University of Salzburg, Institute for Computer Science

PS Introduction to Bioinformatics
SS 1999

Applications of Hopfield Networks (HN)

Mayer August	ags@gmx.net
Wiesbauer Gerald	gerald.wiesbauer@sbg.ac.at
Spitzlinger Markus	mspitzl@cosy.sbg.ac.at
Leader:	
Dipl.Ing. Mag. Roland Schwaiger	rschwaiger@cosy.sbg.ac.at

Version from April 13, 1999

Abstract

This document gives an overview of the structure of Hopfield Networks (HN) and describes some applications in detail.

Contents

Chapter 1

Introduction

1.1 About Neural Networks

Accoring to [**?**] an artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generaliziations of mathematical models of human cognition or neural biology, based on the assumptions that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connections links.
3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by its pattern of connections between the neurons (called its architecture), its method of determining the weights on the connections (called its training, or learning algorithm), and its activation function.

A neural net consists of a large numer of simple processing elements called neurons, units, cells or nodes. Each neuron is connected to other neurons by means of directed communicatons links, each with an associated weight. The weights represent information being used by the net to solve a problem.

Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

1.1.1 A simple Adaptable Node

According to [Ale93] a simple Adaptable Node or Toy Adaptive Nod (TAN) is shown as some sort of electronic component.

Figure 1.1.1 shows a TAN. Conventionally in giving descriptions of computing circuits, ignoring the actual values of voltages or currents in this circuit and simply assuming that connection can be one of two states: 0 and 1.

There are N inputs labelled X1, X2, ... XN, and one output F. The actual 0 and 1 values present at the input will often be referred to as an input bit pattern since the 1s and 0s can be thought of as binary digits or bits, as in conventional computing. The inputs of one node are connected either to the outputs of other nodes or to some external source. The function F of a node is a list (or truth table) which shows what F will be for each of the possible patterns.

Example:
The function of a 3-input node can have the following truth table:

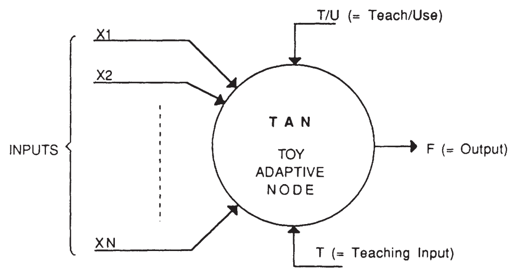

Figure 1.1: The Toy Adaptive Node (TAN).

X1: 0 0 0 0 1 1 1 1
X2: 0 0 1 1 0 0 1 1
X3: 0 1 0 1 0 1 0 1
F: 0 1 0 0 1 1 0 1

The table is a simple means of showing that:

when X1=0, X2=0 and X3=0, then F=0

when X1=0, X2=0 and X3=1, then F=1

etc ...

The node may operate in two modes: it can be taught (during which time its function can change) or used (during which time it cannot change). Which of these modes is operating is determined by setting the T/U terminal to 1 for teaching or to 0 for using. While teaching, the node associates whatever is present (i. e. 0 or 1) at the teaching input T with whatever patern is present at the X1, X2, ... terminals. This means that whenever that particular input pattern occurs again in a subsequent using phase, the learnt or associated output symbol (i. e. 0 or 1) is at F. There is also an undefined output, 0 or 1, chosen at random which exists either if the particular input has not been taught at all, or if there is some sort of confusion about the desired output for a particular input. For example, if the node is taught to output a 1 in response to a particluar input pattern and is then taught to output a 0 in response to the same pattern the output will be undefined.

1.1.2 Network Stuctures

There are two types of networks: feed-forward (or associative) networks and feedback networks.

A feed-forward net is one in which are no closed loops. A single-layer feed-forward net is a net where every output node is connected to every overall input terminal. Nodes which are not directly connected to the output, in other feed-forward nets, are called hidden nodes which cause problems during training.

A feed-back net is one where information can find its way around a loop from the output back to the input. There is one special type of fee-back nets, called autoassociative where no overall input terminals exist. Training takes place so as to use a selected pattern on internal connections as data both for inputs and desired outputs of the nodes. Instead of associating a pattern at an overall input with a pattern at the overerall output (as occurs in the other nets) the net associates the training pattern with itself.

