CONTENTS

LIST OF SYMBOLS

N	: number of neurons
σ_i	: neuron state, with i = 1, 2, …, N
θ_i	: bias
q	: activitiy of the neurons
S	: set of possible neuron states
p	: number of stored patterns
ξ_i^μ	: patterns to be stored, with μ = 1, 2, …, p
A^μ	: activity of the pattern μ.
m	: overlap with the target (first order overlap m^1)
J_{ik}	: weight associated with the link from node k to node i.
h_i	: local field
η_i^μ	: pattern, derived from ξ_i^μ
l	: activity overlap (second order overlap m^2)
K_{ik}	: weight matrix associated with η_i^μ
g	: transfer function
T	: pseudo temperature[1] associated with g_i
β	: the inverse of T

[1] We use a pseudo temperature; for conversion to the absolute temperature use $T_{abs} = T / k_B$ with k_B Boltzmann's constant = 1.38×10^{-16} erg/ K

Chapter 1

INTRODUCTION

The subject of this project are neural networks. Neural networks are physical systems, constructed in such a way that memories can be stored and retrieved. Our brains are an example of such a network. In this first chapter the neural network is discussed in its most general form.

1.1 General form of a neural network

Many of the concepts used in an artificial neural network (ANN) are borrowed from neuroscience. Although the models used in ANN's are extremely simplified it still might be useful to keep the biological counterpart in mind.
In figure 1 a typical biological neuron is depicted.

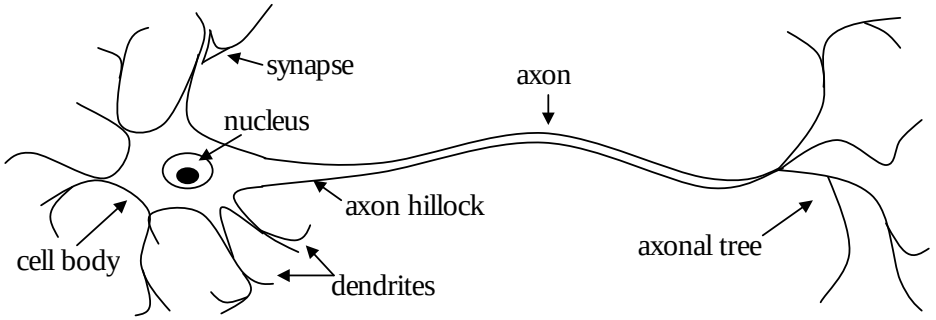

Figure 1: Schematic drawing of a typical biological neuron

Our brain is composed of about 10^{11} such neurons of many different types. The functioning of a neuron can be shortly described by the following. A neuron adds up all its input signals via its dendrites and generates an output via the axon to the synapses. The output depends on the sum of the added input signals. If the potential sums up to a value greater then a certain threshold, then the neuron "fires" and an electrical pulse is sent through the axon to all its synaptic connections. The synapses are connected to the dendrites of other neurons where the same process starts all over. This is why it is called a network; the neurons are connected to each other via thousands of connections, building a huge highly interconnected network. From the insight provided by neurobiology an abstract version was derived which is discussed next.

1.2 Definition of a neural network

Following Müller and Reinhardt (1990) a neural network model is defined in mathematical terms as a directed graph with the following properties:

- A state variable σ_i describing the state associated with each node i.
- A real-valued weight J_{ik} is associated with each link (i-k)
- A real-valued bias θ_i or threshold is associated with each node i.
- A transfer function $g_i[\sigma_i, J_{ik}, \theta_i$, for (k≠i)] is defined, for each node i. This function determines the state of the node as a function of its bias, the weights of its incoming links, and the states of the nodes connected to it by these links.

In standard terminology, the nodes correspond to neurons, the links to synapses, and the bias is known as the activation threshold. The current state is described by the pattern formed by all the neuron states σ_i on that moment in time. For example when somebody is looking at a tree, there actually is a corresponding pattern of activity in his brains, showing more or less a projection of that tree. It is possible to store such patterns in an associative memory.

1.3 Associative memory

To explain what associative memory is, it might be useful to say what it is not. Memory in computers is random access memory (RAM). It uses the memory address as the key to retrieve the stored memory state. So it is essential to know exactly where you stored something in order to be able to retrieve it.
With associative memories on the other hand, the key used is a "guess". In order to retrieve a stored memory state, you have to know roughly or partly how it looked like. This is also how our brain works; if you see a picture of a friend, partly covered so you only see the lower left part, you still can recall how he looks like.

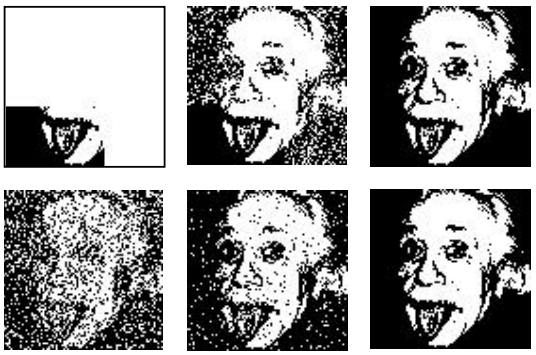

Figure 2: Result obtained with the visualisation program developed during this project, showing how a Hopfield network with binary neurons can recover images of 85 x 85 pixels. Left are the initial states, in the middle an intermediate state is depicted and on the right the fully recoverd target image is presented.

So it is possible to retrieve a pattern stored in memory in response to a presentation of an incomplete or noisy version of that pattern. In other words, the memory is error-correcting. It may be clear that properties like content addressable memory and robustness are desirable for certain applications.
The best studied model for associative memory is the Hopfield model, which is shortly described below. A more elaborate and precise description is given in the following chapters.

1.4 The Hopfield model

Hopfield published his influential paper in 1982, and it was the first time that the physical principle of storing information in a dynamically stable network was formulated in precise terms. The idea was to locate each pattern to be stored at the bottom of a "valley" of an energy landscape, and then permitting a dynamical procedure to minimize the energy of the network in such a way that the valley becomes a basin of attraction.

Now that the general context of this project has been sketched, we can take a look at the different goals and the strategy followed to achieve them.

1.5 Goals

The goals set for this project were described as follows:

- Writing a simulation program for the Hopfield model
- Extending the program such that it can accommodate other models (different architectures, different types of neurons, etc.)
- Applying this extension to a recent model proposed in the literature

In fact this project was intended as the first step for the development of a kind of toolbox which allows an easy implementation of new models and yet has a reasonable performance. A reasonable performance is required since the ultimate goal is to validate the theoretical results obtained for infinite-sized networks. Good simulations can therefore only be obtained using network-sizes which are as large as possible. In order to achieve these goals in an elegant way the extensibility of the code had to be kept in mind already during the development of the code for the Hopfield model. So a strategy was required.

