A new approach to Decimation in High Order Boltzmann Machines

PhD dissertation

Student: E. Farguell (efarguell@salle.url.edu)

Advisor: F. Mazzanti (ferran.mazzanti@upc.edu)

Acknowledgments

There is a lot of people who should be addressed at this moment, and who have helped me in so many ways that I do not have enough words to thank them for their efforts. However, I would specially like to thank my parents and my fiancée for their support all over those years when it was most needed. It has been 5 years Susanna now -not actually counting from 1987 since we first met-, but it is just as the first day.

This note would not be complete if I did not thank my friends, the guys from the old Electronics Department and these great people who move around Enginyeria i Arquitectura La Salle and who I meet so often around there. I would like to thank finally Enginyeria i Arquitectura La Salle for their support over these years.

Preface

Outline

The Boltzmann Machine (BM) is a stochastic neural network with the ability of both learning and extrapolating probability distributions. However, it has never been as widely used as other neural networks such as the perceptron, due to the complexity of both the learning and recalling algorithms, and to the high computational cost required in the learning process: the quantities that are needed at the learning stage are usually estimated by Monte Carlo (MC) through the Simulated Annealing (SA) algorithm. This has led to a situation where the BM is rather considered as an evolution of the Hopfield Neural Network or as a parallel implementation of the Simulated Annealing algorithm.

Despite this relative lack of success, the neural network community has continued to progress in the analysis of the dynamics of the model. One remarkable extension is the *High Order Boltzmann* Machine (HOBM), where weights can connect more than two neurons at a time. Although the learning capabilities of this model have already been discussed by other authors [Kosmatopoulos and Christodoulou, 1994,Albizuri et al., 1995], a formal equivalence between the weights in a standard BM and the high order weights in a HOBM has not yet been established.

We analyze this latter equivalence between a second order BM and a HOBM by proposing an extension of the method known as *decimation* [Itzykson and Drouffe, 1991,Saul and Jordan, 1994]. Decimation is a common tool in statistical physics that may be applied to some kind of Boltzmann Machines, that can be used to obtain analytical expressions for

the n-unit correlation elements required in the learning process. In this way, decimation avoids using the time consuming Simulated Annealing algorithm. However, as it was first conceived, it could only deal with sparsely connected neural networks. The extension that we define in this thesis allows computing the same quantities irrespective of the topology of the network. This method is based on adding enough high order weights to a standard BM to guarantee that the system can be solved.

Next, we establish a direct equivalence between the weights of a HOBM model, the probability distribution to be learnt and Hadamard matrices. The properties of these matrices can be used to easily calculate the value of the weights of the system.

Finally, we define a standard BM with a very specific topology that helps us better understand the exact equivalence between hidden units in a BM and high order weights in a HOBM.

Contents

This memory is organized as follows: in chapter 1 a review of the historical facts that lead to the development of the original neural networks theory is introduced. In this chapter, the behavior of two of the best known neural network models that have been used along the years (the multilayer perceptron and the Hopfield neural network) is also briefly revisited. The dynamics of the BM, its extension to the HOBM model and the common learning techniques that are used on Boltzmann Machines are described in chapter 2

Standard decimation is analyzed in chapter 3. The discussion includes the full explanation on how this method works, as well as its limitations. The way all these problems are overcome is described in the last sections of this chapter. Chapter 4 discusses how a Hadamard matrix can be used to relate the weights of a HOBM with the probability distribution that it represents. In chapter 5 we present a specific BM model where high order weights find a direct equivalence in terms of second order connections and hidden units.

Finally, chapter 6 points out the conclusions that are extracted from this thesis. In the appendix we describe Hadamard matrices and the Walsh-Hadamard transform.

Contents

List of Figures

List of Tables

Chapter 1

The Neural Network

1.1 Introduction

The human nervous system is often described in terms of a powerful, parallel processor that is able to carry out sets of complex calculus in relatively short periods of time. Since the beginnings of century XIX and during century XX, scientists have explored both the learning capacity and the behavior of the human brain. The original concept behind the first artificial neural network (ANN) models was to build a model of a highly complex nervous system, motivated for the subjective human intelligence evaluation. This would be based on the learning and behavior of the human brain, which is often referred to as the *biological* neural network.