1.2 History of Neural Networks

Accoring to [Ale93] the beginning of neural computing was in 1936, when Alan Turing, a British mathematician laid down the principles for defining what is and what is not computable. He also wrote about the way in which the brain did its computations through cycles of activity in neural nets. Another mathmatician, John von Neumann, also talked about neural nets, seeing them as providing a way of doing computations that emerges from the structure of the net itself.

Norbert Wiener, whose name is associated with the definition of cybernetics, thought that the fundamental structure to a mathematical understanding of computation in both brains and machines is the interplay of logic and network structure. But the algorithmic method has always been seen as the well-behaved, predictable kind of computing and so neural networks laid largely dormant for a long time.

Years of definition (1943 - 1969) followed, where a model of the fundamental cell of the living brain, the neuron, was suggested from the neurophysiologist Warren Mc Culloch and the logician Walter Pitts. Another important idea was put forward in 1949 by Donald Hebb, who suggested that a group of neurons could reverberate in different patterns, each of these patterns relating to the recall of a differnet experience. This was the first time a model of dynamic memory had been used by a neuropsychologist.

With this and a few other theories the capabilities of the nets where understood and they could be simulated on contemporary computers. This empirical approach was severely challenged in 1969 by Marvin Minsky and Seymour Papert. They wrote that neural nets are able to learn to perform tasks that requires internal represantations not explicity specified by the training data. The net must somehow work out these representation for itself and this training turns out to be a difficult task and so it is very much part of current debate and most modern learning algorithmus and net structures are being designed with the possibility of solving this difficulty in mind.

Today most nets are studied as programs running on serial computers, but the future turns upon the ability of engineers to make physically parallel systems. There is a little doubt that John Hopfield of the California Institute of Technoloy was responsible for the breakthrough of neurals nets, because his orientation was very practical. He thought that neural nets are a good way of making advanced computer memories.

Neurals nets are used in a wide range of applications. One of the most successful and proven demonstrations of the ability of neural nets is to take symbolic representation of words as letters and generate phoneme (sound segment) code that may be used to drive a voice synthesizer. The opposite problem is the speech recognition and another thing is that a net could be made to follow conversations aubout simple scences and paraphrase simple stories. But probably the major application of neural nets lies in the recognition of visual patterns.

The future of neural computing are hybrid systems, which find out from the communication between the user and the machine what the user wants.

1.3 About Hopfield Networks

In short[1], the main topic in Hopfield networks is that they are recurrent neural networks, where the term ”recurrent” means that the output values are feeded back to the input of the network in an undirected way. Hopfield Networks are mainly used in pattern recognition and optimization. Learning is performed by modifying the strength of the connections between the neurons. The network consists of a layer where the output of each neuron is connected to the input of all others.

1.3.1 Construction

A Hopfield network consists of binary neurons which are connected by a symmetric network structure. Binary means that the neurons can be active (”firing”, state 1) or inactive (”not firing”, 0). The connections are weighted, and depending on the sign of the weight they can be intercepting or activating, e.g. a firing neuron activates all neurons which are connected to it with a positive weight. There is a threshold value for every neuron which the sum of the input values must reach to produce activity.

[1] see [Bäu97] and [**?**, pp.137ff]

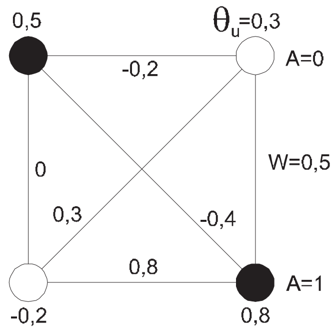

Figure 1.2: A simple example of a Hopfield network.

The network output is calculated by starting with the neurons activation corresponding to the pattern to recognize, then iterating the network and recalculating the state of the neurons until the network is stable, which means that the network state doesn't change any more. This is possible in a finite amount of time and iterations for Hopfield networks. This can also be seen as the minimization of the energy in the net, so that the final state is a minimum of an energy function called *attractor*.