1.6 Strategy

The most natural way to make code easily extensible is by using an object oriented approach. In this way for each important concept a base-class is created. From these base-classes one can derive other classes (also called child-classes) which are slightly modified with regard to the base-class in order to incorporate specific extensions.
A supplemental advantage of object oriented programming is that it implements the saying "divide et impera" in a very natural way. The problem is divided in small controllable portions which are handled within a class.
A very important issue when working with classes is to make sure everything is kept in memory only once, so there is no redundancy concerning data storage. This increases the communication between classes, like when the `net` needs to know the number of neurons, it has to ask this information to the `state`. This may seem inefficient, but when the program becomes larger it reduces greatly the probability of making errors. It also increases the flexibility, since one component can be easily replaced by another one, because all the relevant information used by other components doesn't need to be adapted because they will ask the new component for that information. Of course also hybride constructions are possible where you keep a limited amount of redundant data, which is update appropriately, to encrease performance.

1.7 Choice of the programming language

The two major requirements to be met were: 1) suitability of the language for object oriented programming, 2) the possibility to tweak the program in such a way that large-sized networks can be computed with reasonable computational resources. Now there are two major programming languages used for object oriented programming, namely Java and C++. Script languages like Perl or Phyton were not considerd since they do not compile the code and thus are slower. When it comes to tweaking a program, to squeeze the most out of your system, C++ is definitely the better choice.

It must be marked that in this project the focus was on the object oriented part and no serious attempts were made to increase the performance of the program. Nevertheless the possibility to increase the performance was taken into account when deciding which language to use. So C++ was chosen as the programming language of preference.

The next step was to look into the literature to study the different types of networks.

Chapter 2

LITERATURE STUDY

In this chapter first some general statements about neural networks are made, after which a more detailed description is given of the Hopfield model, the Q-Ising model and the Blume-Emery-Griffiths (BEG) model.

2.1 General description of a neural network

Below a description of a neural network is given in most general terms, following the main concepts.

2.1.1 Neuron state

A neural network is comprised of N neuron states, σ_i. They can take on a value from the set of possible activation states S, thus $\sigma_i \in S,\ i = 1, ..., N$. All these neuron states together describe the current network state, which is also called the pattern of activity or simply pattern.

2.1.2 Patterns

The associative memory problem basically comes down to storing a set of p patterns ξ_i^μ, which are possible network states. When patterns are successfully stored, they are also called condensed patterns. This is done in such a way that when the network is presented with a new pattern, it responds by producing whichever one of the condensed patterns that most closely resembles the new pattern. Often this new pattern is a slightly modified version of one of the stored patterns, namely the target patten. This means that the network will correct errors, so we can say that the target pattern ξ_i^μ is an attractor with a certain basin of attraction. The higher the number of patterns that were stored, the higher the load parameter $\alpha = p/N$ and the smaller the basin of attraction will become.

Remark

Because the Hamiltonian is symmetric for ξ_i^μ with regard to $-\xi_i^\mu$, which makes that both states share the same energy, it may happen that a network, for certain initial conditions , evolves to the exact opposite of the target pattern, called the **reversed state** $-\xi_i^\mu$. There are also stable **mixture states** ξ_i^{mix}, which are not equal to any single pattern, but instead correspond to linear combinations of an odd number of patterns.

2.1.3 Dynamics

As described above a neural network is a dynamic entity. Its state evolves over time. At each timestep an update takes place such that one or more neuron states are updated.

The updating can be done either synchronously or asynchronously. In the synchronous update all units are updated within the same time step. In the asynchronous case one can either select one random unit at each time step which will be updated, or one can let each unit decide at random wether or not it will be updated in that particular time step. These two last cases come down to the same scenario except for the distribution of update intervals.
In order to determine the new value for the neuron state, a transferfunction g is used. The arguments of this function are the local field h_i and the bias θ_i. The local field generally corresponds to the weighted sum of all the signals arriving through the dendrites in the cell body.

$$h_i(t) = \sum_{j \neq i} J_{ij} \sigma_j(t) \qquad (2.1)$$

These signals are weighted since the signals received by the cell body are attenuated when they have to travel a long way through the dendrites.
Every neuron has its own threshold which is translated into the bias θ_i.

$$\sigma_i(t+1) = g[h_i(t), \theta_i] \qquad (2.2)$$

So it is clear that the evolution of the network is determined by the weights and the bias.

2.1.4 Learning

In order to make the network evolve the way it is intended the weights must be chosen with care. During the learning phase the weight matrix J of the network is calculated from the patterns that have to be stored.
The most widely used learning rules are based on the "generalized Hebb rule". As Hebb formulated in his hypothesis in 1949, the synaptic strengths (weights) in the brain change proportional to the correlation between the firing of the pre- and post-synaptic neurons. So during the training phase all the patterns are considered and the weight between two neurons is increased according to the correlation of the states of those two neurons. The more patterns that show this correlation the greater the weight of their connection becomes.

2.2 Hopfield model

A Hopfield neural network consists of binary neuron states, $\sigma_i \in S = \{\pm 1\}, i = 1, ..., N$
For this model a pseudo temperature[2] T can be introduced The conventional way to describe mathematically the effect of thermal fluctuations in a Hopfield model is with Glauber dynamics, see Glauber (1963).

$$\Pr(\sigma_i = \pm 1) = f_\beta(h_i) = \frac{1}{1 + \exp(\mp 2\beta h_i)} \quad with\ \beta = \frac{1}{T} \qquad (2.3)$$

[2] We use a pseudo temperature; for conversion to the absolute temperature use $T_{abs} = T / k_B$ with k_B Boltzmann's constant = 1.38×10^{-16} erg/ K

This function $f_\beta(h_i)$ is a sigmoid with increasing steapness as the temperature drops. In fact this function returns the probability that the neuron state equals ±1. So if the temperature is high, the uncertainty about the outcome is high, introducing a random element into the dynamics. This random element can be seen as thermal noise. In the limit of T=0 the stochastic transferfunction reduces to a simple deterministic Sign function. Then there is no room for uncertainties, the sigmoid function is replaced by a step function.
The Hebb learning rule defines the weight matrix as follows

$$J_{ij} = J_{ji} = \frac{1}{NA^\mu}\sum_{\mu=1}^{p}\xi_i^\mu \xi_j^\mu \quad for\ i \neq j, \quad J_{ii} = 0 \tag{2.4}$$

$$\text{with } A^\mu = \frac{1}{N}\sum_i \xi_i^{\mu\,2} \tag{2.5}$$

the activity of the pattern μ denoted as A^μ.

Energy function
As introduced by Hopfield in 1982, an energy function can be defined whenever there is a symmetric weight matrix.

$$E = -\frac{1}{2}\sum_{ij} J_{ij}\sigma_i\sigma_j \tag{2.6}$$

The i = j term just adds a constant value to the energy, and if one chooses the self-coupling weights $J_{ii} = 0$ they don't contribute at all. It is constructed in such a way that it always decreases or at least remains constant when the system evolves according to its dynamical rule, as can be seen on the following figure.