However, the enthusiasm of the first researchers who began to work in this field soon experimented a hard decay, as they concluded that it was very difficult to create an intelligent being and that the definition of *intelligence* was much broad than first expected: *the extent to which we regard something as behaving in an intelligent manner is determined as much by our own state of mind and training as by the properties of the object under consideration* -Alan Turing (1949) in [Evans and Robertson, 1968]. Further investigation during the next years established the basis of the ANN paradigm.

In this chapter we introduce the concepts of both biological and artificial neural net-

works, and discuss how the first paradigms slowly evolved into the current mathematical models. The structure of this chapter is as follows: the biological neural network is described in section 1.2, briefly reviewing how neurons work and connect to other cells; section 1.3 is devoted to describe the first models that were defined in an attempt to imitate the behavior of the biological network. This model is then compared to current neural network models such as the multilayer perceptron and the Hopfield memory.

1.2 The biological Neural Network

The expression *biological neural network* is extensively used to describe the standard central nervous system of any animal that has a structure such as a brain. In the first part of this section the standard structure of the biological neural network is described; we then proceed with a brief explanation about the mechanism used to transmit information over such systems.

1.2.1 Structure

The biological neural network of an animal allows processing the external information, taking decisions and, in essence, coordinating the behavior of the body. Though nowadays the brain is accepted as the *core* processor of the organism, it has not always been like this: ancient Greek philosophers considered it a refrigerator [Bear et al., 2006] to the emotions of the person; Roman physicians deduced that the brain controlled both thoughts and muscles, transmission was due to hydraulic movement -instead of electrical impulses-, and as the result of an effective combination of four different liquids. It was not until the Renaissance period, about 1500 years later, that those concepts arrived to a dead end. The technological and philosophic achievements of this new era brought another point of view from where to perform research: having Descartes related the concept of *soul* to the brain, closer and careful exploration through the subsequent years brought the modern ideas of *neuro-science* and *psicobiology*; which are the areas that study the human neural

structure and its relation to the human behavior.