1.3.2 Formalism

As described in [**?**], a Hopfield Network $HN = (U, W, A, O, NET, ex)$ is a Neural Network, where

- the weight matrix (network structure) $W : U \times U \to \mathbf{R}$ is symmetric and doesn't connect any neuron to itself, i.e. $W(u, u') = W(u', u)$ and $W(u, u) = 0$,
- there is an activation function $A_u \in A$ for every neuron (unit) $u \in U$ which is $+1$ if $net_u \geq \theta_u$ and -1 otherwise; the threshold $\theta_u \in \mathbf{R}$ is defined statically for every neuron;
- there is an output function $O_u \in O$ for every neuron $u \in U$, which is the identity function.
- there is a network input function (propagation function) $net_u \in NET$; this function is calculated for a neuron by multiplying the outputs of all the other neurons with the weights of the connections to that neuron and then summarizing them up, i.e.

  $$net_u = \sum_{u' \in U} W(u', u) \cdot o'_u;$$

  remind that the weight of the connection from a neuron to itself is always 0 in a HN;
- there is a function $ex(u)$ which specifies the external input value for every neuron, i.e. $ex : U \to \{-1, +1\}$; sometimes also the value set $\{0, 1\}$ is used, which results in some minor formula changes.

The inventor of the HN, John Hopfield, thought more in terms of an energy-based analysis of the magnetic anomalies of the so-called spin-glasses. He calculates the energy of a neuron u as follows:

$$E_u = -a_u \cdot \left(\left(\sum_{u' \in U} W(u', u) \cdot a_{u'} \right) - \theta_u \right) = -a_u \cdot (\text{activation})$$

The energy of the whole system is then calculated as follows (note the factor $\frac{1}{2}$ because of the symmetry of the net):

$$E = -a_u \cdot \sum_{u \in U} \left(\left(\frac{1}{2} \cdot \sum_{u' \in U} W(u', u) \cdot a_{u'} \right) - \theta_u \right)$$

A change in the network activation always results in an energy decrease, because the activation change Δa_u always has the same sign as the network input minus the threshold $net_u - \theta_u$, so that ΔE_u is always negative or zero.

If all neurons of a HN are updated at the same time, there can be cyclic state changes. So the neurons are either updated in a random or a defined order. It can be proven (see [**?**, p142]) that in that way, a final state is reached after at most $n \cdot 2^n$ steps.

According to [**?**, p95], the maximum number of patterns a HN is able to memorize is

$$p_{\max} = 0,138N,$$

where N is the number of neurons in the net. For big N, the formula

$$p_{\max} = \frac{N}{2 \ln N}$$

is valid.

A Hopfield Network also has local minima other than the learned patterns. A common example is the exact inverse pattern, but there are also other *spurios states*. The attractivity of those local minima is usually rather small compared to the learned patterns if the capacity of the net has not been exceeded.

1.3.3 Example

As previously described, the network detects a pattern which is mapped as the initial activity states of the neurons step-by-step in a finite amount of time. Here an example how this works.

First, a neuron is selected, either using a fixed order or a random permutation of the neurons. Then the outputs of all the other neurons are multiplied by the weight of the connections to the active neuron, and these products are summed up. If the sum exceeds the threshold, the new state of the neuron is active, else inactive.

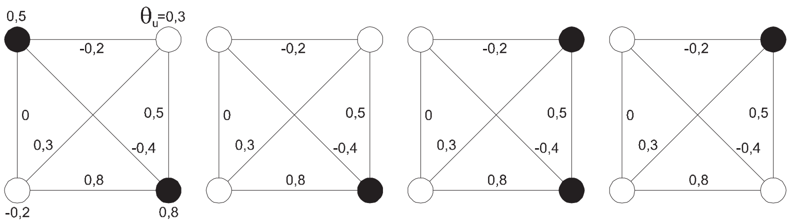

Figure 1.3: First pattern recognition steps.

As shown in figure 1.3, the upper left neuron is selected and its state is recalculated (filled circles mean active neurons, empty inactive ones). So the neuron gets the inputs $0 \cdot -0.2$ (from the upper right neuron), $1 \cdot -0.4$ (lower right neuron) and $0 \cdot 0$ (lower left neuron), which sums up to -0.4. This is smaller than the threshold $\theta_u = 0.5$ (stays the same for all drawings and is only shown on the first image above or below the respective neuron), so the activation of the upper left neuron switches to 0.