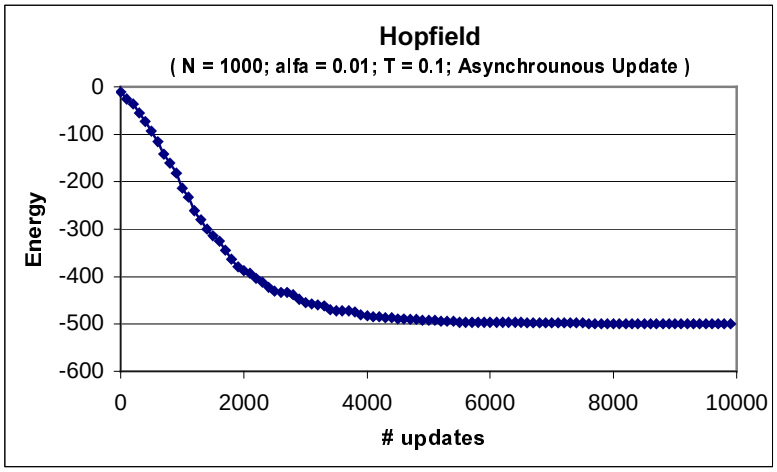

Figure 3: The energy versus the number of updates for the Hopfield model

Under retrieval conditions a target state is retrieved because it represents a local minimum and thus attracts the network. When the temperature becomes too high however, it actually causes the "melting" of the condensed state. Then there is so much thermal noise that the network jumps out of the local minimum.

2.3 Q-Ising model

There are several types of Q-Ising models like: extremely diluted or fully connected, symmetric or asymmetric and even layered feed forward. The neuron states can take values σ_i from a discrete set S, $\sigma_i \in S = \{-1 = s_1 < s_2 < ... < s_Q = +1\}$

The learning rule is the same as for the Hopfield model. The transferfunction however, is only implemented in its deterministic form.

2.4 BEG model

The Blume-Emery-Griffiths model requires ternary neuron states and the definition of two additional concepts, namely the pattern η_i^μ and the weight matrix K_{ik}. They are defined as follows,

$$\eta^\mu = \frac{\left(\xi^\mu\right)^2 - A^\mu}{A^\mu\left(1 - A^\mu\right)}, \qquad (2.7)$$

$$K_{ij} = K_{ji} = \frac{1}{N}\sum_{\mu=1}^{p} \eta_i^\mu \eta_j^\mu \;\; for\; i \neq j,\;\; K_{ii} = 0. \qquad (2.8)$$

The quadratic pattern is used extensively in the calculation of the activity overlap l (see paragraph 2.5 for the definition) and the weight matrix. In order to have a general transferfunction for any integer value of Q a stepfunction is defined. This is a deterministic function corresponding to zero temperature.

$$g[h_i, \theta_i] = sign(h_i)\Theta\left(|h_i| + \theta_i\right) \quad \text{with } \theta_i = \sum_j K_{ij}\sigma_j^2 \qquad (2.9)$$

and h_i as defined in (2.1).

2.5 Network quantities

A short overview of the different quantities that can be calculated for the various networks is given here.

The load parameter $\alpha = p/N$ allows to compare the capacity of networks of different sizes. When α exceeds its critical value α_c, called the critical load, recovery is no longer possible. The overlaps m and l are defined as follows

$$m^\mu = \frac{1}{NA^\mu}\sum_i \xi_i^\mu \sigma_i \quad l^\mu = \frac{1}{N}\sum_i \eta_i^\mu \sigma_i^2. \qquad (2.10)$$

The activitiy of the neurons denoted as q,

$$q = \frac{1}{N}\sum_i \sigma_i^{\,2}, \qquad (2.11)$$

the activity of the pattern μ, A^μ, was already defined in (2.5).

Chapter 3

IMPLEMENTATION

In order to implement the network, an effort was made to divide the concept of a network into subconcepts. This was done to enable the user to assemble his own kind of network out of different available components. It also provides a very good basis for comparative research since one only has to change the component which is under investigation while the rest of the program remains exactly the same.

3.1 General implementation issues

The neuron states are considered to be elements of a finite set S with extrema –1 and +1. They were implemented as `doubles` since they can have fractional values.

The network states can be loaded from textfiles describing bitmap images or they can be generated randomly, in such a way that they are composed out of a collection of independent and identically distributed random variables (i.i.d.r.v.), ξ_i^μ drawn from a set S of possible states with zero mean, $E[\xi_i^\mu]$, and variance $A = \mathrm{Var}[\xi_i^\mu]$. This variance A is a measure for the activity of the patterns. This was done because the main application of this code will be the validation of theoretical results with i.i.d.r.v, calculated with techniques from statistical mechanics.

Normally one would like to store a collection of p `states` (contained within an object `pattern`) into a network and select one of these `states` as the target. Then the `state` of the network is set to an initial value satisfying certain constraints like an initial overlap m_0 with the target. This is done to increase the probability to reach the intended target state.

In this project only the fully connected models with symmetric couplings were considered. Meaning that the weight matrix is symmetric, thus $J_{ij} = J_{ji}$. However the code was written in such a way that extension to the other types should be quite straight forward.

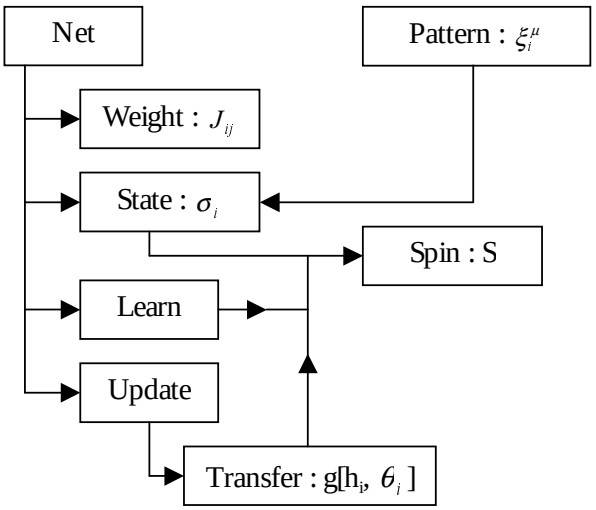

Figure 4: A schematic presentation of the relations between the different classes. The arrows represent pointers.

3.2 Implementation of a generic network

The neural network and the patterns were considered as the two basic concepts. The patterns are comprised out of `states`, which are of the same type as the state of the network. Such `states` (cfr. network states) are build up out of N `spins` (cfr. neuron states). The network has several other components like the weight matrix, a learning rule and an update rule, in which the transfer function is defined. The relations between the different classes corresponding to the components is made clear in figure 4.
In what follows the different classes will be discussed in detail.

3.2.1 Spin

The `Spin` class is the base class of several other derived classes, that is why it has been made an abstract class. Its virtual functions are `getQ(), getRandom()` and `format()` these functions have to be implemented in any child before you can instantiate an object from them. The member function `getQ()` returns the number of possible spins, which corresponds to the size of the set S from which the possible neuron states are drawn. `getRandom()` returns a random element out of the set S. The function `format(x)` was mainly used to convert bitmap color values to the corresponding neuron states. It first scales the bitmap color values to the range [-1, +1]. Then these scaled values are clipped to the nearest neuron state s_i.
This class makes it particularly easy to change the number of possible spins since you only have to replace the `spin` object. There are three child classes implemented namely `BinSpin` for binary spins, `Qspin` for any value of Q and `TerSpin` for ternary spins. There is a special class `TerSpin` because many things could be implemented more easily for Q=3 like for example a variable activity and mean which can be set at will. The implementation of this activity and mean is done by keeping track of the probability distribution of the neuron states. It can be easily visualized as follows.

