns will be applied to the network in the same order. After $\underline{\mathbf{x}}$ = 110, a new node will be committed so $\underline{\mathbf{d}}_3$ = 110 and M = 3. Then $\underline{\mathbf{x}}$ = 111, which will be assigned to output node 3. The learning rule yields the top-down vector of node 3 in this case remains unchanged so $\underline{\mathbf{d}}_1$ = 010, $\underline{\mathbf{d}}_2$ = 010 and $\underline{\mathbf{d}}_3$ = 110. Now $\underline{\mathbf{x}}$ = 010 so $res_1 = res_2 = res_3 = 1$ which results in the same problem if node 3 is selected. If this happens every time when 010 is applied (which is not impossible to occur), the number of output nodes explodes while every output node stores the same prototype. In this way a stable clustering will never be reached so this algorithm, which searches for the best match, is useless.

The reason, that the clustering process goes wrong, is not very difficult to explain. It is due to the fact that the resonance factor for $\underline{\mathbf{d}}_1$ = 010 and $\underline{\mathbf{x}}$ = 010 is equal to the resonance factor of $\underline{\mathbf{d}}_2$ = 110 and $\underline{\mathbf{x}}$ = 010. Because of this, two output nodes can be associated with one input; if 110 will be the one, clustering can go wrong. That this can happen, is not just the case with this example; this problem can be generalized.

Proof 1:

Consider two prototypes $\underline{\mathbf{d}}_1$ and $\underline{\mathbf{d}}_2$, in which $\underline{\mathbf{d}}_2$ is said to be a *subset* of $\underline{\mathbf{d}}_1$ which also means that $\underline{\mathbf{d}}_1$ is a *superset* of $\underline{\mathbf{d}}_2$, then this means that:

$$\begin{aligned} &\text{if } d_{2i} = 1 \text{ then } d_{1i} = 1 \\ &\text{if } d_{2i} = 0 \text{ then } d_{1i} = 0 \vee d_{1i} = 1 \end{aligned} \tag{21}$$

With $\underline{\mathbf{x}} = \underline{\mathbf{d}}_2$ it is easy to prove that $res_1 = res_2 = 1$:

$$res_1 = \frac{|\underline{d}_1 \cap \underline{x}|}{|\underline{x}|} = \frac{|\underline{d}_1 \cap \underline{d}_2|}{|\underline{d}_2|} = \frac{D_2}{D_2} = 1 \tag{22}$$

$$res_2 = \frac{|\underline{d}_2 \cap \underline{x}|}{|\underline{x}|} = \frac{|\underline{d}_2 \cap \underline{d}_2|}{|\underline{d}_2|} = \frac{D_2}{D_2} = 1 \tag{23}$$

in which D_2 denotes the number of ones in prototype $\underline{\mathbf{d}}_2$.

This means that:

if an input $\underline{\mathbf{x}}$ is applied to the network, every output node which is a superset of $\underline{\mathbf{x}}$ will match $\underline{\mathbf{x}}$ exactly according to the distance measure of eq. (19). With "exactly" is meant $res_j = 1$ here. It does not mean $\underline{\mathbf{d}}_j = \underline{\mathbf{x}}$!

This proves, that clustering can go wrong for every dataset which includes patterns and their supersets. Because this will very often be the case, the best fit algorithm is not useful to cluster patterns within an ART 1 network.

❑

This problem can be overcome, if the network is able to distinguish between a pattern and its superset. For the example used to clarify this problem, this yields that the network has to be able to distinguish between the prototypes 010 and 110 if $\underline{x}$ = 010 is applied to the network. It is possible to extend the best fit algorithm of Box 10 to a new algorithm which is able to do this. This algorithm is summarized in Box 11. In this Box $\underline{x}$, d_{ji} and b_{ij} have exactly the same meaning as in Box 8. Therefore $\underline{b}_{j*}$ can be seen as a bottom-up vector. N, MAXOUT, M and res have the same meaning as in Box 10, while out is a new distance measure.

Box 11: The extended best fit first algorithm

Step 1:

Initialise the network:

$$d_{ji} = 1 \text{ and } b_{ij} = 1/(1 + N)$$

for $0 \le i \le N-1$ *and* j: $0 \le j \le MAXOUT$.
Set $M = 0$ *and* ρ *to a value between 0 and 1.*

Step 2:

Apply an input $\underline{x}$ *to the network.*

Step 3:

Calculate res_j *for every* j: $0 \le j \le M$ *according to eq. (19) and select* $res = max\{res_j\}$.

Step 4:

Execute a calculation:

$$out_j = \sum_{i=0}^{N-1} b_{ij} x_i \quad \forall j: res_j = res \tag{24}$$

select the best match by setting j^* *equal to that* j *for which* out_j *is the maximum.*

Step 5:

If ($res_{j^*} \le \rho$)
then { $j^* = M + 1$; $M = M + 1$ }

Adapt weights according to:

$$d_{j^*i}^{(new)} = d_{j^*i}^{(old)} x_i \quad , \quad b_{ij^*} = \frac{x_i}{0.5 + \sum_i x_i} \tag{25}$$

for $0 \le i \le N-1$.

Step 6:

repeat by going to step 2.

If Box 11 is compared to Box 10 it can be concluded, that if the calculations which have something to do with b_{ij} and out_j, are taken out, both algorithms resemble each other. This means that these two quantities are responsible for distinguishing between patterns and their supersets. That this is true can best be made clear with the example which has been used to show the unstable clustering of the best fit algorithm so again ρ = 0.65, M = 0 and $\{\underline{x}_1, \underline{x}_2, \underline{x}_3\}$ = {111, 010, 110}. The order in which these

inputs will be applied is the same as mentioned earlier. The first 6 match cycles are shown in Table 4.3.

		prototypes	
nr.	input	$\underline{d}_1$	$\underline{d}_2$
1	0 1 1	1 1 1	-
2	0 1 0	0 1 0	-
3	1 1 0	0 1 0	1 1 0
4	1 1 1	0 1 0	1 1 0
5	0 1 0	0 1 0	1 1 0
6	1 1 0	0 1 0	1 1 0
		stable coding	

Table 4.3: Stable clustering with extended best fit first algorithm

This Table shows, that if $\underline{x}_1$ is applied, this results in $\underline{d}_1 = \{1, 1, 1\}$, $\underline{b}_1 = \{1/3.5, 1/3.5, 1/3.5\}$ (equation (25)) and M = 1. Then $\underline{x}_2 = 010$ yields: $res_1 = 1$ and $out_1 = 1/3.5$ so: $\underline{d}_1 = \{0, 1, 0\}$ and $\underline{b}_1 = \{0, 1/1.5, 0\}$. M remains one. Applying $\underline{x}_3$ leads to: $res_1 = 1/2$ and $out_1 = 1/1.5$. Because $res_1 < \rho$ a new node is committed: M = 2, $\underline{d}_2 = \{1, 1, 0\}$ and $\underline{b}_2 = \{1/2.5, 1/2.5, 0\}$.
Now again $\underline{x}_1$ is applied: $res_1 = 1/3$, $res_2 = 2/3$ so $j^* = 2$ but $\underline{d}_2$ and $\underline{b}_2$ remain unchanged. Next $\underline{x}_2$ is applied. It is important to note, that the best fit algorithm went wrong here.
For the extended best fit algorithm this is not true. Here $res_1 = 1$ and $res_2 = 1$. Therefore out_1 and out_2 both have to be calculated: $out_1 = 1/1.5$; $out_2 = 1/2.5$ and therefore $j^* = 1$. This shows, that now $\underline{d}_1 = 010$ is chosen as the prototype which best matches $\underline{x} = 010$. After adjustment of the weights $\underline{d}_1$ and $\underline{b}_1$ remain the same. If next $\underline{x}_3 = 110$ is applied: $res_1 = 1/2$ and $res_2 = 1$. In this case output node 2 is chosen which means that $\underline{x} = 110$ matches $\underline{d}_2 = 110$ best.

What does this example show?
It shows that this algorithm, as opposed to the best fit first algorithm, is able to distinguish between a pattern and its supersets and that therefore both patterns can be assigned to their own prototypes. Question now is, whether this is a general property or only true for this example. It is easy to prove that this is a general property.

Proof 2:

Suppose that there are two input patterns $\underline{x}_1$ and $\underline{x}_2$ which both are assigned to an own output node then $\underline{d}_1 = \underline{x}_1$ and $\underline{d}_2 = \underline{x}_2$. Pattern $\underline{d}_2$ is said to be a subset of $\underline{d}_1$ so eq. (21) is valid for $\underline{d}_1$, $\underline{d}_2$, $\underline{x}_1$ and $\underline{x}_2$. Also, D_2 (the number of ones in $\underline{x}_2$) D_1 (the number of ones in $\underline{x}_1$).
It has already been shown, that if $\underline{d}_2$ is a subset of $\underline{d}_1$ and $\underline{x} = \underline{d}_2$ will be applied, this results in $res_1 = res_2 = 1$ in the best fit algorithm. This is also true in this algorithm. This means, that then out_1 and out_2 have to be calculated:

$$out_1 = \sum_i x2_i \cdot b1_i = \frac{1}{0.5 + D_1} \cdot \sum_i x2_i \cdot x1_i = \frac{D_2}{0.5 + D_1} \tag{26}$$

$$out_2 = \sum_i x2_i \cdot b2_i = \frac{1}{0.5 + D_2} \sum x2_i \cdot x2_i = \frac{D_2}{0.5 + D_2} \tag{27}$$

Because $D_2 < D_1$, $out_2 > out_1$ which means, that node 2 will be assigned to input $\underline{x}_2$. This proves, that the network clusters a subset pattern with its subset prototype even if a superset of that pattern is present. It is easy to verify, that a superset pattern will be clustered with a superset prototype even if subset prototypes are present. Therefore this proves, that the extended best fit algorithm can distinguish between patterns and their supersets or subsets.

❑

This proof now is very useful, because it shows, that the extended best fit algorithm overcomes the non-stability problems of the best fit algorithm. A combination of bottom-up and top-down calculations is apparently necessary to overcome these problems. Question now is, why this algorithm is not implemented as such in the ART 1 network?

In order to answer this question, the ART 1 algorithm (Box 8) and the extended best fit first algorithm (Box 11) have to be compared. If this is done, then can be concluded, that both algorithms include the same steps. There is only one important difference between both algorithms: the order in which the bottom-up and top-down calculation takes place. In Box 8 the bottom-up calculation is used to select a winning node. After the top-down calculation together with the vigilance test it is checked whether this choice was right. In Box 11 a possible candidate winning node is selected with a top-down calculation. The ultimate winner is selected with a bottom-up calculation. Whether this choice is correct is also checked with the vigilance test.
The resemblance between both algorithms seems to suppose, that the ART 1 network perhaps also selects a best fit however the different way of assigning a winning node seems to suppose that this is not the case. This gives rise to a new question: does the ART 1 algorithm also implements a best fit first or is the resemblance just a coincidence. The answer to this question can not be given shortly. On one hand, the simulation results in table 4.1 and 4.2 show that ART does not select the best fit first, because the first match cycles searches through output nodes. On the other hand, during the last match cycles stable clustering is reached with direct access to the prototypes which best fit the input. This is nothing but a best fit first! This leads to a final question: do Carpenter and Grossberg also implement an algorithm which selects the best fit eventually and if so why do they do it their way and not according to the algorithm in Box 11 ? The answer to this last question is short: *self organisation.*

When the first input is applied to a network, every output node can be associated with that input due to the initialisation of the bottom-up weights. When a choice is made the input is incorporated into that node's LTM. Then a next input is applied. Again a bottom-up calculation is executed and a winning node is selected. The choice of this winning node now depends on off a few things: first of course whether this input fits the prototype of that node. Second it depends on a trade off between weight initialisation and weight adjustment. It may be clear, that a network with small valued bottom-up weights and high valued adjusted weights will search through committed nodes before uncommitted nodes (unless resemblance is very small). A network with high initial values will assign uncommitted output nodes before committed ones. This shows very clear, that the node assignment depends on a trade off between initial bottom-up weights and the size of weight adjustment after a fit. It also shows that the ART network becomes more flexible and self organising in this way: it can "decide" whether it starts a search or assigns an input to an uncommitted node.
It may be clear, that this problem occurs in the beginning of a simulation. There the trade off is the strongest. It is difficult to choose initial values and weight adjustments balanced so in the beginning it can happen, that an output node which does not resemble the input is chosen before an output node which fits the input.
It may also be clear that this problem becomes less if the network has been trained for a while. During the simulation, weights are adjusted after every fit. This means, that the prototypes resemble the inputs more and more. As a result of this, the trade off between initial bottom-up weights and prototype vectors becomes of less importance because the values of connections in vectors which store prototypes increase. Due to this, the chance that a committed node wins the competitive process increases and it is easy to understand, that the node which resembles the input best has the largest chance to win. This explains how the network during training can slowly organise itself to select the

best fit first. Of course it is still very important that initial values and learning rules are chosen suitable. That the choices in Box 8 and Box 9 can lead to a satisfying clustering can be concluded from the results in Table 4.1 and 4.2.

4.4 Conclusions

In this chapter, the ART 1 network has been treated very detailed. With use of a best fit algorithm and an extension to this algorithm it is tried to clarify the meaning of the different steps in the ART 1 algorithm and how they cooperate in order to come to a network which is able to cluster input patterns by self organisation into a stable clustering. In this clustering every input accesses the cluster to which it belongs directly. Simulations have shown, that the network indeed is capable in doing this task.

Simulations have also shown, that the Lippmann interpretation satisfies the ART 1 network which has been developed by Carpenter and Grossberg. They both cluster input patterns according to the same criteria. The way this clustering develops as a function of time can be different. It has been shown, that this is due to the fact that the adjustment of bottom-up weights is slightly different in both algorithms. Because the assignment of a winning node depends on a trade off between initial bottom-up weight values and adjusted bottom-up weight values, it is not difficult to understand that the clustering processes can differ.

In the simulations which have been done, fast learning was used. Due to this fast learning prototypes in the LTM can "decrease". This can lead to forgetting of earlier learned input patterns which is no solution to the plasticity-stability dilemma which states that a network does not forget past learning if new inputs are applied. This contradiction probably arises due to a different interpretation of the word forgetting. In this report it is assumed, that a pattern is forgotten if it can not be assigned to the node it was assigned to the last time it has been applied.
Due to this forgetting, the ART network is able to reach stable clustering. Condition then is however, that the dataset is applied to the network repeatedly. If that happens a stable clustering will always be reached if there are enough output nodes available. The order in which the inputs are applied can influence the time in which stable clustering is reached.

5 The Adaptive Resonance Theory 2 (ART 2) network

In this chapter 2 different ART 2 algorithms will be discussed. Together with stylistic Brainstem Auditory Evoked Potentials the behaviour of these two networks is studied and it is tried to get some insight in a number of ART 2 properties.

5.1 The network and the network equations

The ART 1 network, as discussed in chapter 4, is able to create stable recognition codes by self organisation in response to arbitrary sequences of binary input patterns. For analog input patterns this network can not be used. In the BAEP problem, the input patterns are analog (grey scaled, discrete valued). This means, that another network has to be used in order to be able to self organise these patterns into stable recognition codes.
ART 2 is a network that is able to do this. It was also developed by Carpenter and Grossberg and was introduced in 1987 [Carpenter and Grossberg, 1987b]. Different architectures are still subject of ongoing research. A typical network architecture is shown in Fig. 5.1.

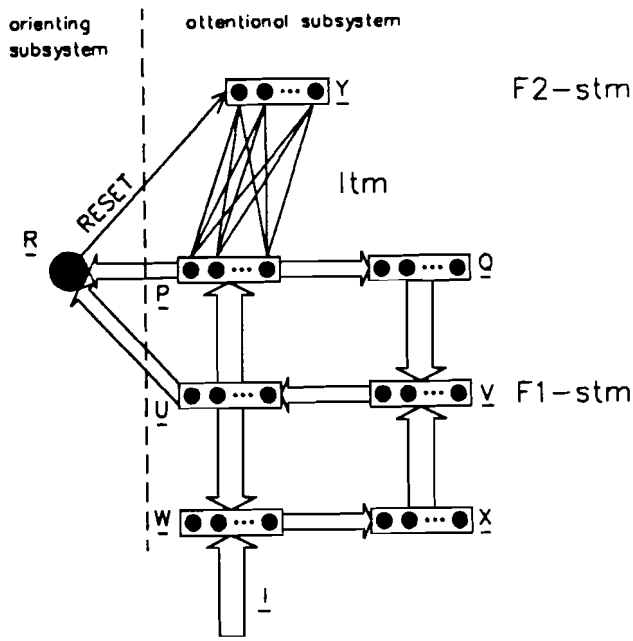

Fig. 5.1: Typical ART 2 network architecture

The network in this figure, does not resemble the ART 1 network at first sight. Yet it will be shown that it contains the same parts, although implemented differently. As can be seen in Fig. 5.1, the ART 2 network consists of a number of arrays which are interconnected. These arrays can be divided in different groups. $\underline{w}$, $\underline{x}$, $\underline{v}$, $\underline{u}$, $\underline{p}$ and $\underline{q}$ form the *feature representation field* F1. Every array within this field has dimension N. $\underline{y}$ is the *category representation field* F2. It has dimension M. The arrows between arrays in the feature representation field denote vector operations on independent nodes within the arrays. Both fields can be regarded as the *Short Term Memory* (STM) of the ART 2 network. The arrows between these fields denote 2 matrices which form ART 2's *Long Term Memory* (LTM). One matrix stores ART's *bottom-up filters*, while the other one stores ART's *top-down filters*. Together with the 2 fields they form the *attentional subsystem* of the ART 2 network.
ART 2 also has an *orienting subsystem*. This system consists of an array $\underline{r}$ with dimension N and a vigilance parameter ρ. It is connected to F1 by two vectors of the same length. The "output" of the orienting subsystem indicates whether a RESET has to take place or not.

Compared to ART 1 the ART 2 architecture looks much more complicated. This does not count for the equations, which are rather simple. The equations of ART 2 can be grouped into 4 sets. The first equation set describes the F1 STM equations:

$$w_i = I_i + a u_i \tag{1}$$

$$x_i = \frac{w_i}{e + \|w\|} \tag{2}$$

$$v_i = f(x_i) + b f(q_i) \tag{3}$$

$$u_i = \frac{v_i}{e + \|v\|} \tag{4}$$

$$p_i = u_i + \sum_j g(y_j) z_{ji} \tag{5}$$

$$q_i = \frac{p_i}{e + \|p\|} \tag{6}$$

In these equations a and b are multiplication factors which can be seen as feedback constants, e is a normalisation constant, $|.|$ denotes the L2-norm of a vector, z_{ji} is a top-down connection, $g(y_j)$ is a kind of switching function whose meaning will become clear later on, and f(.) is a nonlinear function. A possible choice for f(.) is:

$$f(x) = \begin{cases} 0, & 0 \le x < \theta \\ x, & x \ge \theta \end{cases} \tag{7}$$

in which θ is a parameter which expresses the size of non linearity which is incorporated into the network. The function f(.) only passes positive value which exeed θ. It therefore is not suitable if negative valued input patterns are used.

The second group of equations that can be formed are the equations that describe the F2 STM. They resemble the ART 1 equations.

$$T_j = \sum_i p_i z_{ij}, \quad \forall j,\ 0 \le j \le M-1 \tag{8}$$

$$T_J = \max\{T_j : j = 0..M-1\} \tag{9}$$

$$g(y_J) = \begin{cases} d & \text{if } T_J = \max\{T_j\} \\ 0 & \text{else} \end{cases} \tag{10}$$

Here d is a constant between 0 and 1, which can be regarded as a learning rate.