The next neuron is the upper left neuron, and its input $0.2 \cdot 0 + 0.3 \cdot 0 + 0.5 \cdot 1 = 0.5$ is greater than the threshold value $\theta_u = 0.3$, which results in the neuron turned active. After that, the lower right neuron is calculated, and it stays deactivated, and the lower left neuron's activation state becomes one because it gets enough input (can be seen on the next figure 1.4).

Remind that the state of the neurons is calculated one after the other, and the activation values are *not* taken from the network state at the beginning of the iteration, but the actual states are used. Otherwise, the recalculation would be parallel for all neurons, and then the network needn't reach a stable state.

The procedure is continued until a full iteration is done without changing a neuron, which means that a stable state has been reached. The last state of Fig.1.4 is such a state. Note that it is possible to recalculate any neuron without its state getting changed. The resulting states of the neurons form the recognized pattern.

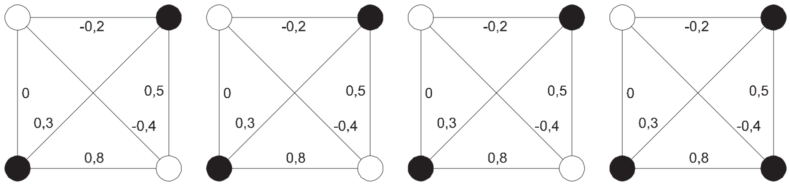

Figure 1.4: Next pattern recognition steps.

1.3.4 Learning in Hopfield Networks

Neural networks, in particular HNs, can learn their stable states from examples[2]. Rules which can be used for this are Hebb's learning rule (1949) and a development of this rule, the Widrow-Hoff rule (1960). Hebb's rule states that the connection strength between two neurons changes proportionally to the frequency of the neurons firing together. The Widrow-Hoff rule or Delta learning rule says that the greatest sources of error are from active lines, so that

$$\text{Weight Change} = (\text{Pre-Weight Line Value}) * \frac{(\text{Error})}{(\text{Number of Inputs})}.$$

The error signal is measured by calculating the activation of a neuron and then comparing this to the required signal, i.e.

$$\text{Error} = (\text{Desired Output}) - (\text{Actual Output}),$$

where only 'active' neurons' weights are recalculated.

So the network performs learning by feeding the desired state to the network; if connection strengths

With this system it is theoretically possible to understand the network by making a step-for-step calculation and evaluation of the network states and the resulting error.

There should be also noted that if an unit has to fire, but its activation is negative, the Widrow-Hoff rule causes it to be made positive, so that it decreases the total energy. If a unit is firing and should not, the activation is made negative along with the value of V_i, which again leads to a reduction of energy[3].

The effect of the Widrow-Hoff learning rule is that the energy of the appropriate state is lowered in comparison with its 'neighbouring' states (i.e. states differing by only one node).

Another aspect is that such networks can be considered as inequalities, and training the former provides an automatic way of solving such equations, which would be relatively time-consuming using standard methods.

1.3.5 The training algorithms

A stabile state in the network is a state where no node changes its firing condition. In this section two ways are described to "train" the network to reach the desired stable states:

- calculating the values of thresholds and strengths by hand – "handcrafted values"
- training the network with examples

[2]For a detailed step-by-step example, see [Ale93, pp.105–109]. Also derived from [Olm98].

[3]For an example, again see [Ale93, p.109]

1.3.6 The calculating method

For example, we have a HN with 4 nodes, each node is connected to each other. The desired stabile states are:

$V_1 = 1, V_2 = 0, V_3 = 1, V_4 = 0$
$V_1 = 1, V_2 = 0, V_3 = 0, V_4 = 1$

Description of the values: V_n is the output condition of the node, $T_ij = T_ji$ is the connection strength between the nodes, and U_n is the threshold. ≥ 0 means this node is active ≤ 0 means the node is not active.