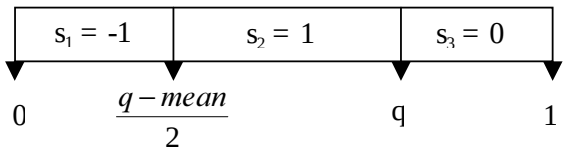

Figure 5: Graphical representation of the way non-uniform ternary patterns are implemented. To get a random state, a random number between 0 and 1 is generated and the corresponding micro state s_i is returned.

Now if one picks a random number between 0 and 1 then $p(s_1) = (q\text{-mean})/2$, $p(s_2) = (q+\text{mean})/2$ and $p(s_3) = 1\text{-}q$. This method is easily generalised to Q>3 and is implemented as such.

3.2.2 State

The concept of a network state as used in this project corresponds to a collection of N `spin` instances. So if we are dealing with binary `spins` a `state` would be an ordered sequence of –1 and +1 's which corresponds to a certain state of the network. On the one hand the class `Pattern` holds a pointer to an array of p `states`; they form the patterns ξ_i^μ that have to be learned. On the other hand the network has a pointer to the same class `State` which represents the current state of the network.

Initializing the initial state

For the initialization of the initial state of the network two different procedures were implemented namely `getBeginState` and `getBeginStateBEG`.
In `getBeginState` a beginstate with a certain deviation (thus with m = 1- deviation). First the target state is copied into the network state, which is then adapted by inserting random changes until m satisfies the desired value. In this way the activity of the pattern

is preserved since this is implemented in the `Spin` class and the random changes are received from a `spin` object.
In `getBeginStateBEG` matters are a bit more complicated since both conditions for m and l have to be satisfied. Once again the target is copied into the network resulting in maximum values for m and l close to 1. Then first the values for both m and l are decreased until one of them reaches its desired value. This is done by changing a nonzero neuron state into a zero. Then m is decreased seperately as long as required by changing nonzeros into their opposite values and l is decreased by transforming zeros into nonzeros.

3.2.3 Pattern

The class `Pattern` is in fact a container for p `State` objects. These `states` can be loaded from textfiles describing bitmap images or they can be generated randomly from a collection of independent and identically distributed random variables (i.i.d.r.v.), ξ_i^μ drawn from a set S of possible states with zero mean, $E[\xi_i^\mu]$, and variance $A = \mathrm{Var}[\xi_i^\mu]$. This variance A is a measure for the activity of the patterns.
For the BEG model a second `pattern`, called denoted by η_i^μ must be computed in order to calculate the activity overlap l and to learn the `pattern`. Therefore a kind of copy constructor was implemented which creates the corresponding η_i^μ for the given pattern.

3.2.4 Update

For both the asynchronous and the synchronous update a child class was derived from the `Update` class, named `UpdateSynch` and `UpdateAsynch` respectively. They implement the virtual function `apply` both for calls with one `weight` and a `state` object or two `weights` and a `state` object. The second case is used for the BEG model where besides the weight J_{ik} and the current network state σ_i also the weight matrix K_{ik} is required.

3.2.5 Transfer function

Four different transfer functions were derived from the abstract class `Transfer`. The class `TransferSigmBin` implements the temperature T for the Hopfield model, according to the Glauber dynamics as described in paragraph 2.2. For the limit case of T = 0, every stochastic aspect disappears and the transferfunction becomes a stepfunction which is implemented in `TransferStepBin`.
For the BEG model `TransferStepBEG` was designed where both the local fields h_i and θ_i are fed into the transferfunction

3.2.6 Learning

There is an abstract class `Learn` which has one child namely `LearnFC`, which implements the Hebbian learning for fully connected networks. Two overloaded virtual functions were defined. They both have the name `apply` but the distinction is made via the arguments passed. The first version simply applies the learning rule for the Hopfield and the Q-Ising model and requires a `Pattern` and a `Weight` object as argument. The second version requires two `Pattern` and two `Weight` objects as argument and is used for the BEG model.

Chapter 4

APPLICATIONS

In order to demonstrate the flexibility of the code developed some more advanced applications were created. The first one is a demonstration program with immediate visualisation of the results, the other one is designed to handle bitmaps.

4.1 Demo for associative memory

In order to facilitate the user input for the many parameters that can be set, an interface program was created in Visual Basic. It makes use of the code that was created during this project through communication with a dll-file.
Several procedures were written that create data for a certain configuration with the C++ code and afterwards these data are read into Visual Basic where it is displayed. All the patterns generated are a collection of independent and identically distributed random variables (i.i.d.r.v.). The different procedures will be discussed shortly.

4.1.1 m versus the temperature

As discussed, for a Hopfield model one can define a pseudo temperature. When this temperature is increased the phenomenon of melting of condensed states appears. This means that the thermal noise becomes large enough to push the network out of the local minima formed by the target states. This can be visualised by plotting the overlap after a number of updates against the temperature. Parameters that can be adjusted are N, type of update, number of updates, α, m^1_0, the maximum and minimum temperature and the number of samples that have to be taken in between those two extrema.

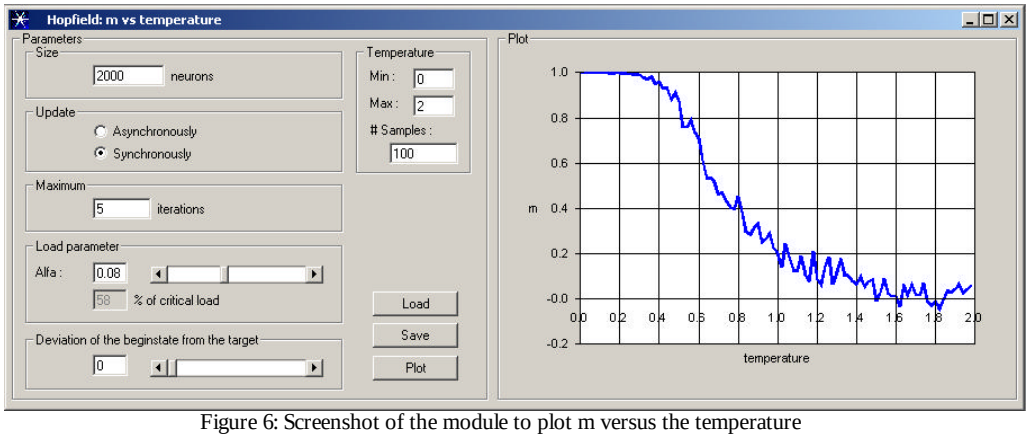

Figure 6: Screenshot of the module to plot m versus the temperature

4.1.2 m versus the number of iterations

For all three types of networks, Hopfield, Q-Ising and BEG, the overlap m is plotted against the number of iterations. On each plot one can define a range of initial overlaps with the target state in order to investigate the magnitude of the basin of attraction. Often the presence of mixture states can be detected too. The parameters that can be set are N, type of update, α, the range and number of samples for the overlap m^1_0, the number of iterations and samples together with some parameters specific to the network type like T, Q, and b.