The third group of equations describe the vigilance test. This test is guided by:

$$r_i = \frac{u_i + c \cdot p_i}{e + \|u\| + \|c \cdot p\|} \tag{11}$$

in which c is a constant. A RESET is generated if:

$$\frac{\rho}{e + \|r\|} > 1 \tag{12}$$

Here ρ is the vigilance parameter which has the same meaning as in ART 1 so $0 \leq \rho \leq 1$.

The last set of equations describes the learning rule(s) of the ART 2 network. For the top-down LTM from a winning node J this rule is:

$$\frac{d}{dt} z_{Ji} = d[p_i - z_{Ji}] = d(1 - d)[\frac{u_i}{1 - d} - z_{Ji}] \tag{13}$$

For the bottom-up LTM to winning node J:

$$\frac{d}{dt} z_{iJ} = d[p_i - z_{iJ}] = d(1 - d)[\frac{u_i}{1 - d} - z_{iJ}] \tag{14}$$

Besides these equation sets, there are also 3 demands which have to be satisfied when initialising the network.
The first demand concerns the initial value of the top-down weights. These have to be set equal to:

$$z_{ji}(0) = 0, \quad \forall\, i,j \tag{15}$$

The second demand concerns the values c and d. These have to satisfy:

$$\frac{c \cdot d}{1 - d} \leq 1 \tag{16}$$

The third demand concerns the initial values of the bottom-up weights. These have to be set to:

$$z_{ij}(0) = \frac{1}{(1 - d)\sqrt{N}} \quad \forall\, i,j \tag{17}$$

The reason for these demands will become clear while discussing the algorithm.

The network which is shown in Fig. 5.1 and which can be described with the above 17 equations, has to implement the same clustering algorithm as the ART 1 network. The only difference is, that it must be able to handle analog input patterns. This means that, if an input is applied, this input must be associated with an output node by a bottom-up calculation. Then it must be tested whether the prototype of this output node fits the input. This is done by the vigilance test which is preceded by a top-down calculation. The result of this test can be two-fold. If it results in a fit, the input has to be incorporated into the prototype of the winning output node. If the test results in a non-fit, a new output node has to be associated to the input and a new vigilance test will be executed. This process will be repeated until a fit has been found. After that, a new input can be applied to the network.

If the ART 2 network is compared to the ART 1 network, a few resemblances and a few differences attract attention. The most important resemblance is, that for both networks the same parts can be distinguished: F1, F2, a bottom-up LTM, a top-down LTM, and a vigilance test. The most important difference is the interpretation of the F1 STM: in ART 2 there are 3 layers which consist of 2 arrays each, instead of one vector with 2 associated arrays in ART 1. Other important differences are the vigilance test and the learning rules.

These differences give rise to a number of questions:

1. What is stored into ART 2's Long Term Memory?
2. Why does the F1 STM consist of 3 layers and what is the meaning of the different steps within this layer?
3. What is the meaning of the RESET function? What does it compare?

When studying literature about this network, answers to these questions can only be found partially [Carpenter & Grossberg, 1987b]. This means, that there remain only two possible solutions to find these answers: analytical analysis or simulation of the ART 2 network. We have chosen for the second possibility. This also was a problem at the beginning of our research on ART 2 because literature didn't give a straight-forward algorithm which implemented the ART 2 network. The order in which network operations have to take place was unknown then. Literature gave only one indication: "matching takes place at the middle F1 layer".
Due to this, the ART 2 algorithm had to be discovered by ourselves. Fortunately literature discussed properties of the network, with which possible candidate algorithms could be checked:

- Stability - Plasticity trade off

This trade off has been mentioned earlier when discussing the ART 1 network. Now a slightly different interpretation is used: "since the plasticity of an ART system is maintained for all times and since input presentation times can be of arbitrary duration, STM processing must be defined in such a way, that a sustained new input pattern does not wash away previous information" [Carpenter & Grossberg, 1987].

- Search - Direct access trade off

The network must carry out a parallel search in order to find an appropriate recognition code during the learning process, but has to engage the search process automatically, if an input pattern becomes familiar. This property has also been mentioned while discussing ART 1.

- Match - RESET trade off

This trade off expresses, that the network must be able to suppress a RESET automatically if an uncommitted node is assigned to the input or if a fit with a committed node is found. It has to generate a RESET to the assigned output node if a non-fit is established.

- Contrast enhancement and noise suppression

The network has to be able to separate signal from noise and to enhance contrasts in input patterns.

- Rapid self stabilisation

The network has to be able to reach a stable clustering in only a few learning trials.

It took a lot of time to find an algorithm which satisfied most of these properties, in literature called *design principles*.

In the mean time the search in literature for a straight-forward algorithm continued, which finally resulted in the discovery of a simulation program from a student of Carpenter and Grossberg (Paulo Gaudiano, Boston University, Centre for Adaptive Systems, Boston). From this program, the right sequence of calculations and thus the algorithm could be found.

This chapter now discusses "our" algorithm and the algorithm which has been abstracted from the simulation program. Section 5.2 treats our interpretation of the ART 2 network. In that section an attempt will be made to show how the different parts of an ART 2 network cooperate in order to implement the design principles. This is done on the basis of computer simulations with which the meaning of the different parameters is investigated. These simulations give answers to the questions which arise when studying ART 2 networks. Section 5.3 then discusses the ART 2 algorithm which has been extracted from the computer program.Because this computer program became available at the end of this research project it has not been tested as extensive as the algorithm in section 5.2. In 5.3 also the differences between both interpretations and the problems to which these differences can lead will be discussed. In both sections attention is only paid to how the ART 2 algorithms handle data. The clustering of datasets and how this clustering process can be influenced is the subject of chapter 6 in which an attempt is made to solve a classification problem with an ART 2 neural network.

5.2 The ART 2 training algorithm (version 1)

The algorithm which has been found, is a result of combining ART 2 literature with ART 1 knowledge and the statement, that matching takes place at the middle F1 layer [Carpenter & Grossberg, 1987b]. As in ART 1, all parameters (a .. e , θ and ρ) and the weights are initialised. Then, all arrays in F1 are set to $\underline{\mathbf{0}}$. An input is applied to the network and calculated through the F1 layer from $\underline{\mathbf{w}}$ via $\underline{\mathbf{x}}$, $\underline{\mathbf{v}}$ and $\underline{\mathbf{u}}$ to $\underline{\mathbf{p}}$. During this calculation the pattern is normalised, it undergoes a non linear operation and is normalised again. The pattern which emerges at the top of F1 undergoes a bottom-up calculation to the output $\underline{\mathbf{y}}$ where, due to some criterion, a winner is selected. Next, a top-down calculation takes place, which results in a new pattern in $\underline{\mathbf{p}}$. This pattern is normalized and fed back via the non linear function to $\underline{\mathbf{v}}$. Simultaneously $\underline{\mathbf{u}}$ is fed back to $\underline{\mathbf{v}}$ via $\underline{\mathbf{w}}$ and $\underline{\mathbf{x}}$. In $\underline{\mathbf{v}}$ both patterns are matched and normalized to $\underline{\mathbf{u}}$. As a last step $\underline{\mathbf{r}}$ is calculated and the vigilance test is executed. The outcome of this test is twofold. If eq. (12) is satisfied, the input does not fit the prototype stored in the LTM of the winning output node. This means that the winner has to be RESET, all F1 arrays must be set to $\underline{\mathbf{0}}$ again and the above described process has to be repeated until it results in the assignment of a winner that fits the input. Then, this input has to be incorporated into ART's LTM. After that, the network is ready to accept a new input pattern in order to associate this pattern with a cluster. If eq. (12) is not satisfied the input also fits. Then the input is also incorporated and a new input can be applied.

In order to test whether this algorithm could be an implementation of the ART 2 network, it has been tested with the same simulations as the ones used to test the ART 1 network. From these simulations two properties become clear:

1. If the input set is applied repeatedly to the network it is possible to reach a stable clustering in which every cluster is direct accesible. This satisfies the fifth design principle mentioned above.
2. Although a stable clustering is reached, the magnitude of the vectors stored into LTM increases.

The first property shows, that an algorithm is implemented, which is able to reach a stable clustering. The fact that the LTM has not been stabilised need not to be in contradiction with this statement because this can be due to the fact that *slow learning* is used. That this consideration is true can be proven if the dataset is applied repeatedly. Then the simulations show that the LTM vectors converge to a final value.

Due to these properties it can be concluded that the algorithm implements a network which groups input patterns into a stable clustering if these patterns are applied repeatedly. Due to the fact that the algorithm is based upon ART 2 literature, it can therefore be expected that it implements the ART 2

algorithm and that it can be used to investigate the ART 2 network. Goal of that research is to find the answers to some questions which arise when studying ART 2 literature. In order to find these answers simulations have been done. The next sections discus the simulations which have to clarify how ART 2 handles data. While studying the different parameters within the ART 2 network we tried to discover what is stored into LTM, what the effect of the different parameters and the nonlinear function is on that LTM and what the meaning is of the RESET function.

5.2.1 Prototype storage in Long Term Memory

In order to discover what is stored into LTM and what the influence of the parameter a .. e , θ and ρ on these prototypes is, a number of experiments has been set up.

In the first experiment a pattern (ep1_47, see Fig. 5.2) has been applied to the network repeatedly. Because this pattern is both positive and negative valued the non linear function has been adjusted in such a way that the network also can handle negative values. During the simulation the following parameter settings have been chosen: a = b = 1.0, c = 0.1, d = 0.9, e = θ = ρ = 0.0.

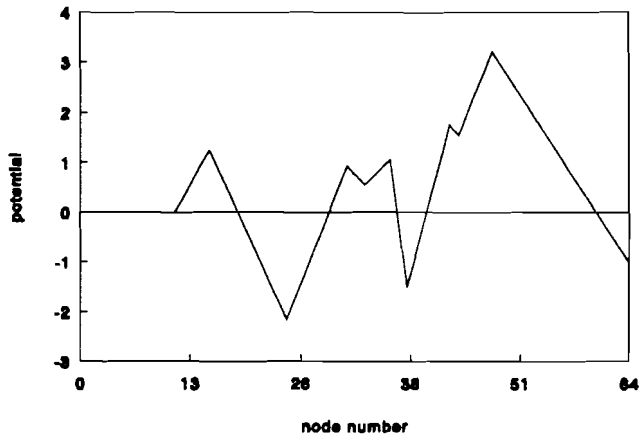

Fig. 5.2: pattern ep1_47

During the simulation, the input patterns have been associated to only one output node. This means that only the LTM vectors to and from this output node have been adjusted during learning. When these vectors are studied, it can be concluded that they both converge to a stable pattern which resembles the input exactly. This result is not amazing. It can be proven (Appendix 1a), that both LTM vectors converge to:

$$\frac{\underline{x}}{\|\underline{x}\|} \cdot \frac{1}{1-d} \qquad (18)$$

if $\underline{x}$ is repeatedly applied to the network. This shows that the pattern which is stored into LTM is proportional to the normalised input.

From this, 2 properties become clear:

1. The patterns stored in the bottom-up and top-down filter resemble.
2. If d is varied, this does not influence the shape of the pattern which will be stored in LTM. It has only effect on its length.

Simulations with different values of d confirm this. From these simulations also can be concluded, that d also affects the time in which the LTM stabilizes. A strange effect occurs as can be seen in Fig. 5.3, which shows that, when if d is taken near 0 and 1 it takes more time to reach 95% of the final length

of the LTM then when d = 0.5. This can be explained by a trade off between the final value of the LTM and the step size with which the weights are adapted.

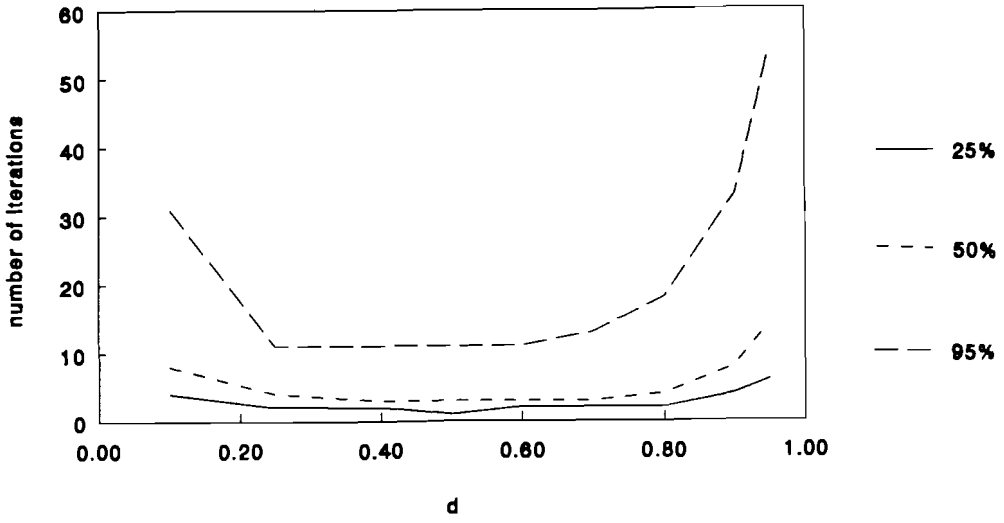

Fig. 5.3: "Stabilising speed" as a function of d

In a second experiment two patterns (**a** and **b**, see also Fig. 5.4 (a) and (b)) are applied to the network alternately. The same parameter settings are used as within the first experiment. Now the initialisation of the bottom-up filter has been chosen such, that both inputs are forced to the same output node. Because $\rho = 0$ these associations are always accepted as a fit. In Fig. 5.4 (c) is shown which pattern is stored into LTM if both patterns are applied alternately and repeatedly and LTM has stabilised. Question now is, whether this pattern can be seen as a mixture of **a** and **b**.

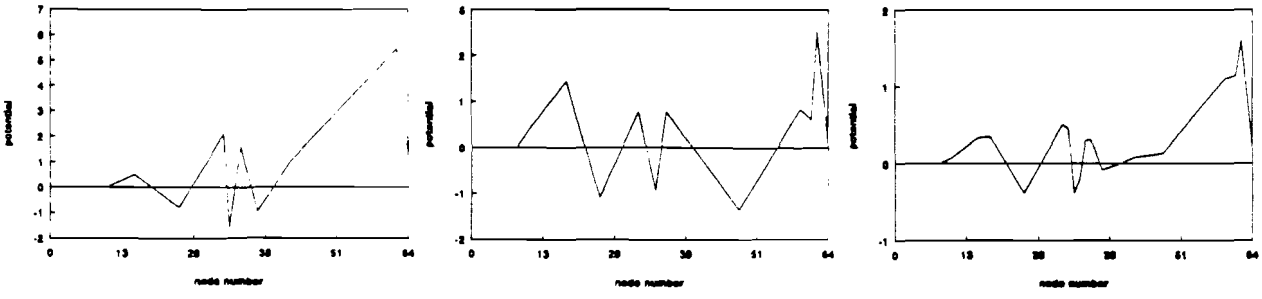

Fig. 5.4: (a) ep16_61, (b) ep17_61 and (c) stabilised LTM prototype

In Appendix 1b it is proven, that this is indeed true. It is shown there, that if 2 patterns **a** and **b** are alternately and repeatedly applied to the network and associated with the same output node, the LTM vectors **z** to and from that output node satisfy:

$\underline{z} = c_1 \cdot \underline{a} + c_2 \cdot \underline{b}$ *if* $\underline{a}$ *is the last applied input*

and

$\underline{z} = c_2 \cdot \underline{a} + c_2 \cdot \underline{b}$ *if* $\underline{b}$ *is the last applied input*

Here $c_1 + c_2 = 1/(1-d)$ *and* $c_1 = c_2 = 0.5/(1-d)$ *if* $d \rightarrow 1$.

This shows, that if d → 1, the average of **a** and **b** is stored into LTM. If d < 1 the prototype switches between 2 stable patterns round this average. It depends on the last applied input which pattern is incorporated slightly more in LTM.

As a last test to see what happens in ART 2's LTM, pattern **a** is applied to the network a number of times (NVEK). After that, **b** is applied to the network also NVEK times. Again **a** and **b** are forced to be associated with the same output node. If a look is taken at the pattern stored into LTM it can be seen, that this pattern resembles **b** more and more if NVEK increases. In Appendix 1c it is proven, that the prototype stored into LTM converges to a pattern which is proportional to **b** if NVEK → ∞. This is also shown in Fig. 5.5. This means, that if pattern **b** is sustained applied to the network, it washes out pattern **a**, which means that the plasticity-stability dilemma is not solved here.

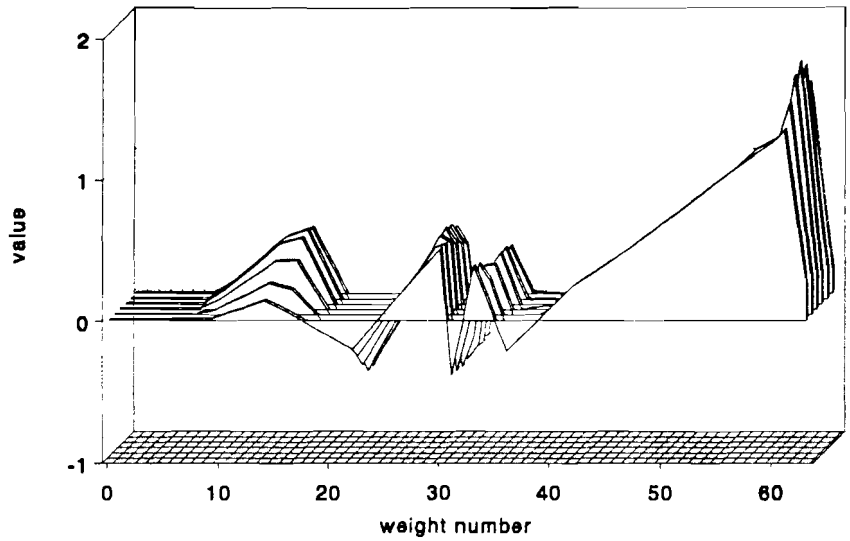

Fig. 5.5: Prototype storage as a function of NVEK (NVEK increases from back to front)

5.2.2 Introduction of a non linearity

In section 5.2.1 has been studied what is stored into LTM if θ = 0. Due to this choice, the patterns which circle within the network always look like input patterns or "mixtures" of these patterns. By assigning θ ≠ 0, non linearities are introduced into the network. It does not need many insight to conclude, that this will influence the patterns within the network and thus the patterns which will be stored into LTM. This influence will be examined in this section. This is done in the same way as in experiment 1, in which pattern ep_47 is applied repeatedly to the network until LTM stabilizes. Then the prototype in LTM is studied. The parameter settings are chosen equal to the settings in preceding experiments; θ only is varied.

Fig. 5.6 shows which patterns are stored into LTM if θ varies. From this figure can be concluded, that the network indeed performs some feature extraction or contrast enhancement. Due to the cooperation between the non linear function and normalisation, the network attenuates maxima in the input patterns.

After this experiment, the influence of a, b and d has been tested for different values of θ. From these experiments, which are not shown here, can be concluded, that d has the same effect on the prototype as when θ = 0: it effects the final length of the prototype and the convergence speed. Variation of a and b does not influence the final shape or final length of the stored patterns. It influences the convergence speed.

Finally, also experiments which have been done with two input patterns **a** and **b** in section 5.2.1 have been repeated with the nonlinearity incorporated into the network. The conclusions which can be drawn from these experiment are the same as in section 5.2.1 although the patterns which are stored are now contrast enhanced patterns.