State 1:
$V_1 : T_{12} * V_2 + T_{13} * V_3 + T_{14} * V_4 - U_1 \geq 0$
$V_2 : T_{21} * V_1 + T_{23} * V_3 + T_{24} * V_4 - U_2 \leq 0$
$V_3 : T_{31} * V_1 + T_{32} * V_2 + T_{34} * V_4 - U_3 \geq 0$
$V_4 : T_{41} * V_1 + T_{42} * V_2 + T_{43} * V_3 - U_4 \leq 0$

According to the (stabile state) values of V_n:

$V_1 : T_{13} - U_1 \geq 0$
$V_2 : T_{21} + T_{23} - U_2 \leq 0$
$V_3 : T_{31} - U_3 \geq 0$
$V_4 : T_{41} + T_{43} - U_4 \leq 0$

State 2:

$V_1 : T_{12} * V_2 + T_{13} * V_3 + T_{14} * V_4 - U_1 \geq 0$
$V_2 : T_{21} * V_1 + T_{23} * V_3 + T_{24} * V_4 - U_2 \leq 0$
$V_3 : T_{31} * V_1 + T_{32} * V_2 + T_{34} * V_4 - U_3 \leq 0$
$V_4 : T_{41} * V_1 + T_{42} * V_2 + T_{43} * V_3 - U_4 \geq 0$

According to the (stabile state) values V_n:

$V_1 : T_{13} - U_1 \geq 0$
$V_2 : T_{21} + T_{24} - U_2 \leq 0$
$V_3 : T_{31} + T_{34} - U_3 \leq 0$
$V_4 : T_{41} - U_4 \geq 0$

A value set which make this two states as stabile one is:

$U_1 = -0.2$
$U_2 = 0.6$
$U_3 = -0.2$
$U_4 = 0.3$
$T_12 = 0.3$
$T_31 = 0.3$
$T_41 = 0.4$
$T_43 = -0.6$
$T_23 = 0.1$
$T_24 = 0.1$

Test these values by put them into the formulars:

State 1:

$V_1 : 0.3 + 0.2 \geq 0$ OK, $V_1 = 1$
$V_2 : 0.3 + 0.1 - 0.6 \leq 0$ OK, $V_2 = 0$
$V_3 : 0.3 + 0.2 \geq 0$ OK, $V_1 = 1$
$V_4 : 0.4 - 0.6 - 0.3 \leq 0$ OK, $V_1 = 0$

State 2:

$V_1 : 0.3 + 0.2 \geq 0$ OK, $V_1 = 1$
$V_2 : 0.3 + 0.1 - 0.6 \leq 0$ OK, $V_2 = 0$
$V_3 : 0.3 - 0.6 + 0.2 \leq 0$ OK, $V_3 = 0$
$V_4 : 0.4 - 0.3 \geq 0$ OK, $V_4 = 1$

After the network is programmed with these values the two desired states are stablie, but there could be a problen with the so-called false wells, these are unwanted stabile states resulting by the choosen values.

1.3.7 Example for training the network

This method uses the Widrow-Hoff rule and starts as follows: The strengths and thresholds are equal in each node and set to a default value, (i.e 0.5 for the strengths and 0.0 for the thresholds)

The next step is to apply these values to the formulas, according to the example described at the calculating method. 2 of 4 nodes are enough to show the algorithm, the whole calculation is too long to be described here.

$V_1 : T_{13} - U_1 \geq 0$
$V_2 : T_{21} + T_{23} - U_2 \leq 0$

this means:

$V_1 : 0.5 - 0.1 = 0.4$
$V_2 : 0.5 + 0.5 - 0.0 = 1.0$

These variables (0.4, 1) are the condition values for firing, but not the desired ones (0.1, -0.2). The difference between these two tuples is the error-signal of the Widrow-Hoff rule, keep these desired values as small as possible.

$V_1 : 0.5 - 0.0 = 0.5$, desired $= 0.1, e = 0.4$
$V_2 : 0.5 + 0.5 - 0.0 = 1.0$, desired $= -0.2, e = 1.2$

The recalculation of the 'active' nodes is as follows:

$V_1 : 0.3 - 0.2 = 0.1 \geq 0$
$V_2 : 0.1 + 0.1 - 0.4 = -0.2 \leq 0$

The algorithm to perform this recalculation is simple. Divide the error-signal by the number of values at the right side and substract each value on this side by the 'divided error signal'.

This recalculation has to be applied to each node until all error-signals are zero.