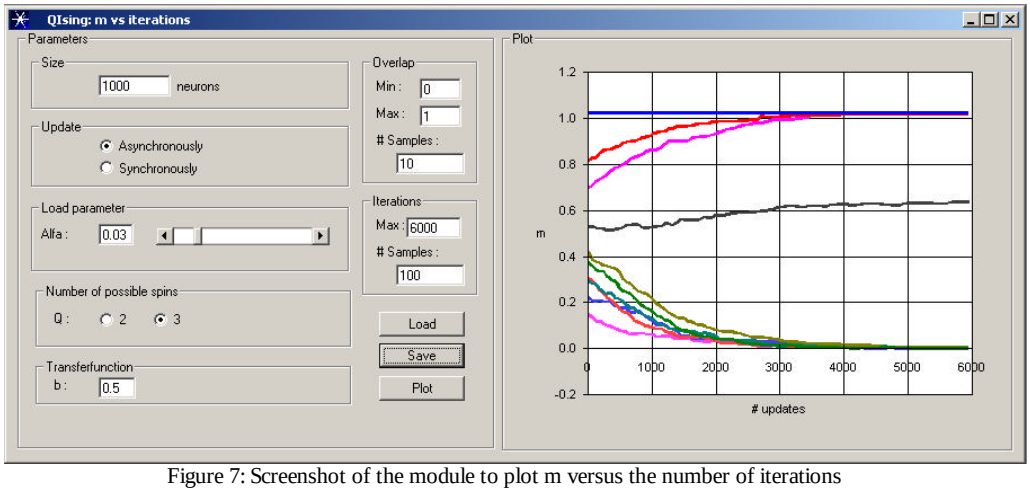

Figure 7: Screenshot of the module to plot m versus the number of iterations

4.1.3 m versus α

For Q-Ising and BEG, the overlap m is plotted against the load parameter α. On each plot one can define a range of values for α. The parameters that can be set are N, m^1_0, type of update, and the number of iterations together with some parameters specific to the network type like Q, and b.

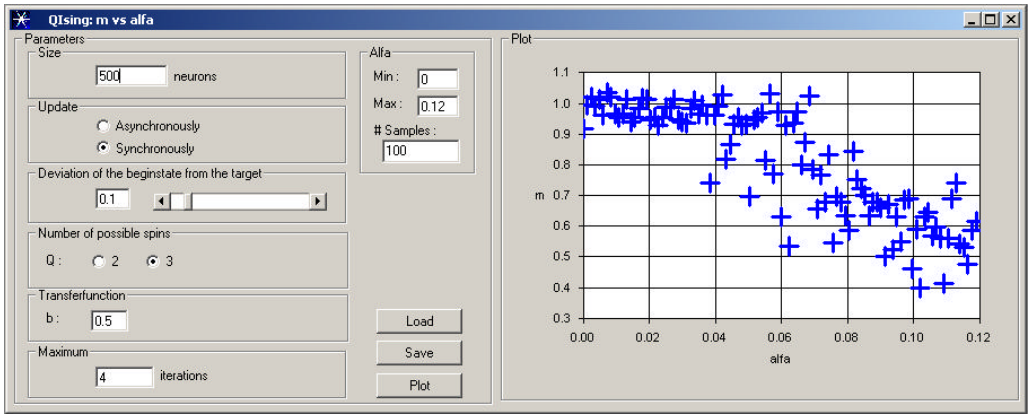

Figure 8: Screenshot of the module to plot m versus α

4.1.4 l versus m

For Q-Ising and BEG, the overlap l is plotted against the first order overlap m. One can define a region of starting positions to which different networks will be initialized and their evolution is followed as a line in the m-l plane. The parameters that can be set are N, α, type of update, and the number of iterations together with some parameters

specific to the network type like Q, and b. In this way one gets an idea of how long it takes to converge to a stable point and how large the basin of attraction is for these points.

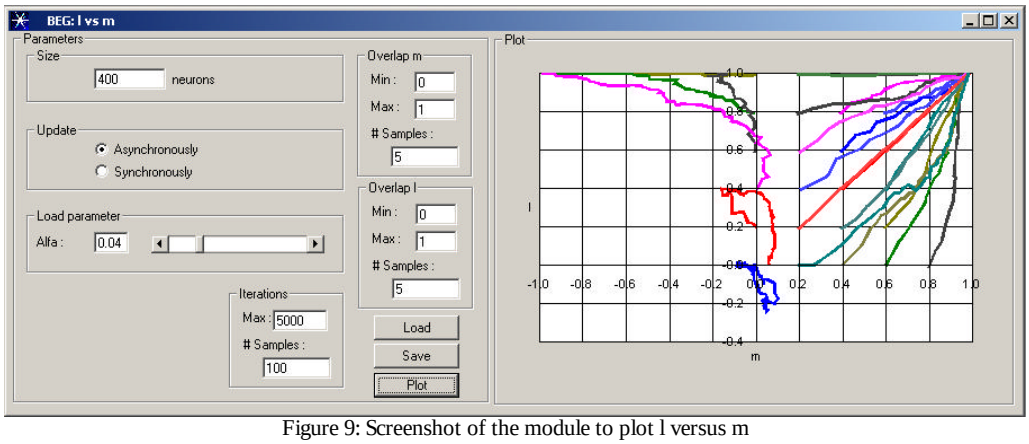

Figure 9: Screenshot of the module to plot l versus m

4.2 Visualisation of associative memory

This application was created in order to visualise the associative memory. Because humans do not have great affection for the random patterns generally used during this project, the storage of bitmaps and the recovery of them was implemented. Such patterns have in general a much lower information contend then random patterns. Between patterns the correlations are much higher. This makes that the whole dynamics will be quite different from that for random patterns.
Code was written to transform bitmaps into textfiles and vice versa. This was done by the use of an external library called SDL.

4.2.1 SDL

SDL stands for Simple DirectMedia Layer and it is a free cross-platform multi media development API created by Sam Lantinga. In this project only a very small part of the library was used.

4.2.2 Benefits of visualisation

When working with real images it might be much easier to detect certain phenomena. A first example computed with the visualisation program was already given in figure 2 in paragraph 1.3 from the introduction, where the image of Einstein was recovered from partial or noisy information. Several experiments were done on the BEG model with the four pictures A, B, C and D, shown in figure 10.

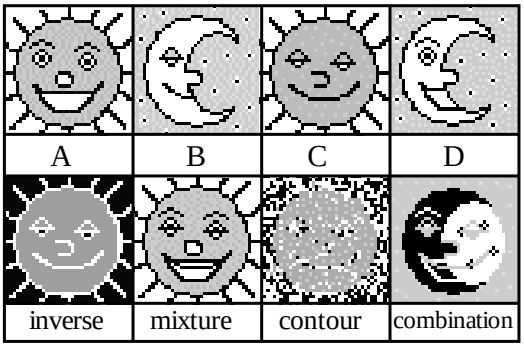

Figure 10: Simulation results for the BEG model. The 40 x 40 pixel patterns A to D were learned. The second row of figures shows some results obtained like an inverse state, a mixture state, a contour recovery and a combination of the above.