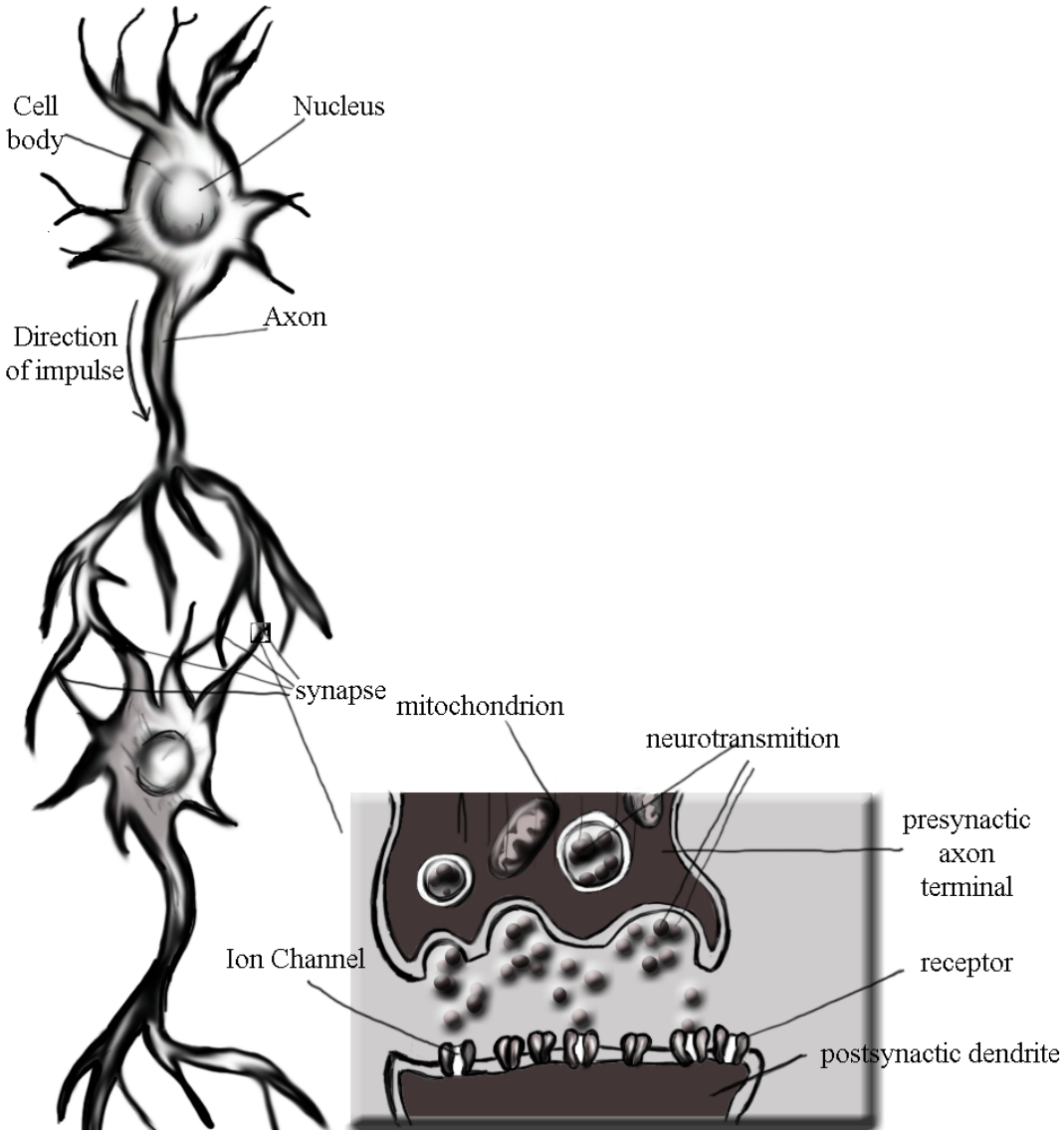

Figure 1.1: Standard neuron structure.

The modern definition of the biological neural network is due to S. Ramón y Cajal [Ramón y Cajal, 2008], who first described the basis of the neural tissue and the most important pair of cells that conform it, the neurons and the glia cells. Any other work previous to him would not consider separate cells or any smaller neural structure, rather describing the brain as a block.

A neuron is a highly specialized cell which, in essence, does one task: to transmit electrical pulses [Bear et al., 2006]. This strange ability prevents the cell from performing any other task: nutrition, protection and disposition of non-profitable materials are jobs externally provided by the glia cells. Furthermore, neurons can not reproduce themselves -thus meaning that when a neuron dies, it is not replaced. The structure of a standard neuron is depicted on Fig. 1.1. Typically, a neuron is composed of a cell body, an axon and the dendrites. The axon is used to send electrical impulses to the neighboring cells and the dendrites are their receiving terminals. The point where dendrites and axons are connected is known as the synapse, and the terminal section from the axon and the dendrites are referred to as the *presynaptic axon terminal* and the *post synaptic dendrites*, respectively. On the other hand, neurotransmitters and ion channels are the physical agents used for electrical impulse transmission. Though the walls of this cell are made of conductive materials, the main body shares many common features with other kind of cells: it includes the nucleus and the internal organs that are used to process the external proteins that feed the neuron.

1.2.2 The transmission of electrical impulses

In order to describe the electrical transmission of impulses between neurons, we must first recall how standard cells work: they are always in contact with a ionic dilution, their semi-permeable walls allowing free passage to the proteins which feed the cell. This permeability is based on a pressure equilibrium on both sides of the wall, and an excess of pressure on one side can make the wall break or poison the cell with more proteins than it is able to process -a situation that is toxic to the cell. Neurons are no exception to this architecture, and even though feeding is provided by the external glia cells, they are still surrounded by a ionic solution.

When a standard neuron is not active, its wall's potential is fixed at $V_{rest} = -60$ mV with respect to the outside [Breedlove et al., 2007]. This potential is due to its own internal ion dissolution, which also keeps a pressure such that equilibrium with the fluids

that surround the cell is guaranteed. The walls of a quiet neuron are transparent to sodium K^+ ions, hence they are free to move across the cell; the holes that ions use to come in and out of the neuron are known as *voltage active gates*, and they are described as specific proteins with the ability to change their physical configuration -thus *opening* and *closing*- when set to a given potential: this property can be activated by changing the potential at the wall of the unit. There are two different kinds of gates: sodium K^+ and potassium Na^+, the later being a *double door* gate. They both become active -this is, they change their physical configuration- when the potential at the neuron's wall reaches -40 mV.