Chapter 2

Applications of Hopfield Networks

2.1 The Travelling Salesman Problem (TSP)

2.1.1 About the Travelling Salesman Problem

According to [?] and [?] the Travelling Salesman Problem) is probably the most well-known member of the wider field of combinatorial optimization (CO) problems. The problem is that a Salesman wants to travel trough N cities and he wants to know the shortest way through those cities.

These are difficult optimization problems where the set of feasible solutions (trial solutions which satisfy the constraints of the problem but are not necessarily optimal) is a finite, though usually very large set. The number of feasible solutions grows as some combinatoric factor such as , where N characterizes the size of the problem.

Example:

You are a salesman and you must visit 20 cities spread across North America. Yo must visit each city once and only once. The question is this: In what order should you visit them to minimize the total distance that you have to travel?

The answer is that there is no simple answer. Reasonable people will make a reasonable choice and accept a reasonably short path.

However there is only one way to find the absolute shortest path, and that is to write down every possible ordering of cities, compute the distance for each of those orderings, and pick the shortest one.

How many orderings are there? They can be counted this way:

For the first city to visit you have 20 choices.

For the second city to visit you then have only 19 choices (because you can't visit the first city again).

For the third city you have 18 choices, and so on ...

For the last city you have 1 choice.

These numbers must be multiplied together to give the total number of orderings:

$20 * 19 * 18 * ... = 20! = 2,432,902,008,176,640,000$ possible orderings.

This number is so big that if your computer could check 1 million orderings every second it would still take 77,000 years to check them all! Thus even though we know how to solve the Travelling Salesman Problem we still can't do it.

2.1.2 Approximation of TSP with Hopfield Networks

Here are a few adresses of WWW-pages where to find programs which simulate the TSP.

A JAVA-Program:
http://www.essex.ac.uk

A C-File:
http://chopin.bme.ohio-state.edu/holgi/projects/C/C.html

Others:
http://www.CSI.utoronto.ca/m̃ackay/itprnn

2.2 Pattern Recognition

2.2.1 About Pattern Recognition

According to [?] many interesting problems fall into the general area of pattern recognition. One specific area in which many neural network applications are used is the automatic recgonition of handwritten characters (digits or letters). Humans can perform relatively easily the large information of variation in sizes, positions and styles of writing but it is very difficult problem for traditional techniques.

General-purpose multilayer neural nets, such as the backprogagation net (a multilayer net trained by backprogation) have been used for handwritten zip codes. Even when an application is based on a standard training algorithm, it is quite common to customize the architecture to improve the performance of the application. This backprogation net has several hidden layers, but the pattern of connections form one layer to the next is quite localized.

An alternative approach to the problem of recognizing handwritten characters is the neocognitron. This net has several layers, each with a highly structured pattern of connections form the previous layer and to the subsequent layer. However, its training is a layer-by-layer process, specialized for such an application.

2.2.2 Pattern Recognition with Hopfield Networks

According to [?] there is a program to make Hopfield networks and a recall program that completes an incomplete pattern using the Hopfield, Boltzman and interactive activation network methods.

INTRODUCTION

This software was done in part for a textbook on AI he has written called

The Pattern Recognition Basis of AI

published by IEEE Computer Society Press, see

http://www.dontveter.com/basisofai/basisofai.html

and/or http://www.computer.org/cspress/catalog/bp07796.htm

Software updates may be found at: http://www.dontveter.com/nnsoft/nnsoft.html

There are two programs included in this package, hop.cpp and recall.cpp. The purpose of hop.cpp is to take a set of patterns in and produce the weight matrix you get from the simple weight making algorithm in the text, the same algorithm used by Hopfield. The sbp program in the backprop package can also be used to produce a weight matrix that has symmetric weights between the units, simply save the weights to the file weights. The recall program implements the Hopfield and Boltzman machine pattern completion algorithms and an interactive activation algorithm as well. The interactive activation method is said to be better at avoiding local minima although with it you do not get a guarantee of finding any minimum at all. There are two examples in this package as well, the data files to do the pi/e completion problem given in the text and the sports problem given as an exercise. This is the same sports example given in the backprop package and in the text.