They were chosen in such a way that mixture states would be easily detectable because of very high correlation between the pictures A to D. And indeed the mixture states could be easily detected. Also the inverse state as a solution was found several times. The picture titled “contour” shows the state in which there is already good recovery of the grey values (spins = 0) but not yet for the black and white (spins = ± 1), . The last one called “combination” shows a combination of the phenomenons described above.
In practice it is difficult, if not impossible to obtain uncorrelated and identically distributed patterns with a visual meaning for humans. This makes that such pictures are not the preferred means to investigate Hebbian learning because this learning rule is not well suited for correlated patterns. However there is a great variety of more advanced learning rules that are able to deal with correlated patterns. Anyway, it must be clear that visualisation can shed some light on the different mechanisms at work in neural nets as was illustrated in figure 10.

Chapter 5

RESULTS

All numerical simulations were done with randomly drawn states and a uniform distribution of the patterns. For Q-Ising this means that A = 1 for Q = 2 and A = 2/3 for Q = 3.

5.1 Critical temperature

For the Hopfield model the critical temperature was investigated. Below the critical temperature the target state can be recovered, above T_c this is no longer possible. This phenomenon corresponds to a phase transition. It is like all the stable states have "melted" and only thermal noise remains.

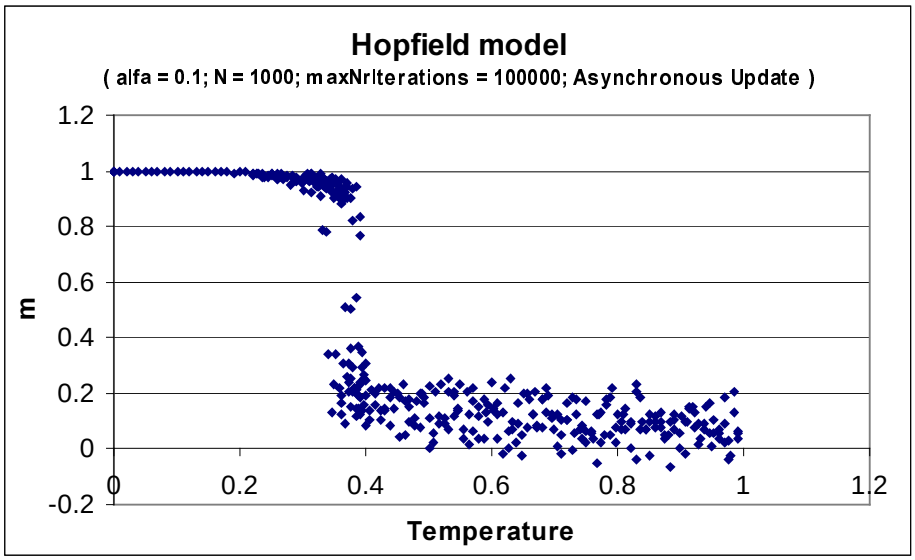

Figure 11: Plot of m versus T for a Hopfield model with a load of 0.1

As can be seen in the phase diagram in figure 12, the critical temperature for a load of 0.1 should indeed be around 0.35.

5.2 Phase diagrams

Phase diagrams can be a very valuable tool for investigating the properties of neural networks. In the following the phase diagrams for the different models are described and they were used to test the code by simulating a number of points.

5.2.1 Hopfield

The theoretically calculated phase diagram for the Hopfield model is shown below, as can be seen the value for the critical load is $\alpha_c \approx 0.138$. In the figures in appendix the different plots (of m versus the number of iterations) are given for the sample points A to H. In these plots 10 different runs can be seen for a network that was initialised with values for m_0 within a range from 0 to 1.

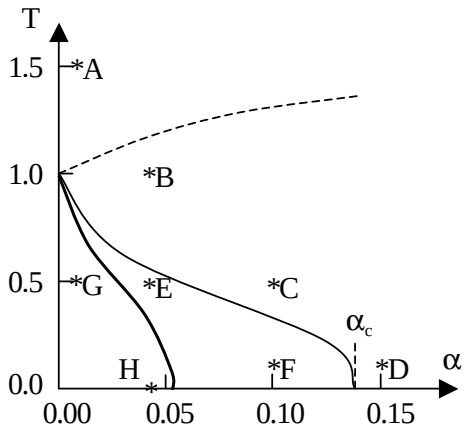

Figure 12: Phase diagram for the Hopfield model. The letters refer to points that were investigated with a plot of m versus the number of iterations

The dashed line in figure 12 represents the transition line of the spin glass phase, above this line no spin glass states are present according to theoretical calculations, only thermal noise can be found in this zone. The plot for point A indeed only shows thermal noise. Below this spin glass border line spin glass states should be present, however it is very difficult to show their existence with simulations, so they will not be considered any further. So qualitatively all the plots of A through D are the same. But one can see an interesting evolution related to the temperature. The lower this temperature gets, the slower the network evolves. It is harder to get out of the local minima, when there is less stochasticity causing random fluctuations. In the plot for point D one can see horizontal parts in the evolution curves, corresponding to the network spending some time around a local minimum.

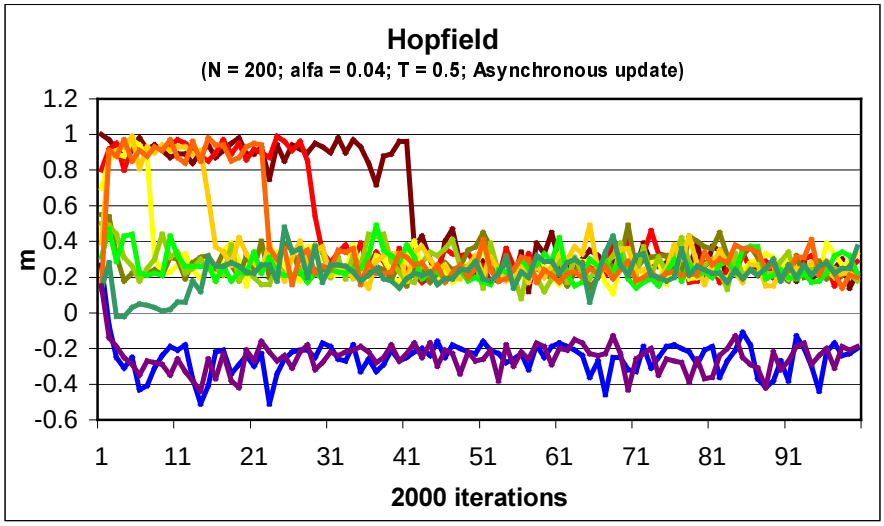

Figure 13: Plot of m vs number of iterations showing that the spin glass states are more stable then the retrieval states in point E from figure 12.

The border of the retrieval zone is indicated by a thin full line. In this zone the recovery of the target pattern is possible. Points E and F are located in this retrieval zone, but the spin glass states are more stable then the condensed states. This can be observed by looking at the evolution of networks with small initial overlap. They tend to be attracted by such spin glass states resulting in zero overlap. In order to show that the retrieval states are not the most stable solutions in this zone, a simulation was done resulting in figure 13. Here jumps from a retrieval state towards a state with zero overlap can be observed several times. These states with zero overlap are most probably such spin glass states.
Finally the points G and H lie into the retrieval zone where the condensed patterns are the most stable. Besides perfect retrieval one can see also the effect of thermal noise, visible in the lower then 1 maximum value in G compared to the perfect retrieval in H.

5.2.2 Q-Ising

For the Q-Ising model Q = 3, the theoretical b versus α phase diagram looks as follows.