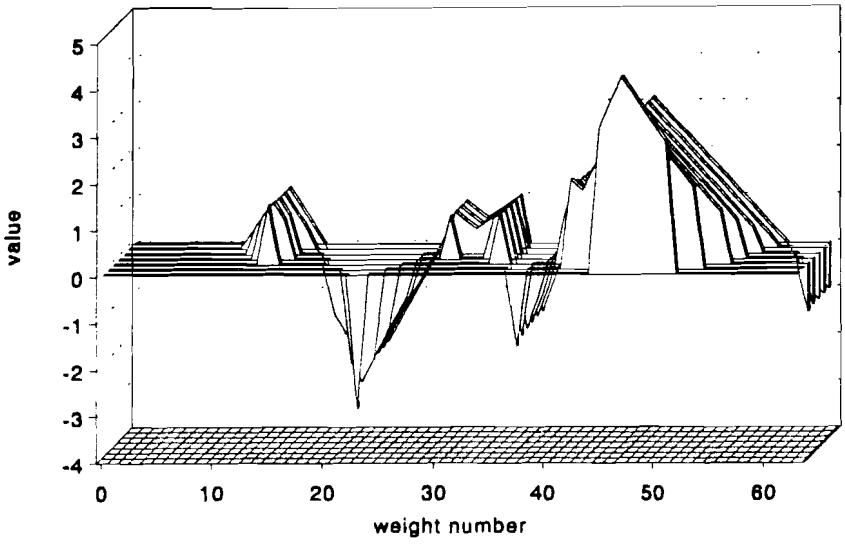

Fig. 5.6: Prototype storage as a function of θ *(*θ *increases from back to front)*

5.2.3 The vigilance test

The vigilance test is guided by 2 parameters: c and ρ. This means, that these 2 parameters determine whether a node is assigned to a cluster or not. The way the network decides this, is guided by eq. (11) and eq. (12). In the test two vectors are compared: **u** and **p**. Pattern **u** can be related to the input pattern; **p** to the prototype stored into LTM. The way these patterns are compared is a little difficult to understand. A kind of normalisation is applied on the sum of the elements of both patterns and the length of this normalised sum **r** is compared to a distance measure, the vigilance parameter, in order to test whether both patterns fit or not.

In order to get insight in the way ART 2 compares two vectors the vigilance test is visualised in 2 dimensional space. **u** and **p** are therefore chosen to be 2 two dimensional vectors: **p** = (1,0) and **u** = (u_1,u_2). Making use of eq. (11) and eq. (12), the areas in 2 dimensional space can be found in which **u** fits **p** for different values of c and ρ. In Appendix 2 is calculated, that the curve for which **r** = ρ can be described by:

$$R = \frac{-2 \cdot m - 1 + \cos\varphi \pm \sqrt{(2 \cdot m + 1 - \cos\varphi)^2 - 4 \cdot m^2}}{2 \cdot m / c} \qquad (19)$$

In this equation (R,φ) denotes 2 dimensional space in polar coordinates while $m = 0.5(\rho^2 - 1)$. This equation describes a cardioid like curve. If **u** lies in the area outside this curve, **u** is said to fit **p**.

In Fig. 5.7 different curves are plotted. In fig. 5.7 (a) ρ is varied while c is set to 0.1. In Fig. 5.7 (b) c is varied while ρ is set to 0.98. Both plots show that for increasing ρ and c the area in which **u** is said to fit **p** becomes smaller. It also shows, that the area which "looks" in the direction of **p** (direction (1,0)) becomes more narrow. Both conclusions lead to the property that increasing ρ and c lead to more clusters. Both plots also show a strange property: if **u** is long enough, **u** is accepted as a fit, even if **u** is perpendicular or even mirrored to **p**. They also show that if the length of **u** is above some limit, **u** will always fit **p** too. Both properties are very annoying if patterns have to be clustered. In ART 2

these properties are ruled out by normalising $\underline{u}$. By doing this, the area at which $\underline{u}$ lays reduces to the unity circle.

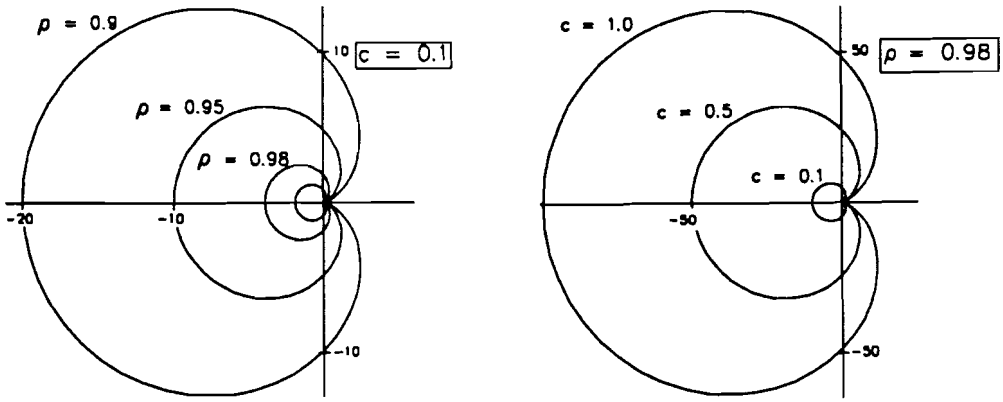

Fig. 5.7: Fit curves as a function of (a) ρ and (b) c

As can be seen in Fig. 5.8 (a) and (b), which are the enlarged versions of Fig. 5.7, the area in which a fit is accepted reduces to the arc of a circle in between the legs of the parabola kind curves.

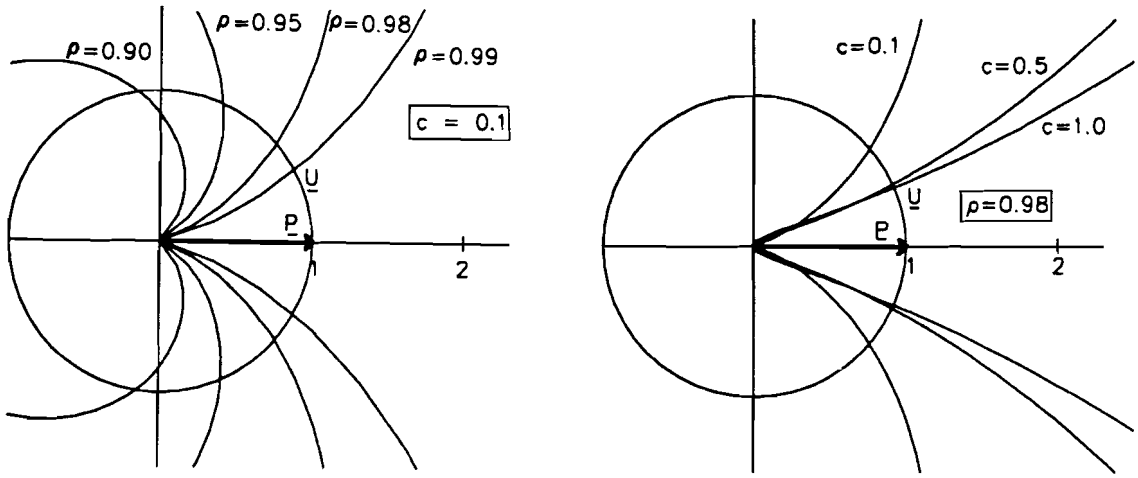

Fig. 5.8: Fit area as a function of (a) ρ and (b) c

These figures now show how the vigilance test compares two patterns. From the origin it looks in the direction of pattern $\underline{p}$ and accepts all vectors which lay between the legs of the "parabola" at distance 1 from the origin, as a fit. The figures show, that this area becomes larger if ρ and c decrease. Fig. 5.8 (a) shows that ρ needs to be rather large is c is small. For ρ = 0.90 this figure shows, that (0,1) is accepted as a fit to (1,0).

Besides testing whether a pattern fits the prototype stored in LTM, eq. (11) and eq. (12) together are responsible for another property of the ART 2 network: implementation of the match-RESET trade off. In literature [Carpenter & Grossberg, 1987] it is shown that cooperation between eq. (11) and eq. (12) is responsible for the property that assignment of an uncommitted node always suppresses a RESET and that of course in case of a fit a RESET will never be generated. In order to reach this effect the initialisation of top-down weights has to satisfy eq. (15).

5.2.4 The normalisation steps

During all preceding simulations parameter e was set to 0. Due to this choice all patterns which circle in F1 are normalised. If e ≠ 0 this changes. If a vector will be normalised then, its length after

normalisation depends on its length before normalisation. With eq. (20) this can be proven. Here $\underline{x}$ is the vector which has to be normalised. $\underline{y}$ is the normalised vector. From eq. (20) can be concluded, that $|\underline{y}|$ becomes a function of $|\underline{x}|$ if e is chosen to be a non zero constant.

$$\underline{y} = \sqrt{\Sigma y_i^2} = \sqrt{\Sigma \left(\frac{x_i}{e + |\underline{x}|} \right)^2} = \frac{1}{1 + e/|\underline{x}|} \tag{20}$$

A test has been set up in order to investigate the effect of variation of e on the patterns stored into LTM. Two simulations have been done. In the first simulation e is varied while for the other parameters the initial parameter settings are used. Fig. 5.9 shows the patterns that are stored after LTM has converged.

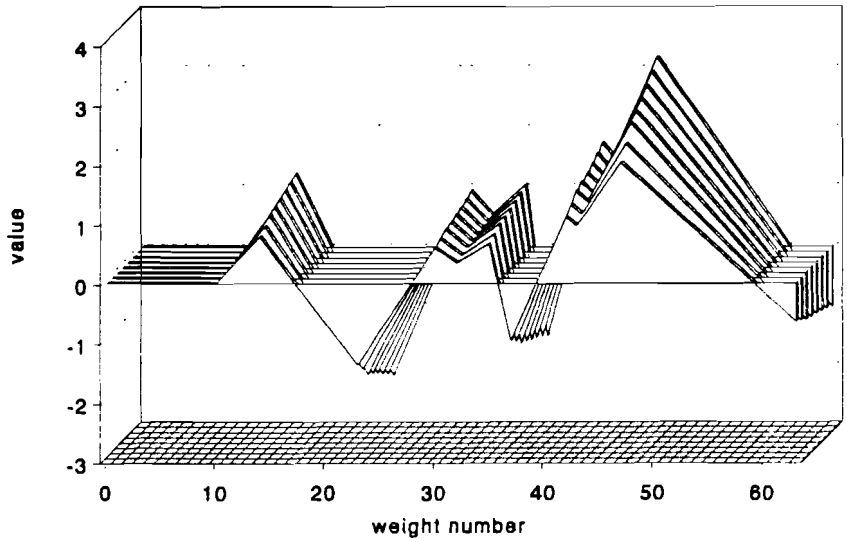

Fig. 5.9: Prototype storage as a function of e (e increases from back to front)

In the second simulation θ is set to 0.1 and e is varied. The prototypes stored in LTM then are shown in Fig. 5.10.

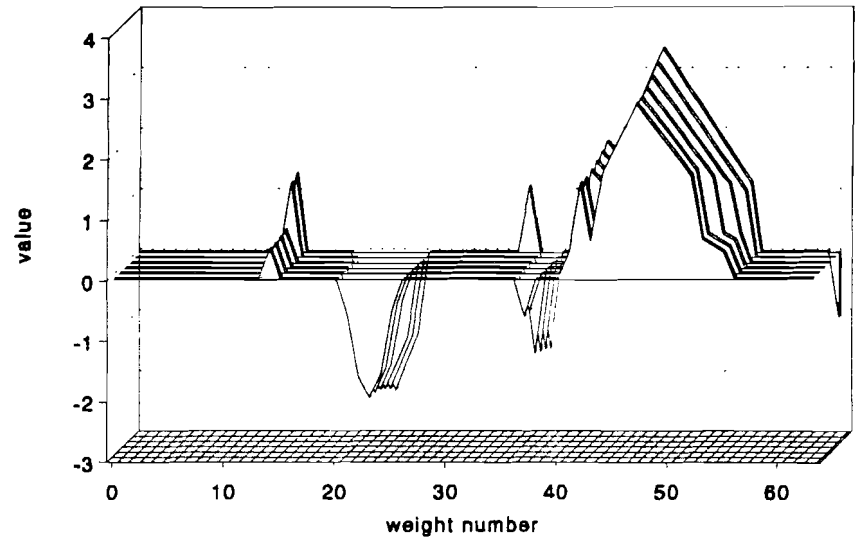

Fig. 5.10: Prototype storage as a function of e while θ = 0

From these figures can be concluded, that if $\theta = 0$ parameter e only influences the size of the patterns stored into LTM. If $\theta \neq 0$ parameter e also influences the shape. Because the shape depends on e and the size of non linearity, it is impossible to draw clear conclusions about how e influences the shape.

5.2.5 Conclusions about ART 2 version 1

The ART 2 algorithm which has been discussed in section 5.2, has shown to be able to cluster a set of input patterns into stable recognition codes, if the input is applied repeatedly to the network. This can even happen in only a few learning trails (rapid self stabilisation). It has also been shown, that the match-RESET trade off is implemented by the equations which describe the vigilance test and a proper choice of the initial top-down weights. Besides that, the network has shown to be able to contrast enhance patterns by cooperation between the non linear function and normalising steps within F1. It looks like the plasticity-stability dilemma is not solved by this network. That the search-direct access trade off is implemented can easily be explained here by referring to chapter 4. There has been stated, that node assignment depends on a trade off between the size of initial bottom-up weights and weight adjustment. If the initial bottom-up weights are set too high, this results in assignment of uncommitted nodes before committed nodes. In order to reach stable clustering and direct access of committed nodes, these nodes have to win the competitive process if stable clustering is reached. This is achieved by initialising the initial bottom-up weights according to eq. (17). With this initialisation it is accomplished, that the length of the vector to an uncommitted node never exceeds 1/(1-d). From eq. (17) it becomes clear that in case of equality the probability, that uncommitted nodes win the competition during learning is the largest. Reducing the initial value of bottom-up weights results in a network in which the probability that an uncommitted node wins the competition reduces. In such a network a search through committed nodes will be prefered above assignment of an input to an uncommitted node.

The above shows, that 4 of the 5 design principles mentioned in 5.1 are implemented. From this should be concluded, that the algorithm does not implement the ART 2 network exactly. The design principle which is not satisfied is the plasticity-stability dilemma. Because this dilemma was not implemented in the ART 1 network according to our opinion, the fact that it is not implemented in this ART 2 algorithm is not used to reject the algorithm as an approximation of the ART 2 network. The fact that with this algorithm and the simulations which have been done with it, the questions that are propound in section 5.1 can be answered indicate, that the algorithm can be seen as a good approximation.

The simulations have shown that ART stores averages of patterns in LTM. If the non linearity is not incorporated into the network, the patterns stored in LTM are proportional to averages of input patterns. If $\theta \neq 0$, these patterns are averages of contrast enhanced patterns. Due to this, the F1 layer can be seen as a *pattern processor*. It processes data in order to cluster these data. It has been shown that cooperation between the non linearity and normalising steps are responsible for this contrast enhancement which is nothing but attenuating maxima in the input pattern. This brings us at the second question stated in 5.1: why does ART 2's F1 STM consist of 3 layers and what is the meaning of the different steps in these layers. A straight forward explanation to this question can not be given but it has been shown (see the previous example), that cooperation between different steps in F1 is responsible for a few properties of the network: contrast enhancement (non linearity and normalisation), matching (feed back loops) and testing for a fit (vigilance test and normalisation). F1 operation therefore has to be seen as a whole: due to the cooperation between various elements desired properties are incorporated.
The last question stated in section 5.1 concerned the meaning of the RESET function. For 2 dimensional space, it has been shown that the vigilance test and the RESET function "look" from the origin in the direction of the prototype stored in the LTM of the winning node and accept everything as a fit which is in the *range of vision* and lies at distance 1 from the origin. The range of vision depends on the parameters which guide the vigilance test. Although not proven, it is assumed that this explanation is also valid if input vectors of higher dimension are applied to and stored in the network.

In the case the areas where inputs and prototypes are accepted as a fit are areas discerned by hyperplanes.

5.3 The ART 2 training algorithm (version 2)

While searching for the ART 2 algorithm version 1 we found a computer program which simulates an ART 2 network. From this program an ART 2 algorithm could be extracted. Because the program was written by a student of Carpenter and Grossberg, it may be considered, that the algorithm implements the sequence of operations which Carpenter and Grossberg must have meant.

The network architecture which is used in the algorithm is not the typical ART 2 architecture which has been shown in Fig. 5.1 although it bears much resemblance to it. The architecture that is used is shown in Fig. 5.11.

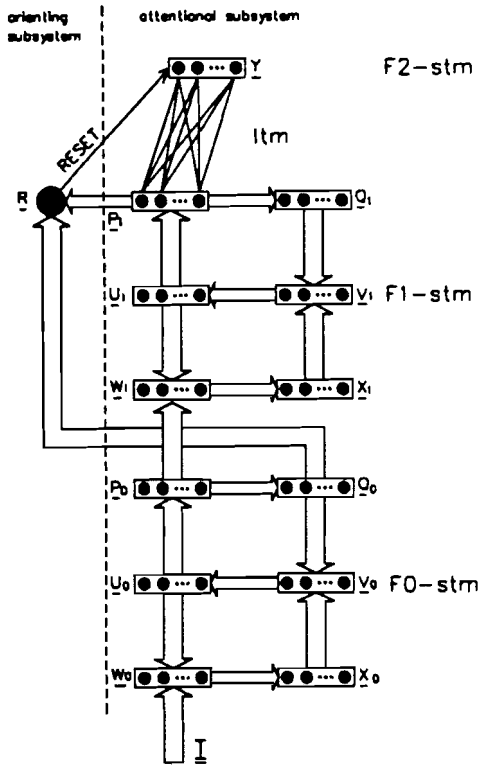

Fig. 5.11: A second ART 2 network architecture

This ART 2 architecture is an extended version of the ART 2 network of Fig. 5.1. It has been extended with a STM layer F0. Within this layer almost the same operations take place as in F1 and therefore almost all equations mentioned in section 5.1 are valid within F0. Only the calculation of $\underline{\mathbf{p}}_0$ differs from the calculation of $\underline{\mathbf{p}}$. In F0 $\underline{\mathbf{p}}_0 = \underline{\mathbf{u}}_0$. Due to the presence of F0 two equations which describe F1 change. First of course the equation in which the input is applied to F1 (eq. (1)) which becomes:

$$w_{1i} = p_{0i} + a \cdot u_{1i} \tag{21}$$

Second eq. (11) which calculates $\underline{\mathbf{r}}$, changes to:

$$r_i = \frac{q_{0i} + c \cdot p_{1i}}{e + |\underline{q}_0| + |c \cdot \underline{p}_1|} \tag{22}$$

The algorithm which is extracted from the program is as follows. First, all parameters (a..e, θ and ρ) are set to their desired initial values, all bottom-up and top-down weights are initialised and, all arrays in F0 and F1 are set to $\underline{0}$. Then, the first input can be applied to the network and simultaneously $\underline{w}_0$ and $\underline{p}_0$ are calculated. These patterns are both normalised, which results in $\underline{x}_0$ and $\underline{q}_0$, and are matched by the nonlinear functions into $\underline{v}_0$. Hereafter their match is normalised to $\underline{u}_0$. This will repeated (($\underline{w}_0$, $\underline{p}_0$) $\rightarrow$ $\underline{x}_0$, $\underline{q}_0 \rightarrow \underline{v}_0 \rightarrow \underline{u}_0$) until $\Delta\underline{w}_0$ becomes smaller than some predefined bound (ac_dif). At that point F0 is said to be stable. All F0 calculations stop and $\underline{p}_0$, the output of F0, becomes the input to F1.
In F1 now comparable calculations happen. First $\underline{w}_1$ and $\underline{p}_1$ are calculated. At this point no output node is assigned yet so $\underline{p}_1 = \underline{u}_1$. Then, these patterns are normalised; ($\underline{x}_1$, $\underline{q}_1$), matched ($\underline{v}_1$) and normalised again ($\underline{u}_1$). Again $\underline{w}_1$ and $\underline{p}_1$ (as well as $\Delta\underline{w}_1$ and $\Delta\underline{p}_1$) are calculated, followed by a bottom-up calculation, assignment of a winning node and a top-down calculation. At this point $\underline{p}_1$ contains information about the prototype which is stored in the LTM of the winning node. This pattern thus can be matched with the input so at this point the vigilance test can be executed which calculates a match between $\underline{p}_1$ and $\underline{q}_0$. If this test results in a RESET, the same happens as in the other ART 2 implementation: the winning node will be RESET, all F1 arrays are set to $\underline{0}$ and a search for a new winner is started by repeating all F1 calculations.
If a fit is established, this algorithm handles differently in compared to version 1. Of course it adjusts the weights $\underline{z}$ to and from the winning node but this algorithm also calculates $\Delta\underline{z}$, the weights change. It uses this change together with $\Delta\underline{p}_1$ and $\Delta\underline{w}_1$ to check whether the patterns in F1 are stable. This is done by comparing $\Delta\underline{p}_1 + \Delta\underline{w}_1$ and $\Delta\underline{z}$ to predefined bounds: ac_dif and z_dif. If the changes exceed these bounds, the network is said to be unstable. In that case the calculations in F1 are repeated (($\underline{x}_1$, $\underline{q}_1$) $\rightarrow \underline{v}_1 \rightarrow \underline{u}_1 \rightarrow$ ($\underline{w}_1$, $\underline{p}_1$) $\rightarrow$ bottom-up, winner and top-down $\underline{p}_1$ calculation). Then again, the vigilance test will be executed which will probably result in a match with the same output node and adjustment of weights to and from that node. This is repeated till F1 and the LTM stabilise according to the earlier mentioned criterion. At that point the input pattern is associated with the winning output node so the first match cycle ends. After setting all F0 and F1 nodes to $\underline{0}$, a new input can be applied.