2.3 Associative memory

A rough description for understanding this system is: The system is a memory, in which you can read and write items. You have addresses for these items, but this kind of memory has the capability to find the correct item. Altrough there is a distortion between the address which has written the item and the address which reads it. Indeed this is a very interesting capability, related to a biological cortex.

Appendix A

Records

A.1 Record of March 10th, 1999

Leader:	DI Mag.Roland Schwaiger
Recorder:	Mayer August
Participants:	Spitzlinger Markus
	Wiesbauer Gerald
Place:	Salzburg
Time:	12^{30}

- General discussion
- Website under http://www.cosy.sbg.ac.at on Mr. Schwaigers Homepage
- Requirements for the paper
  - Structure
  - Records in the appendix
  - Goal: Understanding of the topic, (detailed) description of some applications
- LaTeX-file until Di 13^{00} as TeX-Source
- Possible Applications:
  - TSP (Travelling Salesman Problem)
  - Associative Memory: Picture searching (IBM), denoising of pictures
- Should programs be written?
  - not required, but appreciated
  - also: search ready programs in the Internet
  - utilization of Java
  - would be good for presentations of the institute
- Are there ready-made simulators?
  - SNNS (Stuttgarter Neuronale Netze Simulator) is/will be installed
  - optional: search in the net
- Literature
  - Igor Aleksander, "'Introduction to Neural Computing"'

- Activities until next week:
  - Compressed overview of Hopfield networks
  - Later: search applications in the Internet

A.2 Record of March 17th, 1999

Leader: DI Mag. Roland Schwaiger
Recorder: Spitzlinger Markus
Participants: Mayer August
Wiesbauer Gerald
Place: Salzburg
Time: 12^{30}

- Old points:
  - Title-page
  - Photos (Wiesbauer, Mayer)
  - Picture
- New points:
  - Glossary
  - Studying Igor Aleksander, "'Introduction to Neural Computing"'
  - Detailed description of Hopfield Networks
    - Example
    - How learns a HN?
  - Simulate a HN with SNNS

A.3 Record of March 24th, 1999

Leader: DI Mag. Roland Schwaiger
Recorder: Wiesbauer Gerald
Participants: Mayer August
Spitzlinger Markus
Place: Salzburg
Time: 12^{30}

- Consider the interface between human and machine, and the useability
- Improve the example for the HFN
- Example with 4 neurons to test the SNNS
- Consider the possibility of calculating the attracktors (stabile state)
- Rework the text, which consider the learning of a HFN
- List the kinds of ANN
- Consider a formular description of ANN

- Exercises:
  - Work with the SNNS use example ......
  - List examples of HFN (in the real world)
  - How many stable states are there in an N neuron HFN?
  - Form an accurate description of the components of an ANN
- Additional:
  - EMMA: Discuss the algorithm of current pattern recogition, implemented in ”EMMA’s-brain”
  - Feature space, results after a fourie-transformation
- Books:
  - D.Mauck F.Klawonow R.Kruse, ”Neuronale Netze + Fuzzy Logic”

Appendix B

Glossary

cybernetics: The comparative study of the internal workings of organic and machine processes in order to understand their similarities and differences. Cybernetics often refers to machines that imitate human behaviour.

firing: The word comes from the language of the biologist interested in the real neurons of the brain. A neuron is said to fire when it emits a buzz of electrical pulses at the rate of about 100 per second.

firing rule: It determines how one calculates whether the node should fire for any input pattern.

Neural Computing: Is the study of networks of adaptable nodes which, through a process of learning from task examples, store experiential knowledge and make it available for use.

neuron: Is the adaptive node of the brain.

nodes: Are adaptable of the nodes of the brain, they acquire knowledge through changes in the function of the node by being exposed to examples.

strength: see weight

wells: Wells are stable states.

weight: Firing rules are couched in terms of some weight (or strength) associated with each connection in the net.

Bibliography

[Ale93] I[gor] Aleksander. *An Introduction to Neural Computing*. Chapman&Hall, 1993.

[Bäu97] Ch[ristian] Bäumer. Hopfield netze. Vortrag im Seminar Neuronale Netze und Fuzzy-Systeme im Sommersemester 1997, 1997.

[Olm98] D[avid] D. Olmsted. History and principles of neural networks. . http://www.neurocomputing.org/history.htm, 1998.