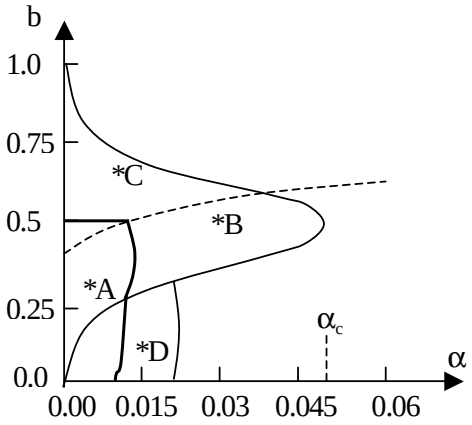

Figure 14: Phase diagram for the Q-ising model. The letters refer to points that were investigated with a plot of m versus the number of iterations

The thin dashed line indicates the spin glass transition, the thin full curve the maximal storage capacity and the thick full curve the thermodynamic transition of the retrieval state. This diagram was verified with simulations in points A to D with an initial activity of 2/ 3, the figures can be found in the appendix.
In point A we find according to previous simulations by Bollé et al. that networks with $m_0 > 0.33$ quickly evolve to the target state. In comparison with the Hopfield model however the basin of attraction is much smaller. In point B which is located above the thermodynamic transition line, a value of $m_0 > 0.6$ is required to obtain full recovery. The boundary between the m=1 and the zero attractor is less sharp as can be seen from the two evolution paths hanging in between, the fact that the retrieval state is no longer the most stable one could only be inferred from the presence of much more stable states with $m \neq 0$. Jumps were not observed since T = 0. Point C lies in the spin glass phase. The global behaviour is similar to what has been found for points A and B, except that the basin of attraction is even smaller. For point D full recovery is obtained for almost any value of m_0.

5.2.3 BEG

The BEG neural network has a much greater capacity then that of the Q = 3-Ising model, theoretically it should be almost twice as large at T = 0, namely 0.09.
An indication for this can be derived from the figure on the right. The quantitative values are hard to derive but the qualitative difference is clearly present as can be seen on figure 15.

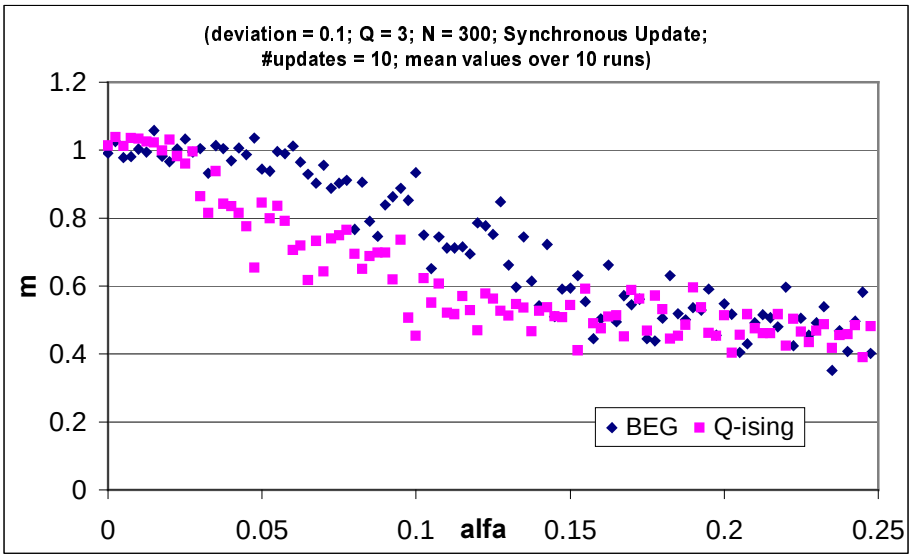

Figure 15: Qualitative difference in the capacity for the BEG and Q-Ising model

Since for the BEG model no temperature was implemented yet, only the points A through C could be investigated. The results can be found in appendix, showing total recovery for point A regardless of the initial overlap.

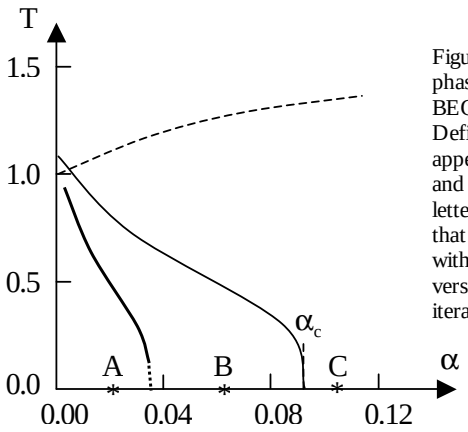

Figure 16: Preliminary phase diagram for the BEG model. Definitive version to appear by Bollé D. and Verbeiren T. The letters refer to points that were investigated with a plot of m versus the number of iterations

In point B, located above the thermodynamic transition line there is still some recovery possible, but other spurious states are more stable which can be seen clearly in the plot for point B in the appendix. In point C lying past the critical load, there is no retrieval possible at all.

5.3 Flow diagrams

In a flow field one can visualize the evolution of a network as a path in 2-dimensional space for Q = 3. The axes correspond to the first order overlap m and the overlap l. The code developed during this project allows the user to initialize a network with values for l_0 and m_0 such that the basin of attraction can be investigated. The retrieval time or speed of convergence can be easily calculated as well.

5.3.1 Q-Ising

A flow diagram is constructed by initialising the network to 36 different beginstates which cover the whole spectrum of possible values for l and m. Then they are allowed to evolve for a number of updates. For each network the path in the lm-plane is plotted.

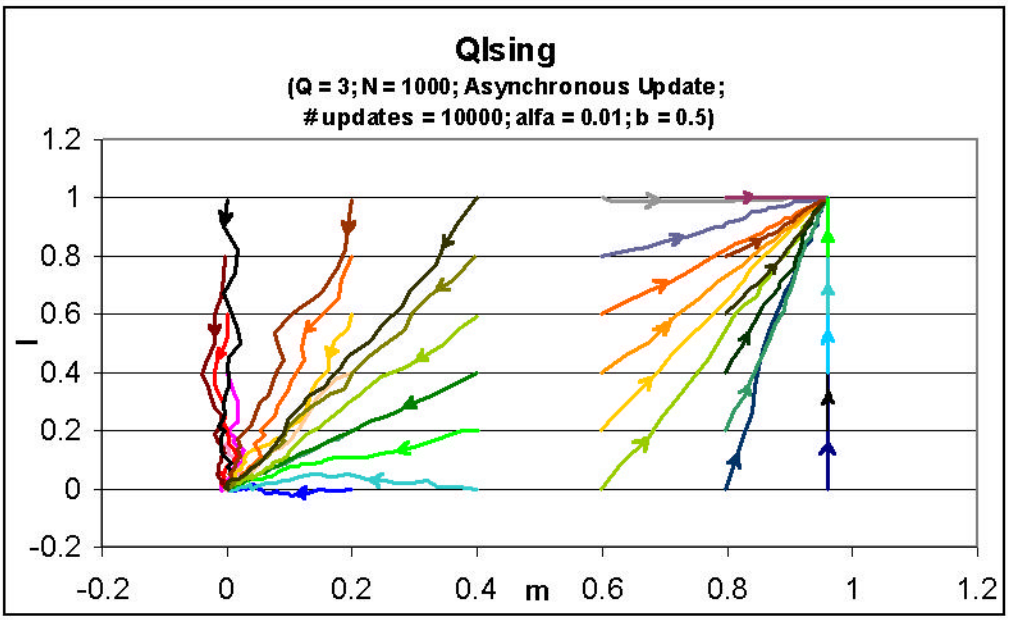

Figure 16: Flow diagram for Q-Ising, the initial positions were laid out on a grid covering the entire range of l and m values. The retrieval state and the zero attractor are clearly present

It is very clear that there is full retrieval as long as $m_0 > 0.6$. In the retrieval phase the network follows a straight path towards the point of full retrieval (1,1). A close up of the region for $0.4 < m_0 < 0.6$ can be found in the appendix.