Compared to the algorithm discussed in section 5.2, this algorithm differs on 3 important points:

1. The F0 layer is added to the architecture.
2. Patterns circle through the network a number of times during one match cycle. Before this match cycle ends, weights are adjusted repeatedly which means that numerous learning trials are executed before one match cycle ends.
3. The vigilance test compares $\underline{p}_1$ and $\underline{q}_0$ in stead of $\underline{p}$ and $\underline{u}$.

In the next sections these differences will be discussed. In these sections this version of ART 2 is not investigated as detailed as ART 2 version 1 in section 5.2. Simulations have only been done in order to get insight in the differences between both versions. One exception is made to this because it has to be tested whether the algorithm is capable of clustering a set of input patterns into stable recognition codes. Reproduction of the simulation which is shown in Fig. 4 in [Carpenter & Grossberg, 1987] showed, that the network indeed is capable of doing this.

5.3.1 F0, a preprocessor

If an input is applied to the network, it circles through F0, until the pattern stabilises within F0. From that point the pattern which is stored in $\underline{q}_0$ is considered to be the input to F1 and the algorithm starts trying to associate this pattern with patterns already stored into the network.
This shows, that F0 is nothing but a preprocessor. In Fig. 5.12 (a) and (b) is shown how it processes input patterns. Fig. 5.12 (a) shows, that F0 is nothing but a normaliser if $\theta = 0$. If $\theta \neq 0$, F0 can be seen as a feature extractor which accentuates maxima in input patterns (Fig. 5.12 (b)).

The fact that a preprocessor is added to the system, does not change the behaviour of the network. It only influences the patterns which are applied to the F1 layer. By adding the F0 layer to the network,

the F1 layers in both versions can operate on different patterns when the same patterns are applied to the network. This can lead to a different clustering.

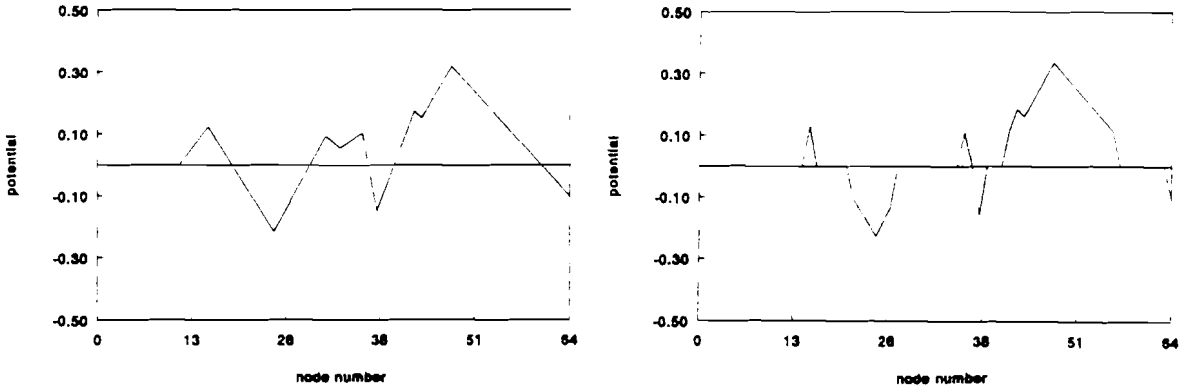

Fig. 5.12: Stabelised F0 output patterns if (a) θ = 0.0 and (b) θ = 0.1

5.3.2 Cycling of patterns within F1

Within ART 2 version 2 a match cycle ends if an input is stabilised in the network. This means, that if the input cycles through F1, the patterns in the different arrays of F1 and the prototypes in the vectors to and from the winning node change less then some predefined bounds. This yields, that during one match cycle a number of learning trials (weight adjustments) can occur. In section 5.2 has been concluded, that LTM stabilisation time depends on the number of learning trials. If every match cycle consists of only one learning trial (ART 2 version 1) the number of match cycles necessary before LTM stabilises is much larger then when match cycles consist of more then one learning trial (ART 2 version 2). Due to this, it looks if LTM stabilises quicker within this ART 2 algorithm. Simulations confirm this. Ep1_47 is repeatedly applied to a network which is defined by: a = b = 1.0, c = 0.1, d = 0.9, e = θ = ρ = 0.0, ac_dif = 0.001 and z_dif = 0.02. The pattern which is stored as the prototype of this input is studied after every match cycle. Comparison of the pattern stored after the first match cycle with the pattern stored finally into LTM shows, that both patterns are the same; the magnitude of the pattern after match cycle 1 equals 99.3% of its final magnitude, which shows, that LTM is almost stable after 1 match cycle. Simulations with θ = 0.1 show, that this is also valid if non linearities are incorporated into the network. In both cases the patterns stored into LTM are proportional to the pattern which is the input to F1, as can be seen if Fig. 5.13 is compared to Fig. 5.12.

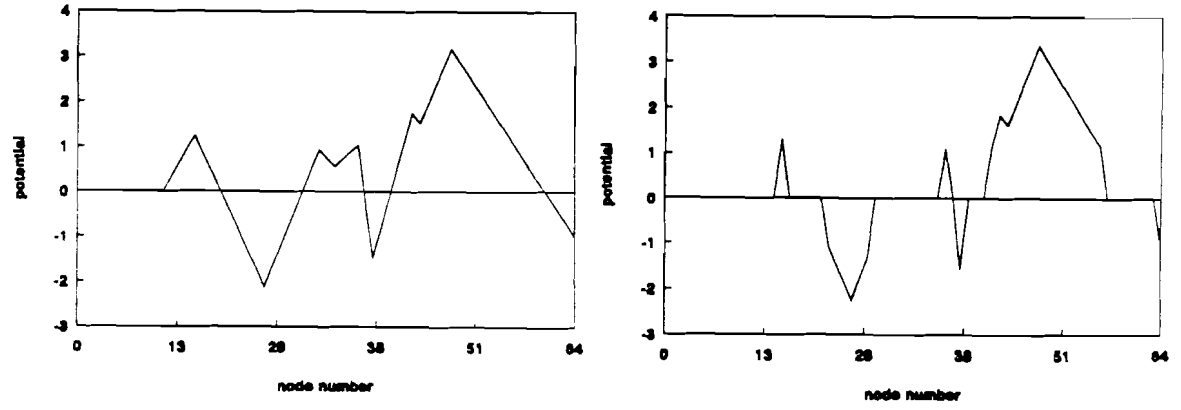

Fig. 5.13: Stable prototype in LTM if (a) θ = 0.0 and (b) θ = 0.1

The fact, that one match cycle can consist of more than one learning trial, is expected to cause problems if one wants to store the average of a number of (preprocessed) patterns as the prototype of one cluster.

Suppose, that two patterns **a** and **b** are applied to the network repeatedly, that ac_dif and z_dif initially are set to small values, that $\rho = 0.0$ and, that the bottom-up weights are chosen such, that both patterns will be associated with the same output node. If **a** is applied to the network, it will be associated with that node. Due to the small values of ac_dif and z_dif a number of learning trials is necessary to stabilise **a** into F1. Finally the net has stabilised, the association is made and the match cycle ends. Then **b** will be applied to the network. It will be assigned to the same output node. Due to $\rho = 0$ this will always be regarded as a fit so next **b** will be incorporated into the LTM to the same output node. Again it will take a number of learning trials before the match cycle ends. Summarised this means, that first the LTM is adjusted repeatedly in order to incorporate **a**. After that, **b** is incorporated into LTM repeatedly. From section 5.2 can be concluded, that this results in *forgetting* of pattern **a**: **a** will be forgotten if the number of learning trials within one match cycle increases, which can happen if ac_dif and z_dif are chosen very small.
Again, simulations confirm this. Fig. 5.14 shows what is stored in LTM after applying the patterns ep16_61 and ep17_61 25 times to the network in which $a = b = 1.0$, $c = 0.1$, $d = 0.9$, $e = \theta = \rho = 0.0$, ac_dif = 0.001 and z_dif = 0.02. If this figure is interpreted making use of Fig. 5.4 (a) and (b), which shows the patterns ep16_61 and ep17_61, it looks if ep17_61 is incorporated much stronger into LTM then ep16_61.

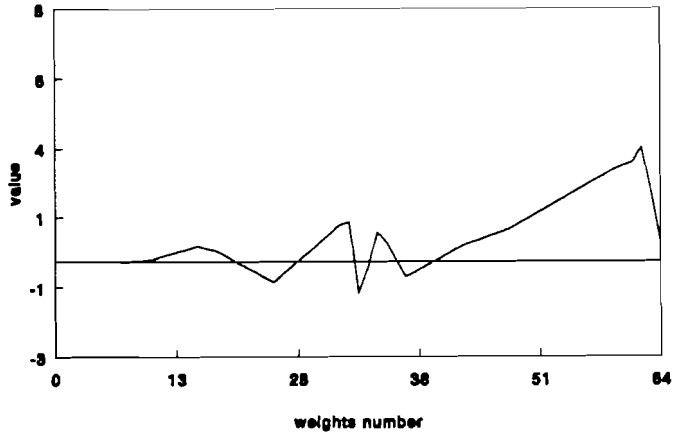

Fig. 5.14: "mixed" LTM prototype (d = 0.9)

Calculations prove that this is true. The patterns stored in LTM can be written as:

$$\underline{z} = c_1 \cdot \frac{\underline{a}}{|\underline{a}|} + c_2 \cdot \frac{\underline{b}}{|\underline{b}|} \qquad (23)$$

which shows, that the pattern stored into LTM is a linear combination of the normalised input patterns. For the pattern in Fig. 5.14 can be found: $c_1 = 1.04$ and $c_2 = 9.72$, which shows, that 90% of the LTM prototype is defined by **b** while only 10% is defined by **a**. This indeed shows that **a** is almost forgotten.

To overcome these problems 2 possible solutions have been studied.

The first possibility is, to increase d. In section 5.2 and Appendix 1a is shown, that if **a** and **b** are applied alternately and repeatedly to ART 2 network version 1 and will be associated with the same output node, this leads to a prototype which switches between two patterns which are "mirrored" round the average of **a** and **b**. When d is increased, these patterns resemble and therefore the average more and more. It will be expected that this also counts if **a** and **b** are alternately and repeatedly applied to ART 2 version 2. Simulations show that this is not true.
The simulation which lead to Fig. 5.14 has been done again but now d = 0.95. After 25 match cycles the prototype into LTM equals the pattern in Fig. 5.15.

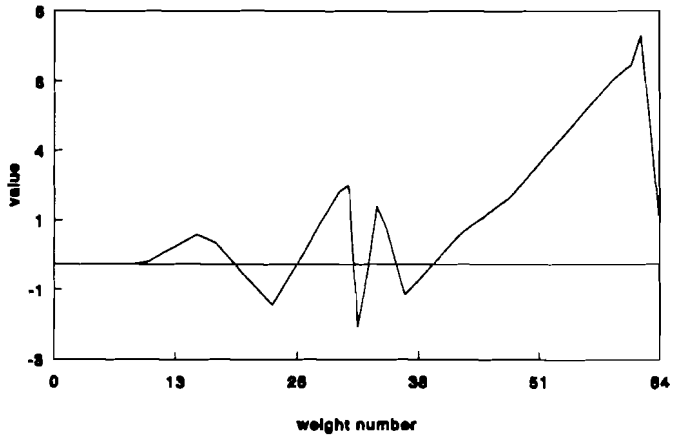

Fig. 5.15: "mixed" LTM prototype (d = 0.95)

Calculations show, that eq. (23) is also valid here. Now c_1 = 2.03 and c_2 = 19.45 so $c_1 : c_2 \approx 1 : 9$. These values show 2 things. First, that increase of d influences the magnitude of the LTM prototype. Second, that it has no influence on the ratio between patterns into LTM.
On second thoughts, this is not very astonishing. With small values for ac_dif and z_dif, a match cycle consists of more than one learning trial. Due to this, the conclusions of Appendix 1b which are mentioned in section 5.2.1 are not valid for ART 2 version 2. Learning in this version resembles the weight adjustments mentioned in Appendix 1c which shows, that in that case increase of d indeed leads to increasing magnitude but that increase of d has no effect on c_1/c_2. Increase of d can thus not overcome forgetting of $\underline{a}$.

The second possibility which has been studied to overcome this problem, is increase of ac_dif and z_dif. The same simulations have been done again but now with d = 0.9, ac_dif = 0.1 and z_dif = 0.5. As can be seen ac_dif and z_dif have been increased very much. Ep_16 and ep_17 have been applied alternately and repeatedly until LTM stabilised, which took almost 200 match cycles. The pattern stored into LTM after exactly 200 match cycles is shown in Fig. 5.16.

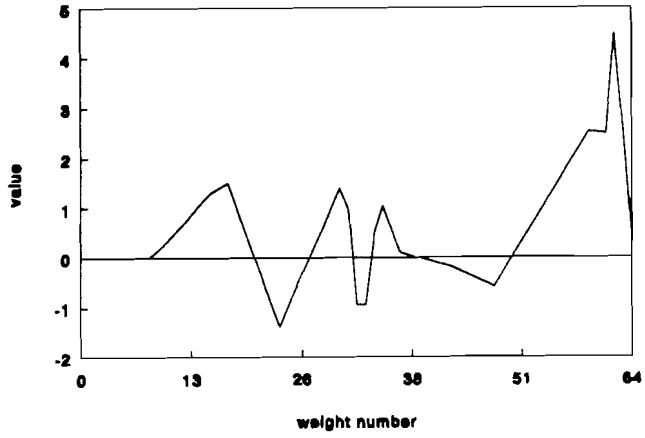

Fig. 5.16: stable "mixed" LTM prototype

For this pattern also c_1 and c_2 can be calculated: c_1 = 5.68 and c_2 = 5.77, which shows that LTM now looks much more like the average of two normalised patterns. This result is not amazing.

By increasing ac_dif and z_dif, bigger changes of $\Delta\underline{\mathbf{w}}_1 + \Delta\underline{\mathbf{p}}_1$ and $\Delta\underline{z}$ are accepted to conclude, that an input pattern has stabilised into F1. This results in fewer learning trials within one match cycle

which means, that if $\underline{\mathbf{a}}$ and $\underline{\mathbf{b}}$ are applied to the network alternately and repeatedly, the weights are also adjusted in order to incorporate $\underline{\mathbf{a}}$ and $\underline{\mathbf{b}}$ alternately more and more. Under those circumstances, the calculation in Appendix 1a becomes also valid for ART 2 version 2, which shows, that for high ac_dif and z_dif values ART 2 version 2 can also store average patterns. In that case forgetting of $\underline{\mathbf{a}}$ can be overcome if $\underline{\mathbf{a}}$ and $\underline{\mathbf{b}}$ are applied repeatedly to the network.
As stated before, the LTM convergence speed is guided by the number of learning trials. Due to the fact that one match cycle now consists of fewer learning trials, it is not difficult to understand that with high values for ac_dif and z_dif, it takes more match cycles to stabilise the LTM prototype. This is in agreement with the simulation.

Now it has been shown, that it is possible to store the average of a set patterns into LTM with this version of ART 2, if it operates under certain circumstances, this gives rise to an important question: how much do both versions resemble under those circumstances?

Suppose that ac_dif and z_ dif are chosen such that every match cycle only includes one learning trial. Due to $\underline{\mathbf{p}}_1 = \underline{\mathbf{0}}$ at the beginning of a match cycle, one match cycle executes the following calculations then:

$$\underline{w}_j \rightarrow \underline{x}_j \rightarrow \underline{v}_j \rightarrow \underline{u}_j \rightarrow \underline{p}_j \rightarrow botup \rightarrow winner \rightarrow (\underline{w}_j, \underline{p}_j) \rightarrow (\underline{x}_j, \underline{q}_j) \rightarrow \underline{v}_j \rightarrow \underline{u}_j \rightarrow vig.\ test$$

If this set of equations is compared to the algorithm discussed at the beginning of section 5.2, it can be concluded, that they resemble each other. There is only one difference between both versions: the interpretation of the vigilance test.
It has been mentioned earlier, that version 2 compares $\underline{\mathbf{p}}_1$ and $\underline{\mathbf{q}}_0$ instead of $\underline{\mathbf{p}}$ and $\underline{\mathbf{u}}$. Due to the fact that $\underline{\mathbf{q}}_0$ is normalised, the way these patterns are compared is exactly the same as the way which has been discussed in section 5.2.3. Due to the fact that $\underline{\mathbf{p}}_1$ can be related to the LTM prototype and $\underline{\mathbf{q}}_0$ to the input, this comparison becomes even easier to understand. Altogether this suggests, that the small difference between both vigilance tests does not effect the behaviour of the net; it only improves understanding.
In one case, this is unfortunately not true. That is the case in which a match cycle includes only one learning trial. Then indeed both algorithms are exactly the same but then the small difference between both vigilance tests has an important effect: the feedback loop from $\underline{\mathbf{p}}_1$ to $\underline{\mathbf{v}}_1$ looses its meaning. The calculations which take place after the top-down calculation of $\underline{\mathbf{p}}_1$ are not used to test whether the input fits the LTM prototype. This is the only time in which the feed back loop from the top F1 layer to the middle F1 layer is used so this feedback looses its meaning. This is an annoying property because this means, that because different pairs of patterns are compared within both vigilance tests, parts of F1 are ruled out if it is tried to choose the parameters such that both algorithms resemble each other. To avoid that parts of the network never will be used, it must be avoided, that all match cycles include only one learning trial. As a consequence of this, it is thus not possible to store average's of patterns into LTM with this version of ART 2.