There is also a figure in the appendix displaying no retrieval ($\alpha = 0.1$). The dynamics for m close to 1 show that the network always starts climbing upwards along the l axis , even in a non-retrieval phase, after which it slides down along a common path which is described by all other networks failing to retrieve the target.

5.3.2 BEG

The BEG model displays a much greater basin of attraction as can be seen on figure 17.

For networks with an initial state with m close to zero, there still can be recovery as can be seen on the flow field in figure 17. There is even an example of the retrieval of an inverse state (m = -1 and l = 1).

This is normal since when the network is near the point (m = 0, l = 1), it has all the information about the region in which the black and the white pixels are located, but doesn't know which value to assign to each pixel. This situation corresponds to what was called the contour retrieval in figure 10 in paragraph 4.2.2.

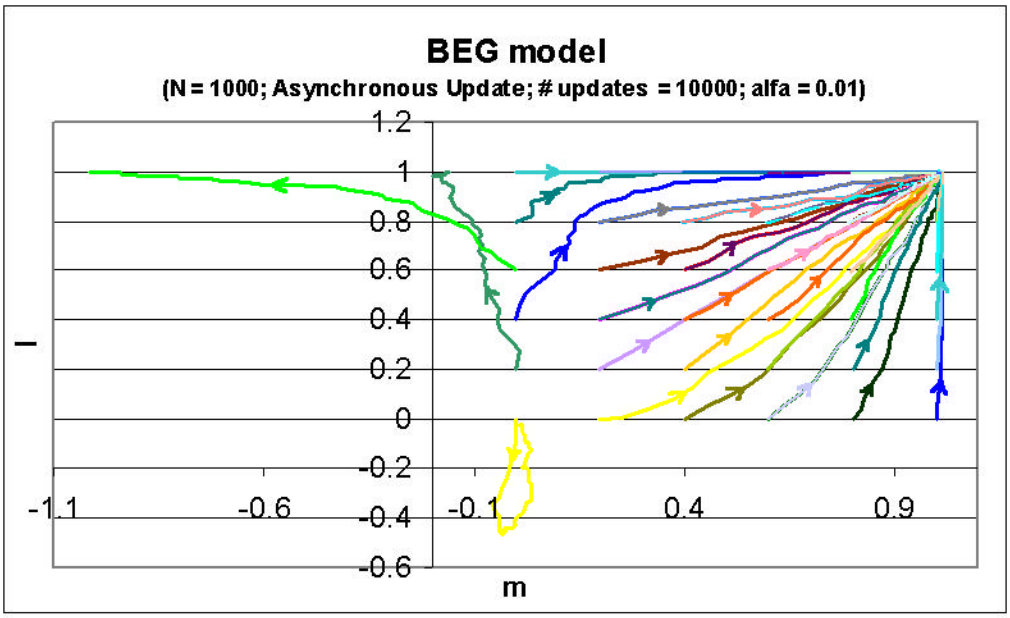

Figure 17: Flow diagram for the BEG model.

In the case of no retrieval we also observe the sliding down following a common path as can be seen in the figure in appendix.

Chapter 6

CONCLUDING REMARKS

6.1 Realised goals

As stated in chapter 2 of this report the goals for this project were:

- Writing a simulation program for the Hopfield model
- Extending the program such that it can accommodate other models (different architectures, different types of neurons, etc.)
- Applying this extension to a recent model proposed in the literature

These goals were realised successfully by writing object oriented code in C++ for the Hopfield model incorporating T, which was extended for neurons with $Q \neq 2$. The flexible structures of objects representing the main concepts involved in neural networks allowed to create new architectures in an easy way from these building blocks. The extensions were used to implement a general version of the Q-Ising network and the BEG model for $T = 0$. This code was then used to set up some simulation experiments.

6.2 Results accomplished

Various plots were made for all three models confirming the theoretical results as well as previous simulations. For the Hopfield model the critical temperature and the Energy function were investigated together with a sampling of different points in the phase diagram. For the Q-Ising, $Q = 3$, and the BEG model the phase diagram was investigated. Furthermore, flow fields were constructed showing very clearly that the basin of attraction is much greater for the BEG model in comparison with the Q-Ising, $Q = 3$, model. Visualisations of the recovery of memorised pictures were made revealing stable mixture states, the occurrence of inverse states and contour retrieval.

6.3 Further research

It must be clear that this project was only a first step in the develepment of a robust and flexible toolbox for the simulations of all kinds of associative memory networks. The results described here were merely a preliminary illustration of the potentials of such a toolbox. This toolbox was created with the foresight to make extensions to other kinds of models.
I believe this is a most promising path, definitely worth further investigation.

REFERENCES

Bollé D., Jongen G., Shim G. M., (1998). Parallel Dynamics of Fully Connected Q-Ising Neural Networks, Journal of Statistical Physics, Vol 91: 125.

Bollé D., Jongen G., Shim G. M., (1998). Q-Ising Neural Network Dynamics: a Comparative Review of Various Architectures. Proceedings of the Int. Conf. on Math. Phys. and Stochastic Analysis (Lisbon, October, 1998).

Bollé D., Jongen G., Shim G. M., (1999). Parallel Dynamics of Extremely Diluted Symmetric Q-Ising Neural Networks, Journal of Statistical Physics, Vol 96: 861-882.

Bollé D., Verbeiren T., (2002) An optimal Q-state neural network using mutual information. Preprint to appear in Phys. Lett. A.

Bollé D., Verbeiren T., (in preparation) Thermodynamics of the fully connected BEG neural network.

Dominguez Carreta D. R., Korutcheva E., (2002). Three-state neural network: from mutual information to the Hamiltonian. Physical Review E, Vol. 62: 2620-2628.

Glauber R. J., (1963). Time-Dependant Statistics of the Ising Model. Journal of Mathematical Physics, Vol 4: 294-307.

Hebb D. O., (1949). The Organization of Behavior. Wiley, New York.

Hertz J., Krogh A., Palmer R. G., (1991). Introduction to the theory of neural computation. Addison Wesley, Redwood City, 327 p.

Hopfield J. J., (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Sciences, Vol 79: 2554-2558.

Lantinga S., Simple DirectMedia Layer. http:/ / www.libsdl.org

Müller B., Reinhardt J., (1990). Neural Networks: An introduction. Springer-Verlag, New York.