5.3.3 Conclusions about ART 2 version 2

The algorithm which has been discussed in this chapter, has shown to be able to cluster patterns into stable recognition codes, if the input pattern set is applied to the network repeatedly. Compared to the algorithm discussed in section 5.2, it differs at 3 points. In this section it is clarified how these differences influence the behaviour of the network.

It has been shown, that adding of the F0 STM layer to the network is nothing but adding a preprocessor to the network, which extracts particular features from the input patterns. This preprocessor influences the input patterns which thus can influence the final clustering of the patterns. Applying different inputs to an algorithm does not influence the algorithm itself, so behaviour of the net remains the same with or without preprocessor. From the fact that patterns circle within F1 before F1 STM stabilises this can not be said. This influences the behaviour of the network very much due

to the fact, that the vigilance test compares a pattern within F0 with a pattern in F1. This has two consequences for the behaviour of the network. First, 2 new parameters (ac_dif and z_dif) are introduced in order to check whether patterns have stabilised within F1. Due to this, the clustering process is guided by 9 in stead of 7 parameters. Second, the network will never be able to store exact averages of patterns in its LTM without avoiding that parts of the network lose their meaning. Whether this is a problem is discussable now. ART 2 version 1 is able to store average patterns but this happens only if d = 1. In practice, d will never be chosen equal to one because LTM has to go to infinity in order to stabilise. This means that in practice within version 1 the last applied input will always be a little bit more present into LTM. If the number of learning trails within one match cycle in version 2 is kept small this will also be the case in this network. A suitable choice of ac_dif and z_dif can realise this. In that case the "cycling patterns" need not to be a problem.

Therefore this algorithm will also be used as a tool to survey a classification problem. The results of this survey will be treated in chapter 6.

6 How ART 2 clusters patterns

In this chapter will be discussed how ART 2 clusters BAEP patterns. First the BAEP classification problem will be discussed. Next will be clarified how stylistic BAEP patterns have been generated. After that an overview is given of the trial and error which has been done in order to get insight in the way ART 2 clusters stylistic BAEP patterns.

6.1 Introduction

In the previous chapter has been discussed, how ART 2 networks handle input patterns. It has been shown how the different parameters influence the patterns that are stored in the LTM and how these patterns are mixed. In this chapter will be discussed how both versions cluster patterns: which patterns will be grouped together and why and how can this grouping be influenced. Before studying this, one has to realise what the reasons for studying this subject are. These reasons can be very different. Some people are interested in neural networks because of what they are. These people wonder how networks can be developed in order to approximate the behaviour of the human brain and thus concentrate on networks which are expected to perform well in human like tasks like pattern and speech recognition. Others are interested in solving pattern recognition problems and regard neural networks as a possible tool. Their interest lays in the field of how a particular network can be adjusted in order to solve their pattern recognition problem.

As stated in the introduction (Chapter 1), this report describes a research project which is executed as a part of the *Aneasthetic Depth Project* of the Division of Medical Electrical Engineering of the Eindhoven University of Technology. In order to control the aneasthesia of a patient, one has to be able to measure *aneasthetic depth*. Research has shown, that there is a possible correlation between changes in the *Auditory Evoked Potential* (AEP) and variations in aneasthetic depth [Cluitmans, 1990]. Therefore, the AEP may serve as a possible monitor for aneasthetic depth. Because the exact relation between both is not known yet, one tries to extract physical measurable parameters from these AEP patterns which correlate with aneasthetic depth. If such parameters exist, this means, that if these parameters can be extracted automatically from the AEP, they can be used in a monitor for aneasthetic depth. This now shows, why neural networks are studied in the aneasthetic depth project, because neural networks are capable of extracting features from patterns. This also shows, from what point of view neural networks are used within this project: they are used as a possible tool to solve a pattern recognition problem: extracting features from AEP patterns.
For this report this is unfortunately not totally true. As stated in chapter 3, it has already been studied whether it is possible to extract one feature from an AEP pattern with a Multi Layer Perceptron and with a Counter Propagation Network. This report treats a third neural network: ART. Goal was indeed to investigate whether this network was also capable of extracting that feature from an AEP in order to compare the behaviour of different types of networks on the same pattern recognition problems. Due to the complexity of the network, this goal has unfortunately not yet been reached. Instead of solving the problem a lot of attention was paid to get insight in the network.

Altogether this means that, although we are interested in solving a pattern recognition problem, the network initially has been studied because of what it is: a neural network. This shows, that the various basic reasons for studying neural networks, mentioned at the beginning of this chapter, meet here. That they can meet in this research project, is not difficult to realise: when trying to solve the pattern recognition problem, the behaviour of the network is studied in order to get insight in the ART 2 algorithm. It is very important to remember this, when reading this chapter.

6.2 The classification problem

An AEP is the response of the central nervous system to an acoustic stimulus. This stimulus can be generated by applying "clicks" to the ear of a patient repeatedly. The AEP can be seen as the response

of the ear and the auditory pathway on these clicks. It is superimposed on the *electro encephalogram* (EEG) and can be extracted from this EEG by averaging periods of EEG activity measured after each of many stimuli. An example of such an AEP pattern can be seen in Fig. 6.1.

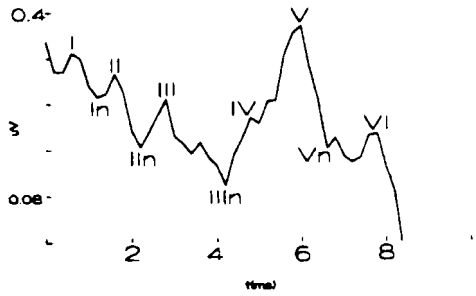

Fig. 6.1: Auditory Evoked Potential

In this figure the first 8 ms of the AEP are shown. Because this part of the AEP is generated in the brainstem by acoustic stimuli, this part is also denoted as the *Brainstem Auditory Evoked Potential* (BAEP).

The task of the neural network we are studying is to extract features from this BAEP pattern. Possible features which can be extracted are shown in Fig. 6.1: specific minima and maxima within the pattern. A possible feature is the local maximum top V. The pattern recognition problem thus can be: detection of the *latency* of top V which means, that the time between applying the acoustic stimulus and the occurrence of top V has to be recognised by the network. Suppose that an accuracy of 0.1 ms is desired then this means, that if a number of BAEP patterns will be applied to the ART 2 network, the network must cluster all patterns with a latency of top V which equals for example 5.4 ms together in one cluster in the ideal case. Even so all patterns for which this latency equals 5.5 ms etc.

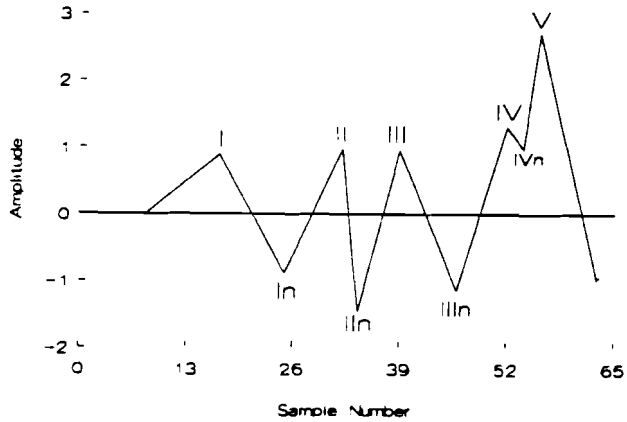

Fig. 6.2: Stylistic Brainstem Auditory Evoked Potential

It has already been discussed, that solving the pattern recognition problem and gaining insight in ART 2 networks was combined during the investigations described in this report. Because of the last interest, BAEP patterns are not used in this report, but *stylistic* BAEP patterns are used. An example of such a stylistic BAEP is shown in Fig. 6.2. Of course this stylistic BAEP resembles the measured BAEP to some extent. Thanks to research which has been done on BAEP patterns, it is known how the latencies of the maxima are distributed [Grundy et al., 1982]. Results of this research are shown in Table 6.1.

Component	Latency(ms)*
I	1.62 0.12
II	2.80 0.19
III	3.75 0.17
IV	4.89 0.23
V	5.62 0.23
VI	7.14 0.29

Table 6.1: Average and standard deviation maxima in the BAEP

With these results, stylistic BAEP pattern have been generated. This has been done as follows.
For every BAEP the first 6.4 ms are simulated with 64 samples. If applied to a neural network each sample coincides with a node. First the latency of top V is determined by generating a random number according to a normal distribution with the average as mentioned in Table 6.1 and a standard deviation which equals 2.4 times the standard deviation in Table 6.1 (see also [Habraken, 1990] in which these patterns are also used). This is repeated for IV, III, II, I and VI, of course with their own average and standard deviation. While doing this it is taken into account that first I appears, then II, III etc. Top V is forced to be incorporated within the first 6.4 ms.
After generating the latencies of the maxima, the latencies of the minima (In, IIn, IIIn, IVn and Vn) are generated. This is done by selecting a node randomly which is normally distributed around the average of the latencies of I and II, II and III etc.
Next, the amplitudes of the different components have to be determined. These amplitudes are also normally distributed. For the maxima I, II, III, IV and VI, the average equals +1.2. For the minima In, IIn, IIIn and Vn the average equals -1.2. The standard deviation for all amplitudes equals 0.5.
After this stage, all components have been placed except IVn and V, because the amplitude of these two extrema has not been established yet. With exception of these 2 points all other points can be connected with straight lines (I with In, In with II etc.). Placement of IVn and V is slightly complicated, because top V is the feature which has to be extracted and the stylistic BAEP pattern must resemble real BAEP patterns as good as possible near this top. First will be decided whether IVn has to cause a "dip" in the BAEP. This will be decided by comparing a randomly generated number with a predefined bound. If this check results in the decision, that no dip must be introduced within the pattern, a positive valued tangent is generated. A line is interpolated from IV with this tangent till the latency of V is reached. The amplitude of this line at that point becomes the amplitude of V. V is therefore known and can thus be connected to Vn. The BAEP is now almost ready. Of course has been taken into account while generating the tangent, that the latency of V will never exceed the latency of Vn.
If it has been decided, that a dip has to be incorporated within the pattern, the applied strategy becomes a little bit different. Then again a tangent is generated but its value becomes negative now. From IV a line is interpolated till the latency of IVn with this tangent. This results in placement of IVn. Next, the amplitude of V is determined by generating a random number according to a normal distribution with average 2.2 and standard deviation of 0.95. This results in assignment of V. If this point is connected with IVn and Vn (Of course taking into account the sequence of extrema), this BAEP is also almost ready.
In both cases 2 things have still to been done. First it has to be determined from what node the value of the BAEP becomes nonzero. This will be determined by generating a random number, normal distributed with average 9.5 and standard deviation 1. The outcome of this denotes the node from which the pattern becomes nonzero. This node will be connected with I. All nodes before this node get value 0. At the end of the pattern something similar must happen. There precautions must be taken in order to overcome, that not all nodes are used.

For these stylistic patterns, the classification problem (determination of top V) reduces to a simple clustering problem: all patterns with the same latency for top V have to be grouped together in the ideal case. In that case clusters arise which have each one feature in common: top V at output node 53, top V at output node 54 etc. A neural network which is capable in arranging this clustering is capable in detection of top V. Whether this can be reached with the 2 different ART 2 versions of chapter 5 will be treated in the sections 6.3 and 6.4.

6.3 Clustering patterns with ART 2 version 1

In order to study the clustering of ART 2 version 1, a set of 25 stylistic BAEP patterns was generated according to the strategy which has been sketched in section 6.2. The feature which will be studied in this chapter will be the latency of top V. Within the pattern set this latency varies from 4.7 ms to 6.1 ms (node 47 to 61).

Studying the clustering of this pattern set, this set has to be applied repeatedly to the network in order to reach a stable clustering. This can be regarded as training of the network. The clustering which is the result of this training depends on the choice of all parameter values which guide the training. Studying the stable clustering after each training, gives rise to 2 important questions:

1. What is the criterion to group patterns into a particular cluster?
2. How can this criterion, and thus the clustering, be influenced by the different parameters in order to extract the latency of top V from an input pattern?

The answer to these questions is not simple. This is mainly due to the fact that the network is governed by 7 parameters which means that 7 variables can influence the clustering process. Every variable has to be set to a suitable value when the network is initialised. It is not necessary to mention, that it is virtually impossible to find the optimal parameter settings.

It thus may be interesting to do some tests in order to get insight in the meaning of the different parameters in relation to the clustering of patterns. Therefore, a lot of tests have been done in which 6 parameters are varied. For simplicity normalising parameter e was set to 0.0 during all simulations. Every test lead to a different clustering. An example of such a clustering is shown in Table 6.2 in which is shown which top V latencies are grouped together within one cluster. The numbers within this table denote the nodes at which this top occurs (e.g. 57 means top V at 5.7 ms).

a = 1.0; b = 1.0; c = 0.1; d = 0.75; e = 0.0; θ = 0.00; ρ = 0.98			
cluster	0	1	2
latency top V	47 52 53 54 56 52 53 53	56 57 59 60 61 57 60 61 57 60 61 61 61	52 54 55 52

Table 6.2: Stable clustering of an input set

From the various tests a number of properties of the network become clear.

Most parameter settings indeed lead to a *stable clustering* if the training set is applied repeatedly to the network. The stable clustering is preceded by a number of cycles in which the network *searches* through output nodes. After a while, a stable categorisation arises and the network has *direct access* to output nodes.

Sometimes, a stable clustering with direct accessible output nodes is not reached. It then can happen that, although the clustering is stable, search is necessary in order to come to an association. In other tests it happens, that inputs *oscillate* between prototypes.

When studying the effect of the parameters on the clustering, a solution was found which overcomes the problem of unstable clustering. By varying the parameters a and b, a stable clustering can be achieved. On such a stable clustering, the effect of the other parameters can be investigated. From the tests that have been performed can be concluded, that both c, θ and ρ influence the number of clusters. The effect of vigilance parameter ρ on the number of clusters is very clear: increase of ρ leads to an increasing number of clusters. For the proportional constant c and non linear threshold θ this effect is less clear, the effect of learning rate d is also indistinct. Interpretation of the simulation results also leads to another very important conclusion: sometimes the network clusters the input patterns in a very reasonable grouping. The clustering of Table 6.2 is an example of such a clustering.

Of course it interesting to know, how parameters influence the number of clusters but that is not what we are interested in. We are interested in which patterns are clustered together and why and how this can be influenced by changing the parameters. In order to get insights in these aspects, new simulations were performed, in which attention was focused on the parameters θ and ρ. In the next sections the results of these simulations will be discussed.

6.3.1 The clustering criterion

As can be seen in eq. (11) and eq. (12) in chapter 5, ρ is one of the 2 parameters which guide the vigilance test. There it has been concluded, that for a particular value of c, ρ defines the "range of vision" in which a fit is accepted. If ρ increases, the range of vision becomes more narrow which means that patterns must resemble a prototype more and more in order to be associated with that prototype. This leads to an increase of clusters. In section 6.3, it has been shown, that ρ indeed has this effect on the number of clusters but, when looking at Table 6.2, this property also gives rise to a new question: how has ρ to be chosen in order to be able to distinguish between patterns? In Table 6.2 there exist 3 clusters if $\rho = 0.98$, that divide the input set reasonably. In order to solve to classification problem it is desired, that every possible top V latency can be assigned to its own cluster. Because the position of top V varies from node 47 to node 61 in this input set this means, that at least 15 clusters are needed. It is not hard to imagine, that it is impossible to succeed in this by only changing ρ to a suitable value because an increase of clusters implies an increase of ρ. Since the maximum value of ρ equals 1, this means that if the problem can be solved by only varying ρ the optimal value lies within a range of 0.02 [0.98-1.00]. Within that range the number of clusters has to increase to at least 15. The value of ρ thus is very critical.

Due to this critical choice it is probably impossible to distinguish between input patterns with the parameter settings of Table 6.2, which means, that these settings have to change. For some parameters it is not difficult to predict in which direction they have to change. That direction has to improve the possibility to distinguish between input patterns. The patterns have to be *contrast enhanced*, which means that features which have to be distinguished have to become more clear.
In chapter 5 already was concluded, that the network is able to amplify maxima in the network if θ will be increased. For our classification problem, this can be a nice property because this means that also top V will be amplified if θ increases. The feature that we want to detect will be enhanced in this way. This means that patterns in which this feature differs, resemble less if θ increases and can thus better be distinguished. For ρ this yields, that its value becomes less critical if θ increases, which probably means, that for the same value of ρ as in Table 6.2 more clusters arise in simulations in which θ has a larger value than in Table 6.2. Simulations show, that this expectation is indeed justified. In Table 6.3, the final stable clustering is shown of a simulation in which (compared to the simulation of Table 6.2) only θ has been changed from 0.0 to 0.2. Table 6.3 shows, that the number of clusters has indeed increased.

a = 1.0; b = 1.0; c = 0.1; d = 0.75; e = 0.0 ; θ = 0.2; ρ = 0.98								
cluster	0	1	2	3	4	5	6	7
latency top V	59 60 61 60 61 60 61 61	54 61	52 53 54 55 56 57 57	52 52	47 53	56	52	53 57

Table 6.3: Stable clustering of the input set

The way the patterns are clustered within this test gives rise to new questions: how can it happen, that some top V latencies are grouped together very reasonable while others are incorporated in different clusters (latency = 5.2 ms in 3 different clusters) and how can it happen, that the clustering becomes worse if θ is increased. In order to answer these questions the shapes of the BAEP patterns must be taken into account. Then the answers become very clear immediately: if top V is the global maximum of the pattern this pattern will likely to be clustered reasonable. If top V is not the global maximum, clustering goes wrong. This shows, that apparently the position of the global maximum is a very important criterion to cluster patterns with an ART 2 network. Perhaps this is not a general conclusion but for patterns which fluctuate so much as our BAEP patterns this statement is certainly valid. It is now easy to show why clustering goes wrong for the 4 patterns with a top V latency of 5.2 ms because for 3 of these patterns top V is not the global maximum. Due to the non linear function and the fact that θ = 0.2 other maxima than top V are more enhanced in these patterns and therefore the patterns are assigned to different clusters.
In order to solve the classification problem, problems like this have to be overcome because due to the fact that top V is not the global maximum and the network focusses on the global maximum, it can be predicted which pattern will be clustered wrong. There are 2 solutions to overcome this. The first one is, to change the criterion to which ART 2 clusters patterns to a criterion which clusters the patterns better in relation with the classification problem. The second solution is to preprocess the patterns in such a way that top V becomes the global maximum and thus can be distinguished with the ART 2 network.
It may be clear, that the second solution has to be chosen in order to investigate whether it is possible to solve the classification problem with this neural network. If the first solution would be chosen, this would probably change the network so much, that it can not be regarded as an ART 2 network any more. If this choice would lead to the solution of the classification problem, it does not fall within the scope of this research project (top V detection with an ART 2 network) any more. The solution which is investigated is preprocessing of the input patterns. This solution will be discussed in the next section, which is the last section in which ART 2 version 1 will be discussed.

6.3.2 Preprocessing of the input patterns

Preprocessing of BAEP patterns has to effectuate, that top V becomes the global or absolute maximum of the pattern if it is not the global maximum yet. This can be reached (not guaranteed !!) if those parts of the pattern which are likely to contain top V are amplified one way or the other, while those parts which can not contain top V are attenuated. This can be seen as a kind of filtering in which the pattern is multiplicated with a window which has the same width as the pattern. In this window it is defined how much the value of a particular input pattern node has to be amplified or attenuated.

In this section a very simple window will be treated, which equals 0 at those nodes which can not contain top V, and equals 1 at all the other nodes. This means that after preprocessing those parts of the pattern remain present at which top V can probably occur. It is not necessary to explain that it must be prevented, that it can ever happen, that top V does not pass the preprocessing stage.
After examining a lot of patterns it has been concluded, that if the first 41 nodes are set to 0, it will always be guaranteed, that top V will be passed. The effect of this kind of preprocessing is very clear

now: it removes all global maxima which lie in the range [0..4.1 ms]. This increases the probability that top V becomes the global maximum. That this can not be guaranteed is shown in Fig. 6.3 in which pattern ep7_52 is shown before and after preprocessing. Fig. 6.3 (b) shows that after preprocessing top V is still not the global maximum.

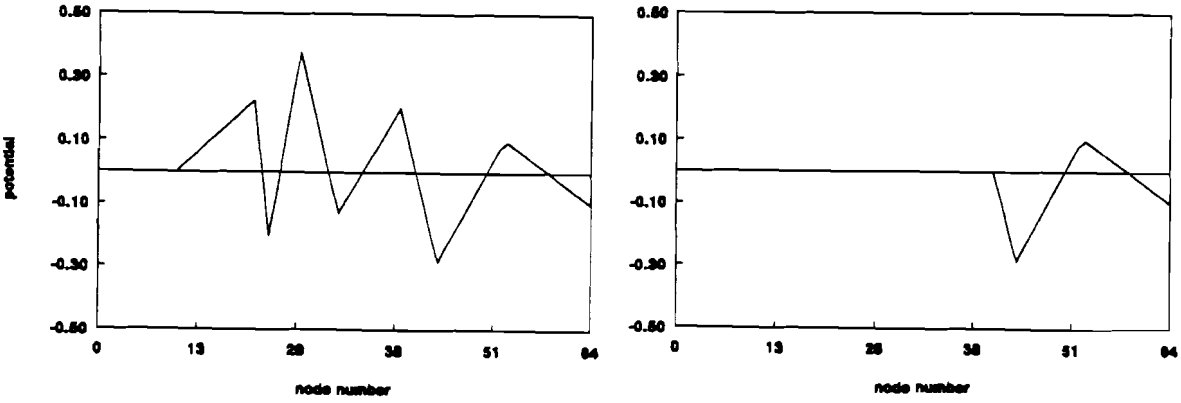

Fig. 6.3: ep7_52 (a) before and (b) after preprocessing

For this pattern the clustering will not improve, but investigation of the other 3 patterns showed, that top V has become the absolute maximum so for these patterns clustering will improve. Due to this it is assumed, that the classification problem perhaps can not be solved exactly if the input patterns are preprocessed but, that the clustering process certainly can be improved. The results shown in Table 6.4 illustrate this. In this table the final clustering of a clustering process is shown in which the same parameter settings are used as in the simulation which leads to the clustering of Table 6.3. Now the input patterns have been preprocessed according to the above described method.

a = 1.0; b = 1.0; c = 0.1; d = 0.75; e = 0.0; $\theta = 0.2$; $\rho = 0.98$				
cluster	0	1	2	3
latency top V	52 53 54 55 56 57 52 53 56 52 53	52 54 61	47	57 59 60 61 57 60 61 60 61 61

Table 6.4: Stable clustering of preprocessed input patterns

If both clusterings are compared, two properties attract attention. First it is shown, that preprocessing of the input patterns indeed has the desired effect on the clustering. The simulation unfortunately also shows a second property: it is again more difficult to distinguish between patterns with by chosing an appropriate value for ρ. This last property can be a problem because in this simulation θ already has a large value. While in Table 6.2 the problem of a large value for ρ and only a few clusters could be overcome by increasing θ, this solution can not be used here because θ can not become too large. It thus looks like that we are back where we started: a small number of clusters with a large value for ρ. The choice of this window as a preprocessor therefore does not look a good solution in order to solve the classification problem with ART 2 version 1, although from this result can be concluded that preprocessing causes an important effect: the input patterns are transformed to patterns in which the feature which we have to extract fits the criterion which ART 2 uses to distinguish between patterns. In this way we thus have succeeded in "fitting" a classification problem to a network. Although there are still a lot of problems which have to be solved and the clustering is still very coarse this already is discarded as an important step in the direction of solving the classification problem with an ART 2 network.

6.4 Clustering patterns with ART 2 version 2

In section 6.3 it has been shown, that ART 2 version 1 (ART 2-1) is able to cluster patterns into a stable clustering according to a criterion in which the position of the global maximum of the patterns is very important. It has been shown, that a few problems arise when this network has to be used to solve the classification problem. The most important problem is, that it is difficult to distinguish between patterns with ρ. This can be overcome by increasing the threshold θ which increases the effect of the non linearity on the input pattern increases. Problem then is, that clustering becomes worse. The reason for this is, that top V is not always the global maximum of the input pattern. A solution to this is to make use of preprocessing. Then some parts of the pattern are filtered which indeed leads to the effect that clustering improves but then also the problem of distinguishing between patterns with a proper value of ρ comes back.

6.4.1 Simulation results

When studying ART 2 version 2 (ART 2-2), attention was focused on the question whether the same problems occur as in ART 2-1. First, some general simulations have been done in order to check whether the effect of the different parameters on the number of clusters and the patterns within these clusters is the same as in ART 2-1. For these simulations a number of training sets with BAEP patterns were used. With every set a number of tests has been done with different parameter settings. In every test the training set has been applied to the network repeatedly which led to a stable clustering in most cases: in some tests input patterns oscillate between clusters. As in ART 2-1 this can be overcome by changing the parameters a or b. It then attracts attention, that it takes considerably more time to do simulations if the same training set is applied the same number of times to ART 2-2 as to ART 2-1. Another property which attracts attention is the number of clusters in relation to the value of ρ. Even if $\rho = 0$, which yields that a RESET can never occur, the training set is divided into a number of clusters. This suggests, that one way or the other ART 2-2 can distinguish between patterns better with ρ than ART 2-1. The simulations confirm this: with ART 2-2 the number of clusters considerably increases if ρ varies between 0.8 and 0.9. This shows that in ART 2-2 the choice of ρ is less critical in order to distinguish between patterns and, that the problems which arise with ART 2-1 probably will not occur with ART 2-2. The effect, that the number of clusters increases with an increase of ρ is in agreement with ART 2-1. This also counts for the behaviour of c and θ. For this last parameter can be concluded, that it improves the clustering slightly. Clustering within ART 2-2 with regard to the latency of top V is very bad during most simulations.

From all this a number of conclusions can be drawn. First, it can be concluded, that the direct effect of the parameters on the number of clusters is the same as in ART 2-1. Increase of ρ still leads to an increase of clusters whereas decrease of θ diminishes the feature extraction within the pattern. The fact that simulations with ART 2-2 take more time than simulations with ART 2-1 if the same training set is applied as often, needs some explanation because this suggests, that it takes more time to reach a stable clustering. It is expected, that this is not true. In section 5.3 has been shown, that it takes more time to make one association with ART 2-2 than with ART 2-1. This is due to the fact that a match cycle contains only one learning trail in ART 2-1 and a match cycle contains a number of learning trials in ART 2-2. This yields, that if a training set is applied to ART 2-2 as often as to ART 2-1, it takes more time before one match cycle ends because patterns circle within F1. This explains why it takes more time to do the same simulation with ART 2-2 compared to ART 2-1.
In section 5.3 it has also been shown, that fewer match cycles are necessary to reach a stable clustering. Simulations with ART 2-2 show this property also. It is now expected, that the effects of fewer match cycles but more learning trials counterbalance each other which then results in the same stabilisation time for the training set. Whether this is indeed true needs to be checked.

The fact that patterns circle within F1 has an important influence on the sensitivity of ρ in relation to the number of clusters. It has been shown, that the choice of the value of ρ is less critical in ART 2-2. Reason for this is, that prototypes within ART 2-2's LTM have already reached a reasonable part

of their final magnitude after only one match cycle. Because of this it is much more clear that a pattern and a prototype do not fit if they do not resemble. Patterns and prototypes therefore can be distinguished much better which means, that the value of ρ can be chosen less critical; a small change does not result in a different clustering immediately. This now is a very nice conclusion because this means that if contrast enhancement with θ is used in combination with pattern preprocessing, the problems which occur with ART 2-1 do not arise. Knowing this, a start can be made in order to find a solution to the classification problem. This start is not very difficult to make because a lot of knowledge is present about the network now. The next two sections discuss how an acceptable clustering is found and how this clustering behaves if a test set which contains patterns that have not been used to train the network is applied on a trained network.

6.4.2 Towards an acceptable clustering

The first simulations with ART 2-2 which have been discussed sofar, show a bad clustering with regard to the top V latency. The reason for this will be clear: the network distinguishes according to a criterion which separates between the global maximum of the input patterns while the classification problem requires, that particular local maxima can be distinguished. In section 6.3 preprocessing showed to be a solution to improve clustering if, due to this preprocessing, top V becomes the absolute maximum. If not, it can not be guaranteed that the clustering will be correct.

This conclusion is now used as a starting point to solve the classification problem from a different point of view. This point of view is as follows: if the latency of top V has to be extracted from a pattern with an ART 2 network and (due to preprocessing) top V has been related to the absolute maximum of the pattern, it can be of interest to investigate whether the network is capable in distinguishing between top V latencies. If it is possible to come to a good clustering with these patterns this means, that in order to solve the classification problem a suitable preprocessor has to be found which transforms top V latencies into global maxima with high probability. If it is impossible to come to a reasonable clustering, this almost yields, that it will be rather difficult to solve this classification problem with ART 2.

This means, that it can be very interesting to do simulations with a training set which consists of patterns in which top V is the global maximum. Therefore a test has been performed with 100 stylistic BAEP patterns. This test will be referred to as TRAINSET1. From this set all patterns are removed in which top V is not the global maximum. Then 51 patterns are left. These patterns can be seen as 64 dimensional vectors. These "vectors" will be normalised so, that their length becomes equal to 1. These 51 normalised patterns will be used to train the network. Therefore they have been applied to the network repeatedly. After trial and error, parameter settings have been found which cluster the set into a reasonable clustering. This clustering is shown in Table 6.5.

a = 1.0; b = 1.0; c = 0.1; d = 0.90; e = 0.0 ; θ = 0.25; ρ = 0.0; ac_dif = 0.001; z_dif = 0.02						
cluster	0	1	2	3	4	5
latency top V	52 53 54 55 56 53 55 55 55	59 60 61 62 63 59 60 62 60 62 60	57 58 59 60 57 58 59 57 58 59 57 58 57 58 58 58	47 48 49	49 49 49	50 51 53 50 51 53 51 53 51

Table 6.5: Reasonable clustering of patterns for which top V is the global maximum

In Table 6.5 it is shown, that the input set is divided into 6 clusters which partially overlap. Every cluster contains patterns of at most 5 different latencies which do not differ much. Because $\rho = 0$, this means that the network comes to this clustering "automatically"; a RESET will never be generated.

Of course this clustering has to be improved in order to speak about an acceptable clustering. A possible solution has shown to be an increase of θ. This possibility does not work here. Because the patterns are normalised, the maximum value within a pattern (top V), is near 0.30. It must be prevented, that θ exceeds this value because the pattern will not pass the non linearity then and clustering will go wrong. Simulations have shown, that $\theta = 0.25$ is a safe maximum value.
An other solution, is to increase ρ. Then it is indeed reached, that the 6 clusters are divided into more narrow clusters so the number of clusters indeed increases. Unfortunately this division is not such , that the clustering improves considerably: the clusters become indeed more narrow but this is accompanied by an increasing number of clusters which store only one pattern. This process shows, that in general it is indeed true, that patterns with latencies which are near to each other look alike very much while patterns with more different latencies resemble less. It also shows, that it often happens, that 2 patterns with the same top V latency resemble less than two patterns with a top V latency that differs 0.1 ms.

In order to overcome problems of this kind, the maxima within the training set have to be emphasised even more. A possible solution that realises this very strongly is to apply an exponential function on every input pattern. This can be seen as an extra preprocessing stage. To test the effect of this extra preprocessing stage on the clustering process TRAINSET1 is used. Again all patterns are removed for which top V is not the global maximum. The 51 patterns which are left then undergo an exponential operation. This operation yields that the value of every node within each patterns is recalculated according to:

$$x_i^{(new)} = \frac{x_i^{(old)}}{|x_i^{(old)}|} \cdot \exp(|x_i^{(old)}| - 1) \qquad (1)$$

After that every pattern will be normalised again.

With the set which arises then, tests have been done. After some trial and error an acceptable clustering has been found. This clustering is shown in Table 6.6. Since it contains the same patterns as the clustering described in Table 6.5, both tables can be compared and conclusions can be drawn about the influence of this extra preprocessing stage on the clustering.

a = 1.0; b = 1.0; c = 0.1; d = 0.75; e = 0.0 ; θ = 0.25; ρ = 0.9; ac_dif = 0.001; z_dif = 0.02													
cluster	0	1	2	3	4	5	6	7	8	9	10	11	12
	49	60	52 53 54	57	51 53	55 56	50 51	61 62 63	59 60	47 49	59	57 58	48
	49	60		57	53	55	50 51	62	59 60		59	57 58	
latency	49	60			53	55	51	62			59	57 58	
top V					53	55						58	
												58	
												58	

Table 6.6: Stable clustering if an extra preprocessing stage is applied

The influence of this extra preprocessing stage becomes very clear: Table 6.6 shows that the clustering has considerably improved. In this table input patterns are clustered into a stable clustering in which the deviation of top V latencies is at most 0.1 ms. For the time being this is acceptable.
From this now a very important conclusion can be drawn which yields, that ART 2 networks show to be capable ofclustering BAEP patterns into an acceptable clustering if top V is the global maximum of the input pattern. In the beginning of this section has been stated, that if ART 2 was able to do this,

the stylistic BAEP-classification problem (in which top V is not always the global maximum) could possibly be solved if a suitable preprocessor could be found. In that case the solution of the BAEP pattern recognition problem depends very much on the fact whether such a preprocessor can be found or not.

Investigation of this has to be subject of further research because this project has to come to an end. There is one property which has to be checked before finishing it. This property can be described as follows.

ART 2 is a neural network and as stated in the beginning of this report neural networks are able to generalise. This means, that if the network is trained with a training set in which some relation is stored and this training set is suitable chosen, then the network is able to generate an output which also satisfies this relation if an input is applied to it. This even has to count if that input is not a member of the training set. In that way the network has not just learned a number of patterns or a number of input/output combinations but it has extracted some general relations from the training set and stored into its weights.

Before finishing this project it has been investigated whether this property is present within ART 2. Therefore a number of tests have been done. These tests will be discussed in section 6.4.3.

6.4.3 Testing of a trained network

The way to test whether a neural network is capable of generalisation is as follows. It consists of 2 parts. First, a training set is applied to the network and the network is trained. For ART 2 this means that a stable clustering has to be reached which has to group the patterns of the training set into acceptable clusters. It is important to study this clustering very well because it has to be used as a reference in the second part of the test in which a test set is applied to the network. Every pattern of this testset is applied to the network only once. The neural network handles these patterns in almost the same way as the patterns of the training set, that is it associates every pattern with a cluster. The difference between training and testing the network is, that after an association is made while testing the network, the input is not incorporated within the weights of the network, learning does not take place. In this way testing yields nothing but looking how a trained network clusters a test set. This clustering now is very important because conclusions can be drawn from this clustering if the patterns of the test set are clustered in the same way as the patterns of the training set. In that case it is said, that the network has learned some general relation and can thus be used to apply this relation on an input pattern in order to perform pattern recognition.

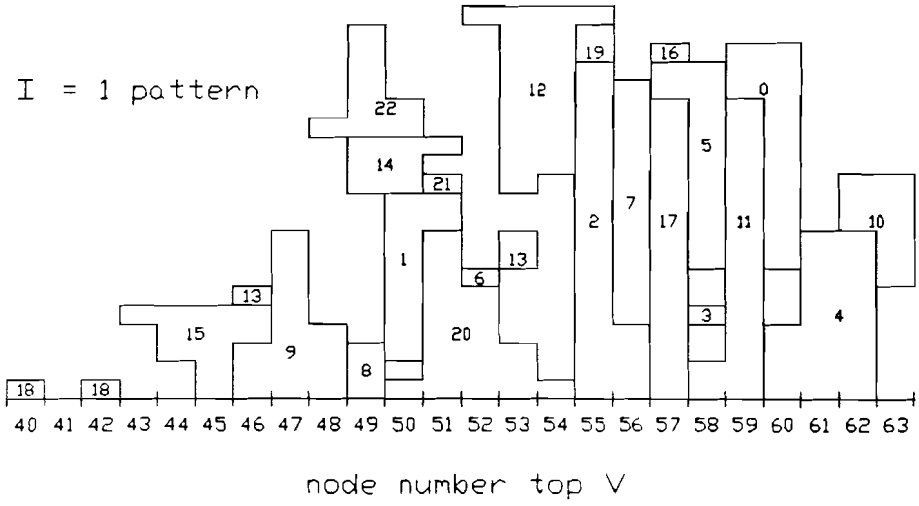

Fig. 6.4: Stable clustering of 240 preprocessed patterns (see text for explanation)

The training set which has been used to train the network is generated as follows. First a set is generated which consists of 400 stylistic BAEP patterns. From this set the patterns in which top V is not the global maximum are removed. On the 240 patterns which remain the exponential operation has been applied. After that, these patterns are normalised. This TRAINSET is then used to train the network. After some trial and error parameter settings have been found which lead to the acceptable stable clustering which is shown in Fig. 6.4.
In this figure every seperate numbered area denotes one cluster. On the x-axis is shown which top V latencies are incorporated in a particular cluster. The values at this axis denote the node number at which top V appears. The relation between these node numbers and top V latencies will be clear: node 53 means top V at 5.3 ms. In "y-direction" is shown how many patterns of a particular latency are assigned to a cluster. Cluster 7 for example has contains 13 patterns at which top V equals 5.6 ms. Cluster 18 consists of 2 patterns. One with top V at 4.0 ms and one with top V at 4.2 ms. From Fig. 6.4 can be concluded, that the clusters which contain patterns with a top V latency near 5.6 ms (node 56) are very narrow and contain mostly patterns of only one latency. This is not amazing because 5.62 ms is the average of the normal distribution with which these patterns are generated (Table 6.1). For the clusters which contain patterns of smaller or larger top V latency this is not true. These clusters contain patterns of more different latencies which are near to each other.

With this TRAINSET incorporated into LTM the network now can be tested. Therefore TESTSET1 is generated in exactly the same way as the TRAINSET. In that way new patterns arise which meet the same properties. This TESTSET1 contains 129 patterns. It has been applied to the network only once with the same parameter settings as the network has been trained with. The way this TESTSET1 is clustered is shown in Fig. 6.5.

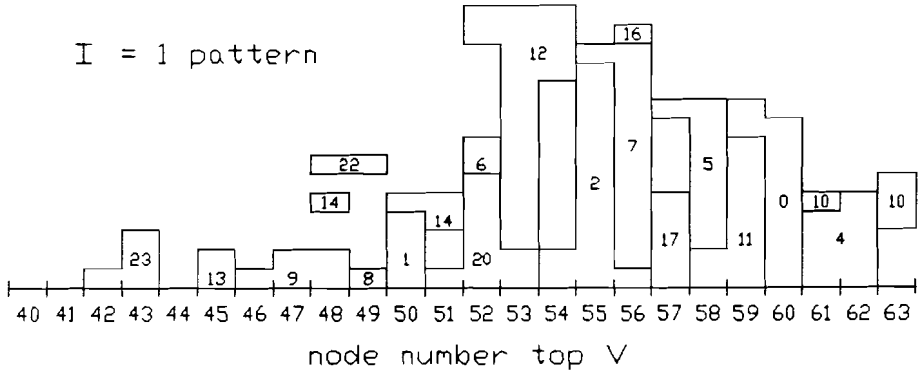

Fig. 6.5: Clustering of a TESTSET with a trained ART 2 network

If this figure is compared to Fig. 6.4 it can be concluded, that in both figures the stylistic BAEP patterns are distributed in the same manner. It is difficult to prove this mathematically because both pattern sets do not contain the same patterns but if it is said, that the areas which are identically numbered in Fig. 6.4 and Fig. 6.5 represent the same clusters, then it can be concluded, that every cluster indeed represents a (set of) pattern(s) with a particular latency and that the network thus has learned to attract some general features from the input patterns and uses these features as a means to cluster patterns. For the latencies near 5.7 ms the behaviour of the network is very good. There the network can distinguish between latencies which differ 0.1 ms. A last test illustrates this even more. For this test a new testset (TESTSET2) has been generated which consists of patterns in which the top V latency varies from 5.5 ms to 5.9 ms. This test has also been applied to the trained network with the same parameter settings as used during the training of the network. This set will then be clustered as shown in Fig. 6.6.

From this now can be concluded, that the clusters which represent patterns with top V latencies in the range [5.5..5.9 ms] are indeed capable of distinguishing these latencies with an accuracy of 0.1 ms with high probability.

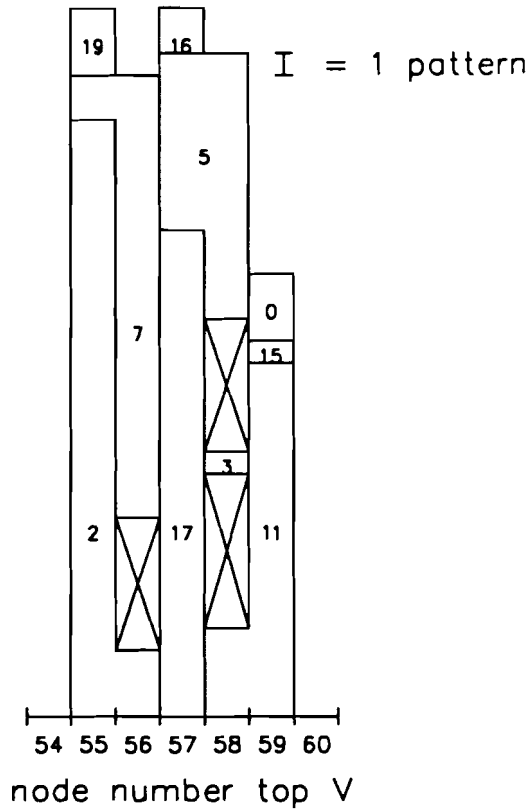

Fig. 6.6: Clustering of TESTSET2 with a trained ART 2 network

6.5 Conclusions

In this chapter it has been investigated how ART 2 networks cluster stylistic BAEP patterns. The reason for this was twofold. First, as a possibility to get insight in the behaviour of the ART 2 network. Second, to get insight in how a particular pattern recognition problem can be solved. This chapter has shown, that the ART 2 clusters stylistic BAEP patterns according to a clear criterion: the latency of the global maximum. It has been stated that this is perhaps not a general conclusion but that this is certainly true if patterns fluctuate so much as the stylistic BAEP patterns.
Due to this clear criterion it can be predicted, that if top V is not the global maximum clustering can go wrong. To overcome this problem some preprocessing operations are applied to the input patterns. These operations indeed improve the desired effect.

With regard to the different parameters and their influence on the clustering process a number of conclusions can be drawn. One of the most important conclusions is, that θ is a very important parameter because with θ the non linearity is incorporated into the network. It has been shown, that with increasing θ the contrast enhancement of patterns within the network increases.
For ρ can be concluded, that it is possible to distinguish between patterns by choosing a suitable value for this parameter. Increase of ρ leads to an increasing number of clusters. It is important to state here, that if ρ has to be used as a stable parameter it is necessary that ART 2-2 is used in order to overcome problems, because the value of ρ has to be chosen very critical in ART 2-1.

A last conclusion which has to be mentioned here, is that ART 2 showed to be capable in recognising patterns in a testset with a trained network if patterns in the testset meet the same properties as the patterns within the training set.

7 Conclusions and recommendations

The contents of this report can roughly be divided into 3 parts:

1. a general study of neural networks in which a number of networks has been discussed.
2. a study of both the ART 1 and the ART 2 network in which it is tried to clarify why ART networks are so complicated.
3. a study of how ART 2 networks handle BAEP patterns with which is tried to get insight in how the BAEP problem can possibly be solved by making use of an ART 2 network.

In this chapter conclusions and recommendations will be treated for each part separately.

7.1 Neural networks

In the first part of this report a number of neural networks has been treated. After comparing these neural networks, 3 networks where chosen that are used as a possible tool to solve the BAEP classification problem. These networks are:

- the Multi Layer Perceptron
- the Counter Propagation network
- the Adaptive Resonance Theory network

Each one of these networks is or will be evaluated for its use to solve the BAEP problem.

The way neural networks were treated here differs from the way neural networks are commonly treated in literature. As stated in chapter 2, a neural network consists of 4 "parts". In order to develop a neural network a choice has to be made for each part:

1. the neuron model
2. the network architecture
3. the training algorithm
4. the learning rule

In this report for every network these choices are clearly mentioned. It is avoided to come to some grouping of the networks which is encountered in literature. Because grouping is often based upon only one one part of the network this does not give insight in the choices of the three other parts: due to this, it is suggested, that networks have a lot in common when they only have one feature in common. This hampers a quick literature research.
Therefore, in this report a plea is made to mention only the 4 parts of a network and to avoid to come to any kind of grouping. If every scientist which works in the field of neural networks follows this plea, it is expected, that the field of neural networks becomes much more clear and that it is much easier to get a quick impression about neural networks and what they can do.

7.2 Adaptive Resonance Theory networks, their behaviour

In the second part of this research project it is tried to clarify how ART networks behave and why they are so complicated. In the treatment of this network a distinction is made between the ART network which handles binary valued input patterns (ART 1) and ART networks which handles analog valued inputs (ART 2). Both networks have shown to be able to cluster a set of input patterns into a stable grouping in which output nodes are direct accessible. This is done by self-organisation according to a clustering algorithm in which the neural network is critical to itself which means, that it will always "ask itself" whether a pattern and a prototype which are associated indeed fit in order to accept

that association. It is important to note, that this stable clustering could only be reached if the input set is applied to the network repeatedly.

When treating neural networks we remarked the fact, that human qualities are very often attached to these networks. In ART literature this happens very often. Later on in this section will be clarified that one must take care with this because human qualities can be interpreted in different ways.

7.2.1 The ART 1 network

When discussing ART 1, two interpretations have been treated. Of course the network which has been developed by Carpenter and Grossberg, but also the Lippmann interpretation of this ART 1 network was discussed. It was shown, that both networks indeed implement the same clustering algorithm, although a small difference within one of the learning rules can result in a different clustering if the same input set is applied to both networks. Because even then both clustering still satisfy the same criteria, both interpretations were considered to be similar. The fact that both interpretations differ so much is due to the fact that Carpenter and Grossberg look at the network from another point of view than Lippmann. Carpenter and Grossberg look at the network from a hard-ware point of view in which they use data to control the network, Lippmann leaves out all control tasks and simplifies the ART network to an algorithm which can easily be used to simulate a neural network.

With the ART 1 network an important property of this type of networks has been clarified: why are ART networks so complicated and need a combination of bottom up and top down calculations? In section 4 it was shown, that this is necessary in order to be able to cluster patterns into stable recognition codes by self organisation.

Simulations have shown that the stability-plasticity dilemma is not solved with the ART 1 network because it often happens that an input is assigned to a new cluster even if it has been incorporated to an other cluster before. In ART 1 this happens because fast learning is used. Later on in this chapter will be shown that according to our opinion ART networks do not solve the stability-plasticity at all while Carpenter and Grossberg state that this network solves the dilemma. This misunderstanding is probably due to a different interpretation of the word "forgetting".

7.2.2 The ART 2 network

The ART network which is able to handle analog input patterns is denoted as ART 2. It is important to note, that the ART 2 network is still subject of current research and that there are a number of different architectures. Therefore it is difficult to speak about "the" ART 2 architecture.
In this report two ART 2 interpretations have been discussed: ART 2-1, which is our interpretation of an ART 2 network. It is a network in which the clustering depends on the choice of 7 parameters and the initial values of the weights. ART 2-2 is the interpretation of an ART 2 network of 2 students of Carpenter and Grossberg. The clustering of patterns with that network depends of the choice of 9 parameters and the initial values of the weights.

For these networks some general conclusions can be drawn.

If ART 2 networks are compared to ART 1 networks it attracts attention, that the F1 layer is divided into 3 layers. It has been shown in this report, that a number of operations take place within this layer and that due to the cooperation between these operations some attractive properties are incorporated within the network. It is worthwhile to mention two properties here. The first one is the contrast enhancement which takes place within F1. This occurs due to the cooperation between the non-linear functions and the normalisation steps. A second property is the clear way in which patterns are compared due to the cooperation between the vigilance test a normalisation step in F1. These properties can be reached because F1 transforms patterns to other useful patterns. This shows, that F1

thus can be seen as a pattern processor and, that the cooperation of the different parts of the network with each other is very important in an ART 2 network.

For both interpretations also some separate conclusions can be drawn.

It has been shown for ART 2-1, that 4 of the 5 design principles mentioned in chapter 5 are satisfied. Only the stability-plasticity dilemma is, according to our opinion, not solved. This is a surprising conclusion because it is the second time this has been concluded while literature states that this dilemma is solved with the ART 2 network. A more interesting conclusion which corresponds better to literature, that simulations have shown and mathematics have proven, that it is possible to store averages of patterns as the prototype of a cluster if certain parameter settings are used.

With ART 2-2 it has been shown, that this property can also be achieved although this has important consequences for the choices of the values of the two parameters which discriminate ART 2-2 from ART 2-1. It has been shown, that ART 2-2 resembles ART 2-1 very much if this property has been achieved. The only difference between both interpretations then is, that ART 2-2 has an extra STM layer F0. Simulations showed, that this layer can be regarded as a preprocessor.
The two parameters mentioned above can also be chosen such, that ART 2-2 does not store averages of patterns as a prototype of its clusters. In that case it has been shown for the third time, that an ART network does not solve the stability plasticity dilemma because the prototype always resembles the last associated input more than earlier associated input patterns. In some cases it even happened that earlier applied inputs are hardly stored in LTM any more. Because this is not the first time it has been shown, that the dilemma is not solved, it is worthwhile to focus on this dilemma for a while before starting with the conclusions about how ART 2 can be used in order to solve the classification problem.

7.2.3 The stability-plasticity dilemma

Although Carpenter and Grossberg state, that the stability-plasticity dilemma is solved with an ART network, our simulations show the contrary. This gives rise to a number of questions: are we doing something wrong? Is what Carpenter and Grossberg do wrong? or do we misunderstand each other? The last possibility seems the most logic one, which thus means that we must try to understand what Carpenter and Grossberg mean with this dilemma.

If ART networks are compared to other unsupervised neural networks there is indeed one case in which ART 2 does not wash away previous learning while other neural networks do. Suppose, that there is an unsupervised neural network whose weights are all set due to learning. If an input will be applied to this network it will be associated with the output node which resembles the input most. Because these neural nets have no fit criteria an association will always be made. If the input does not resemble any pattern at all the best fit can better be called the least worse fit. In that case an input is incorporated into weights which do not resemble that input. Previous learned information will thus be washed out.
In ART networks this can never happen due to the fact that ART is critical to itself and is always able to generate a new cluster if the input does not fit any prototype. In this case past learning is indeed protected from present learning. Simulations unfortunately have shown, that information is washed out. This happens if an association is made. In ART 2 networks patterns do fit which thus means, that they have something in common. The fact that previous learning will be forgotten a little bit is then not so dramatically because the old prototype will be replaced by a new prototype which resembles the old prototype so on balance it does not change much.

If the above is what Carpenter and Grossberg mean with the stability-plasticity dilemma the network indeed solves the dilemma. This then is unfortunately a little bit disappointing because the way they talk about the dilemma in literature as that ART networks do forget nothing. This is certainly not true.

7.3 Clustering stylistic BAEP patterns with ART 2

The third part of this report discussed how ART 2 networks cluster patterns, according to which criterion and how. This has been done with stylistic BAEP patterns which yields, that simultaneously it was possible to get insight in how the stylistic BAEP pattern recognition problem could possibly be solved.

The most important conclusion which can be drawn from this part of the report is, that if input patterns are used which fluctuate so much as the stylistic BAEP patterns that have been used, the criterion according to which ART 2 clusters patterns becomes very clear: it groups patterns together for which the latencies of the absolute maximum look alike. How much patterns have to look alike in order to be associated is defined by the vigilance test. The way patterns are compared within this test has become also very clear. In this test an area is spanned in M dimensional space in the "direction" of the prototype. Those patterns which lie within this area at distance 1 from the origin will be associated with the prototype if they are applied to the network. The input patterns which lie outside this area will not be associated. The size of the area can vary because it depends on the value of ρ, the vigilance parameter and c, a proportional constant. Increase of the value of one of these parameters will lead to a more narrow area and thus to a network in which patterns have to resemble another more in order to be assigned to the same cluster. The reason why ART 2 networks cluster patterns according to such a clear criterion is not difficult to understand. This is due to two properties of the network. The first property is the cooperation between the non linear function and normalisation steps within F1 which emphasise maxima within the pattern. If the non linear parameter is large enough only the absolute maximum of the pattern will remain present after applying the non linear function at the pattern. The second property is the cooperation between the bottom up calculation and the competitive process which selects the output node with a maximum value as the winner after a bottom up calculation. Because this calculation is a vector multiplication between an input pattern and a prototype it is likely that prototype whose largest peak is most near to the absolute of the input winner will win the competitive process.

With such a clear criterion a start is made with the solution of the classification problem. Although some simulations have shown, that the network is able to come to some global clustering it is immediately clear that preprocessing is necessary because top V is not the global but a local maximum. This preprocessing has to achieve, that top V becomes the global maximum with high probability in order to be able to detect top V with this network. Simulations have shown that preprocessing indeed improves the clustering. In combination with suitable preprocessing the meaning of 2 parameters becomes clear: increase of θ, the non linear threshold, and increase of ρ, the vigilance parameter, leads to an important improved clustering if the values for both parameters is kept beyond certain bounds. For ρ it is necessary to mention, that ART 2-2 has to be used in order to be able to distinguish between patterns by choosing a suitable value for ρ.

The simulations which have been done with the pattern set which consist of patterns in which top V is the absolute maximum have shown, that it is possible to come to an acceptable clustering in which the deviation of latencies within one cluster is at most 0.1 ms. Taking into account, that this is reached with an unsupervised network in which 9 parameters have to be estimated, this can be seen as a good result. For such a network it has been shown, that the network is capable in storing some general information. Applying a test set to a trained network leads to a clustering in which the distribution of patterns in different clusters resembles the distribution of the patterns of the training set after training of the net. This result gives rise to a final question which will be the subject of the last section of this report:

7.4 Can ART 2 solve the BAEP problem?

Whether ART 2 can solve the BAEP problem can not be concluded from the study described in this report. We only studied the behaviour of ART networks making use of simplified BAEP patterns. The

results of this study can be found in this report. In order to find an answer to the question the (stylistic) BAEP problem has to be studied extensively. We hope that this report contributes to a solution to this problem.
It is expected, that this problem is not easy to solve because the criterion to which ART 2 clusters patterns is very clear. Due to this, the way input patterns have to be preprocessed is very clear: the patterns have to be transformed in such a way, that top V becomes the global maximum with high probability in order to reach a good clustering. If this is reached, the ART 2 network "reduces" to a network which only extracts the global maximum from an input pattern. It is doubtful then whether a neural network has advantages above a network which is simply developed to detect maxima in patterns. It is even doubtful, that if the pattern recognition problem can be solved by combining ART 2 with a suitable preprocessor, it has nothing to do with neural networks because there has to be incorporated so much knowledge into the preprocessor that it is difficult to speak about "self organisation".

In order to find an answer to all these doubts and questions we therefore recommend to try to solve the stylistic BAEP pattern recognition problem first with ART 2-2. It is best to choose the non linear threshold θ as small as possible then, in order to provide, that the "global maximum criterion" becomes less important. If in that case an acceptable clustering can be found it is very important to investigate how the network behaves if noisy test sets are applied to a trained network. If these investigations show, that the network is still able to extract top V with the same distribution as reached with the training set, the neural network has indeed advantages above ordinary "maximum selectors". Then it can be tried to solve the real BAEP pattern recognition problem making use of experience gained while solving the stylistic BAEP problem. The solutions which will be found then, can be compared with earlier mentioned solutions of the BAEP problem with a Multi Layer Perceptron and the Counter Propagation network. If simulations with noisy patterns show bad results, then ART 2 has no clear advantages above ordinary maximum selectors. It is best to conclude then, that ART 2 can not be used as a tool to extract top V from BAEP patterns.

References

[Ackley & Hinton, 1985]
Ackley, D.H., Hinton, G.E. and Sejnowski, T.J., "A Learning Algorithm for Boltzman Machines", *Cognitive Science*, vol. 9, pp. 147-169, 1985.

[Barna & Kaski, 1989]
Barna, G. and Kaski, K., "Variations on the Boltzman machine", *J. Phys. A: Math.*, vol. 22, pp. 5143-5151, 1989.

[Carpenter & Grossberg, 1986]
Carpenter, G.A. and Grossberg, S., "Neural Dynamics of Category Learning and Recognition: Attention, Memory Consolidation and Amnesia", in Davis, J., Newburgh, R. and Wegman, E. (eds.): *Brain, Structure, Learning and Memory*, AAAS Symposium, 1986.

[Carpenter & Grossberg, 1987a]
Carpenter, G.A. and Grossberg, S., "ART 2: self-organization of stable recognition codes for analog input patterns", *Proceedings of the First IEEE Conference on Neural Networks*, pp. II 727-II 735, 1987.

[Carpenter & Grossberg, 1987b]
Carpenter, G.A. and Grossberg, S., "ART 2: self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, vol. 26, no. 23, pp. 4919-4930, 1987.

[Carpenter & Grossberg, 1988]
Carpenter, G.A. and Grossberg, S., "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network", *Computer*, pp. 77-88, March 1988.

[Carpenter & Grossberg, 1990]
Carpenter, G.A. and Grossberg, S., "ART 3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures", *Neural Networks*, Vol. 3, pp. 129-152, 1990.

[Cluitmans, 1990]
Cluitmans, P.J.M., *Neurophysiological Monitoring of Anesthetic Depth*. Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, June 1990.

[Denker, 1986]
Denker, J.S., "Neural network models of learning and adaptation", *Physica D*, vol. 22, pp. 216-232, 1986.

[Fisher & McKusish, 1988]
Fisher, D.H. and McKusich, K.B., "An Empirical Comparison of ID3 and Back-Propagation", Techn. Rep. TR CS-88-14, Department of Computer Science, Vanderbilt University, Nashville, TN, 1988.

[Fukushima, 1975]
Fukushima, K., "Cognitron: A self-organizing multi-layered neural network", *Biol. Cybern.*, vol 20, no. 3/4, pp. 121-136. 1975.

[Gorman & Sejnowski, 1988]
Gorman, R.P. and Sejnowski, T.J., "Learned Classification of Sonar Targets using a Massively Parallel Network", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 36, pp. 1135-1140, July 1988.

[Grossberg, 1976]
Grossberg, S., "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors", *Biol. Cybern.*, vol. 23, pp. 121-134, 1976.

[Grossberg, 1982]
Grossberg, S., *Studies of Mind and Brain: Neural principles of learning, perception, development cognition and motor control.* Boston: Reidel Press, 1982.

[Grundy et al., 1982]
Grundy, B.L., Jannetta, P.J., Procopio, P.T., et al., "Intraoperative monitoring of brain-stem auditory evoked potentials", *Journal of Neurosurgery*, vol. 57, pp. 674-681, Nov. 1982.

[Habraken, 1991]
Habraken, J.B.A., *Pattern recognition of evoked potentials with Neural Networks.* M. Sc. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, May 1991.

[Hartigan, 1975]
Hartigan, J.A., *Clustering Algorithms.* New York: John Wiley & Sons, 1975.

[Hebb, 1949]
Hebb, *The Organization of Behaviour.* New York: John Wiley & Sons, 1959.

[Hecht-Nielsen, 1987]
Hecht-Nielsen, R., "Counterpropagation Networks", *Proceedings of the International Conference on Neural Networks I*, pp. II 19-II 32, 1987.

[Hecht-Nielsen, 1989a]
Hecht-Nielsen, R., "Theory of the Backpropagation Neural Network", *IEEE INNS International Joint Conference on Neural Networks*, vol. I, pp. 593-605, 1989.

[Hecht-Nielsen, 1989b]
Hecht-Nielsen, R., *Neurocomputing.* Reading, Massachusetts: Addison-Wesley, first ed., 1990.

[Hopfield, 1982]
Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities", *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554-2558, 1982.

[Hopfield, 1984]
Hopfield, J.J., "Neurons with graded response have collective computational properties like those of two-state neurons", *Proc. Natl. Acad. Sci. USA*, vol. 81, pp. 3088-3092, 1984.

[Hopfield & Tank, 1985]
Hopfield, J.J. and Tank, D.W., "Neural Computation of Decisions in Optimization Problems", *Biol. Cybern.*, vol. 52, pp. 141-152, 1985.

[Jansen, 1990]
Jansen, J.W., *Artefact detectie en verwijdering in auditieve evoked potentials.* M. SC. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, Aug. 1989. (in Dutch).

[Kirkpatrick et al., 1983]
Kirkpatrick, S., Gelatt, C.D. Jr. and Vecchi, M.P., "Optimization by simulated annealing", *Science*, vol. 220, pp. 671-680, 1983.

[Kohonen, 1984]
Kohonen, T., *Self-Organization and Associative Memory*. Berlin: Springer-Verlag, 1984.

[Kolmogorov, 1957]
Kolgomorov, A.N., "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition". Dokl. Akad. Nauk USSR, vol. 114, pp. 953-956, 1957. (in Russian).

[Lawrence, 1990]
Lawrence, L., "Untangling Neural Nets", *Dr. Dobb's Journal*, pp. 38-44, April 1990.

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

[Lippmann, 1987b]
Lippmann, R.P., Gold, B. and Malpass, M.L., "A comparison of Hamming and Hopfield neural nets for pattern classification", Tech. Rep. no. 769, MIT Lincoln Laboratory, Lexington, Massachusetts, May 1987.

[Lippmann, 1989]
Lippman, R.P., "Pattern Classification Using Neural Networks", *IEEE Communications Magazine*, November 1989.

[Maren, 1990]
Maren, A., Harston, C. and Pap, R., *Handbook of Neural Computing Applications*. San Diego: Academic Press Inc., pp. 155-172, 1990.

[McCulloch & Pitts, 1943]
McCulloch, W.S. and Pitts, W., "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Math. Bio.*, vol. 5, 1943.

[Metropolis et al., 1953]
Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E., "Equation of state calculations by fast computing machines", *J. Chemical Pfysics*, vol. 21, no. 6, pp. 1087-1092, June 1953.

[Minsky & Papert, 1969]
Minsky, M. and Papert, S., *Perceptrons*. Cambridge, MA: MIT Press, 1969.

[Mirchandari, 1989]
Mirchandari, G. and Cao, W., "On hidden nodes for neural nets", *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 661-664,
May 1989.

[Prager & Fallside, 1987]
Prager, R.W. and Fallside, F, "A comparison of the Boltzmann machine and the back propagation network as a recognizers of static speech patterns", *Computer Speech and Language*, vol. 2, pp. 179-183, 1987.

[Rosenblatt, 1958]
Rosenblatt, R., *Principles of Neurodynamics*. New York: Spartan Books, 1958.

[Rumelhart et al., 1986]
Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol. 1, Cambridge, MA: MIT Press, 1986.

[Sejnowski & Rosenberg, 1986]
Sejnowski, T.J. and Rosenberg, C.R., "NETtalk: a parallel network that learns to read alloud", *The John Hopkins University Electrical Engineering and Computer Science Technical Report,* JHU/EECS-86/01, 1986.

[Werbos, 1974]
Werbos, P.J., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph.D. Thesis, Applied Mathemetics, Harvard University, November 1974.

[Widrow & Hoff, 1960]
Widrow, B. and Hoff, M.E., "Adaptive Switching Circuits", *1960 IRE WESCON Conv. Record, Part 4*, pp. 96-104, August 1960.

Appendix 1 Prototype storage in LTM

In this appendix will be proven what will be stored in LTM in three particular cases:

1. A pattern **a** applied repeatedly to the network.
2. A pattern set {**a**, **b**} applied repeatedly to the network while both patterns are forced to be associated with one cluster.
3. A pattern set {**a** ← 1 → **a**, **b** ← 1 → **b**} applied to the network while **a** and **b** are again forced to be associated with one cluster.

The equations which describe the storage of patterns in ART 2's LTM are:

$$\begin{cases} \Delta z_{ij\cdot} = d(p_i - z_{ij\cdot}) \\ p_i = u_i + g \cdot z_{ij\cdot} \\ u_i = \dfrac{x_i}{e + \|\underline{x}\|} \end{cases} \tag{1}$$

From this set of equations can be concluded, that every operation only takes place one nodes with the same index. Therefore eq. (1) can be rewritten as:

$$\begin{cases} \Delta \underline{z}_{j\cdot} = d(\underline{p} - \underline{z}_{j\cdot}) \\ \underline{p} = \underline{u} + g \cdot z_{j\cdot} \\ \underline{u} = \dfrac{\underline{x}}{\|\underline{x}\|} \end{cases} \tag{2}$$

In this equation-set j^* denotes the winning output node and e is set to 0.

Due to the resemblance between eq. (1) and eq. (2) the "component" and vector notation can be exchanged at any time.

In order to examine the behaviour of ART 2's LTM in the three described situations, $\underline{z}_{j\cdot}$ has to be calculated as a function of $\underline{x}$. This can easily be done if this problem is regarded as a control engineering problem in which $\underline{x}$ denotes the input to a "control system" described by eq. (1) or eq. (2). $\underline{z}_{j\cdot}$ then denotes the output of such a system. If we rewrite eq. (2) again in order to get a set of system equations in which x(k) denotes the input to that system at sample moment k while y(k) denotes the output, we get:

$$\begin{cases} \Delta y(k) = d(p(k) - y(k)) \\ p(k) = u(k) + g \cdot y(k) \\ u(k) = \dfrac{x(k)}{\|\underline{x}\|} \end{cases} \tag{3}$$

In this eq. set y(k) denotes one weight of the LTM prototype to a winning node and x(k) denotes the corresponding node of the input vector. p(k) and u(k) denote the corresponding nodes within F1.

The equations of eq. (3) can be combined to one equation:

$$y(k+1) - y(k) = d\left(\frac{x(k)}{\|\underline{x}\|} - (1-d) \cdot y(k) \right) \tag{4}$$

This equation describes the input-output relation of the "control system". Because we want to calculate the output of this system as a function of a varying input it is necessary to calculate the transfer function of this "system". Therefore, first the input- and output-variables in eq. (4) have to be separated:

$$y(k+1) - y(k)(-d^2 + d - 1) = d \cdot \frac{x(k)}{|x|} \tag{5}$$

If y(k) = 0, $\forall$ k $\leq$ 0 and x(k) = 0, $\forall$ x < 0, the z transformation of eq. (5) equals:

$$zY(z) - zy(0) + (-d^2 + d - 1)Y(z) = \frac{d}{|x|} \cdot X(z) \tag{6}$$

so

$$Y(z)(z + (-d^2 + d - 1)) = \frac{d}{|x|} \cdot X(z) \tag{7}$$

and thus

$$H(z) = \frac{1}{z + (-d^2 + d - 1)} \cdot \frac{1}{|x|} \tag{8}$$

With this equation it is very easy to calculate the output of the system as a function of a varying input and thus the patterns stored in LTM while the input to ART 2 changes according to 1, 2 or 3.

Appendix 1a: Pattern a applied repeatedly

If a pattern **a** will be applied repeatedly to the network this means, that one particular input node receives a pattern set {a,a,a...a}. This can mathematically be expressed by

$$x(k) = a \cdot \varepsilon(k) \tag{9}$$

in which ε(k) denotes the unity step function.

The z-transformation of this equation is:

$$X(z) = \frac{a \cdot z}{z - 1} \tag{10}$$

so

$$Y(z) = H(z) \cdot X(z) = \frac{1}{1 - d} \frac{a}{|a|} \left\{ \frac{1}{z - 1} - \frac{1}{z + (-d^2 + d - 1)} \right\} \tag{11}$$

We are interested in the behaviour of y(k) if k goes to infinity. Simulations have shown, that LTM converges then. The pattern to which LTM converges then equals:

$$\lim_{k \to \infty} y(k) = \lim_{z \to 1} \frac{z - 1}{z} \cdot Y(z) = \frac{a}{|a|} \cdot \frac{1}{1 - d} \tag{12}$$

For a common input $\underline{x}$ this equation means that the prototype converges to

$$\frac{\underline{x}}{\|\underline{x}\|}\cdot\frac{1}{(1-d)} \tag{13}$$

which means, that if an input pattern $\underline{x}$ will be applied repeatedly to the network, the prototype of the cluster to which $\underline{x}$ is associated converges to a pattern which is proportional to $\underline{x}$.

Appendix 1b: Pattern set {a, b) applied repeatedly

If a pattern set $\{\underline{a}, \underline{b}\}$ will be applied repeatedly to the network, one particular input node receives a pattern set {a, b, a, b, a,..., b} as its input. Mathematically this can be written as:

$$x(k) = \tfrac{1}{2}(a+b)\varepsilon(k) - \tfrac{1}{2}(b-a)(-1)^k\varepsilon(k) \tag{14}$$

in which ε(k) denotes the unity step.

The z-transformation of this equation is:

$$X(z) = \tfrac{1}{2}(a+b)\frac{z}{z-1} - \tfrac{1}{2}(b-a)\frac{z}{z+1} \tag{15}$$

so now

$$Y(z) = \frac{d}{z+(-d^2+d-1)}\left\{\tfrac{1}{2}(a+b)\frac{z}{z-1} - \tfrac{1}{2}(b-a)\frac{z}{z+1}\right\} \tag{16}$$

In order to investigate which value is stored into LTM this equation has to be inverse z-transformed:

$$y(k) = \frac{a+b}{2(d-1)}\{(d^2-d+1)^k - 1\}\varepsilon(k) - \frac{d(b-a)}{2(d^2-d+2)}\{(d^2-d+1)^k - (-1)^k\}\varepsilon(k) \tag{17}$$

We now distinguish two situations.
In the first situation we investigate the value/pattern which is stored in LTM if $\underline{a}$ is the last applied input. This situation can be described as follows:

$$y(2k+1) = \frac{a+b}{2(d-1)}\{(d^2-d+1)^{2k+1} - 1\}\varepsilon(k) - \frac{d(b-a)}{2(d^2-d+2)}\{(d^2-d+1)^{2k+1} + 1\}\varepsilon(k) \tag{18}$$

After z-transformation of this equation y(2k+1) for k to infinity can be calculated according to:

$$\lim_{z\to 1}\frac{z}{z-1}Y(z)\Big|_{2k+1} = \left\{\frac{d}{2(d^2-d+2)} - \frac{1}{2(d-1)}\right\}a + \left\{\frac{-1}{2(d-1)} - \frac{d}{(d^2-d+2)}\right\}b \tag{19}$$

if this limit exists. Then this can be simplified to

$$y(2k+1)\Big|_{k\to\infty} = c_1 a + c_2 b \tag{20}$$

In the second situation we investigate the value/pattern stored in LTM if $\underline{b}$ is the last applied input. Then we have to calculate y(2k):

$$y(2k) = ((d^2-d+1)^{2k} - 1)\left\{\frac{a+b}{2(d-1)} - \frac{d(b-a)}{2(d^2-d+2)}\right\}\varepsilon(k) \tag{21}$$

After z-transformation of this equation y(2k) for k to infinity can be calculated according to:

$$\lim_{z \to 1} \frac{z}{z-1} Y(z)\Big|_{2k} = \left\{\frac{-1}{2(d-1)} - \frac{d}{2(d^2-d+2)}\right\} a + \left\{\frac{d}{2(d^2-d+2)} - \frac{1}{2(d-1)}\right\} b \qquad (22)$$

if this limit exists. This equation now can be simplified to:

$$y(2k)\Big|_{k \to \infty} = c_2 b + c_1 a \qquad (23)$$

From eq. (20) and eq. (23) now a number of important conclusions can be drawn. These conclusions can best be discussed with the vector notation equivalent of eq. (20) and eq. (23):

$$\begin{aligned} \underline{y}(2k+1) &\rightarrow c_1 \underline{a} + c_2 \underline{b}, \quad \textit{if } \underline{a} \textit{ is the last applied input} \\ \underline{y}(2k) &\rightarrow c_2 \underline{a} + c_1 \underline{b}, \quad \textit{if } \underline{b} \textit{ is the last applied input} \end{aligned} \qquad (24)$$

Simulations have shown, that both limits indeed exist. This means, that if k reaches infinity, the prototype switches repeatedly between the two stable patterns $\underline{y}$ of eq. (24). It depends on the last applied input to which one.
It can be shown very easily, that if d increases, then $\underline{y}(2k+1)$ and $\underline{y}(2k)$ resemble more and more if k reaches infinity. For the maximal value: d = 1 it is easy to show that:

$$c_1 = c_2 = \left\{\frac{d}{(d^2-d+2)} - \frac{1}{2(d-1)}\right\}_{d=1} = \frac{1}{2} \qquad (25)$$

In that case $\underline{y}(2k+1)$ and $\underline{y}(2k)$ become exactly the same. The prototype stored into LTM converges to the average of $\underline{a}$ and $\underline{b}$.
If we add c_1 and c_2 we get:

$$c_1 + c_2 = -\frac{1}{2(d-1)} - \frac{1}{2(d-1)} = \frac{1}{1-d} \qquad (26)$$

This is true for all values of k and for every value of d between 0 and 1. This proportional constant is exactly the same as has been found in Appendix 1a.

Appendix 1c: Pattern set $\{\underline{a} \leftarrow l \rightarrow \underline{a}, \underline{b} \leftarrow l \rightarrow \underline{b}\}$ applied to the network

If a pattern set $\{\underline{a} \leftarrow l \rightarrow \underline{a}, \underline{b} \leftarrow l \rightarrow \underline{b}\}$ will be applied to the network, one particular input node receives l time value a first. Next it receives l times value b. Mathematically this can be written as:

$$x(k) = a \cdot \varepsilon(k) + (b-a)\varepsilon(k-l) \qquad \forall k, \quad 0 \le k \le 2l \qquad (27)$$

We are interested in the value/pattern stored in LTM at k = 2l. In that case the z-transform of y(k) equals:

$$X(z) = a\frac{z}{z-1} + (b-a)\frac{z^{-l} \cdot z}{z-1} \qquad (28)$$

so

$$Y(z) = \frac{d}{z + (-d^2+d-1)} \left\{ a\frac{z}{z-1} + (b-a)\frac{z^{-l} \cdot z}{z-1} \right\} \qquad (29)$$

As in appendix 1a and 1b we can calculate the prototype stored into LTM if k → ∞. This prototype equals:

$$\lim_{k \to \infty} y(k) = \lim_{z \to 1} \frac{z-1}{z} Y(z) = \frac{b}{1 - d} \tag{30}$$

if this limit exists. Simulations have shown that this is true.
Since the prototype can be seen as a weighted mixture of a and b it can be concluded, that:

$$y \to c_1 a + c_2 b = \frac{1}{1 - d} \cdot b \tag{31}$$

which shows, that the "mixture" only includes value b, the last applied input. Value a will thus be forgotten .

Appendix 2 The vigilance test

As described in chapter 5, the vigilance test is described by two equations. If e is set to 0 these equations are as follows:

$$r_i = \frac{|u_i| + |cp_i|}{|\underline{u}| + |c\underline{p}|} \tag{1}$$

$$\frac{\rho}{|\underline{r}|} > 1 \tag{2}$$

In eq. (1) a weighted summation is calculated of the patterns $\underline{\mathbf{u}}$ and $\underline{\mathbf{p}}$ which have to be compared. eq. (2) expresses when these patterns are not accepted as a fit. In that case the network has to cause a RESET.

In order to examine the meaning of this vigilance test, we investigated which areas in 2 dimensional space are accepted as a fit to a particular prototype $\underline{\mathbf{p}}$ as a function of ρ and c. We choose $\underline{\mathbf{p}} = (p, 0)$ as a prototype and have calculated for which regions $\underline{\mathbf{u}} = (u_1, u_2)$ in 2 dimensional space $\rho/|\underline{r}| \leq 1$.

Because $\underline{\mathbf{r}}$ is a pattern which can be seen as a vector, the length of this vector can be calculated according to:

$$|\underline{r}| = \sqrt{\Sigma\, r_i^2} \tag{3}$$

Therefore

$$|\underline{r}| = \frac{(|\underline{u}|^2 + 2c|\underline{u}||\underline{p}|\cos\theta + c^2|\underline{p}|^2)^{\frac{1}{2}}}{(|\underline{u}| + |c\underline{p}|)^2} \tag{4}$$

From eq. (2) and eq. (4) can be concluded, that $\underline{\mathbf{p}}$ and $\underline{\mathbf{u}}$ fit if:

$$\rho^2(|\underline{u}| + |c\underline{p}|)^2 \leq (|\underline{u}|^2 + 2c|\underline{u}||\underline{p}|\cos\theta + c^2|\underline{p}|^2) \tag{5}$$

We now choose $\underline{\mathbf{p}} = (p, 0)$; $|\underline{\mathbf{p}}| = p$ and switch to polar coordinates (r, φ). If θ can be seen as the angle between $\underline{\mathbf{p}}$ and $\underline{\mathbf{u}}$, this yields: $r = u$ and $\varphi = \theta$.
Eq. (5) reduces to

$$\rho^2(r + cp)^2 \leq (r^2 + 2crpcos\varphi + c^2p^2) \tag{6}$$

then. Rewriting of this equation leads to

$$\tfrac{1}{2}(\rho^2 - 1)(r^2 + c^2p^2) + cpr\rho^2 \leq cpr\cos\varphi \tag{7}$$

With $m = (\rho^2 - 1)/2$ and thus $\rho^2 = 2m + 1$ this equation reduces to

$$m(r^2 + c^2p^2) + cpr(2m + 1) \leq cpr\cos\varphi \tag{8}$$

If we separate the terms with r^2 and r we get:

$$r^2\frac{m}{cp} + r(\,2m + 1 - \cos\varphi\,) + mcp \leq 0 \qquad \textbf{(9)}$$

The solution for this equation is not difficult. With p = 1 the solution for the equality in eq. (9) becomes:

$$r = \frac{-2m - 1 + \cos\varphi \quad \sqrt{(\,2m + 1 - \cos\varphi\,)^2 - 4m^2}}{2m/c} \qquad \textbf{(10)}$$

This is the equation which is also shown in chapter 5.