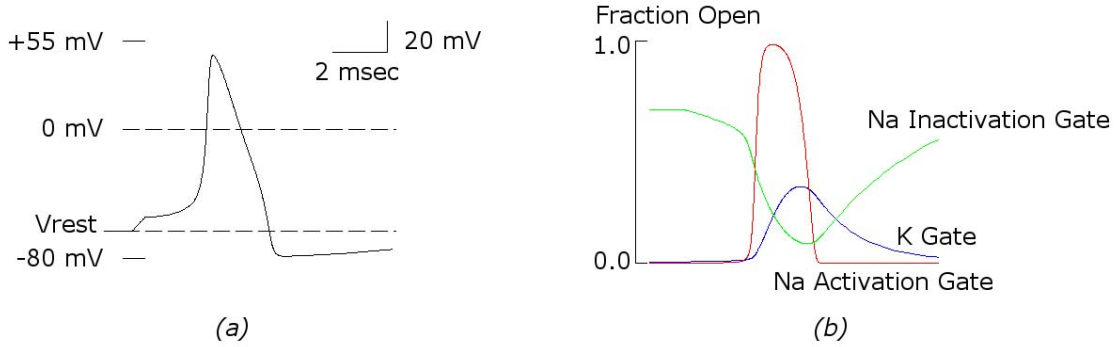

Figure 1.2: Activation process. V_{rest} is set at -60 mV, notice the time and voltage scale represented in the upper part of the image.

When the neuron is inactive, K^+ gates remain opened and ions move freely, while Na^+ gates remain closed. Activation on a neuron begins when its dendrites read a total voltage above the -40 mV threshold from their neighboring cells. The gates become now active: K^+ gates only allow ions to go outside the cell, while Na^+ gates allow ions to get inside it. These penetrate in order to equilibrate the pressure at the wall, while K^+ ions leave because they are repelled by the Na^+ ions. This same process continues until the wall reaches the *activation potential*, which is usually set at a positive value of $+40$ mV -thus meaning that there has been a $\Delta V = 100$ mV voltage difference. At this point, gates recall their original structure, and K^+ ions enter again in order to normalize the pressure between the wall and the ion dissolution. However, Na^+ ions have not yet been

expelled and the cell will have to process them; this will take some milliseconds and this is why the neuron needs some time to recover from each activation. We can see this process summarized on a picture made with the *NeuralSim* application from Ref. [Kandel et al., 1995] on Fig. 1.2: it is shown in Fig. 1.2a how the potential at the wall of the cell changes during the process, while Fig. 1.2b shows how the two doors of the Na^+ gate -which are known as *inactivation* and *activation* gate- work at different rates.

In essence, the voltage the cell is reading is somehow a combination of the potential of the other neurons; it can be seen that activation happens on a non-linear basis, and it does actually depend on the connection strengths. Even though these parameters are yet to be rigorously defined, this is the main idea behind the analytical modeling that goes from biological to artificial networks.

1.3 The Artificial Neural Network

In this section, the *artificial neural network* (ANN) is presented. According to its original definition [Culloch and Pitts, 1943], an artificial neural network is a formalism designed to emulate the behavior of a biological neural network. Since the description of the model comprises both an explanation of its dynamical rule and a topology, the first part of this section is devoted to introduce the basic concepts defining them. We also describe some of the traditionally most popular models, which are the *multilayer perceptron* and the Hopfield neural network. Next, the learning processes that have been defined for those models are discussed. We conclude the section by introducing an enhanced neural network model that no longer emulates a biological structure, but that rather appears as an evolution of the mathematical formulation of the previous models with an improved learning capacity.

1.3.1 Dynamics and Topology

The original ANN of 1943

The human brain is often seen as a multi-core processor, by considering the neurons as a combined set of units that are able to process information in a parallel asynchronous mode: a neuron is excited depending on the state of its neighbors and the way they are connected. The scientists who began their research in this field were interested on defining a mathematical model that would imitate this behavior; they would use units to represent the neurons and weights to represent their connections. A symbolic formalism was then developed by W. S. Mc Culloch and W. Pitts [Culloch and Pitts, 1943], defining a relation function between the current state of a neuron S and the cells connected to it as an unknown combination of AND and OR logical operations.

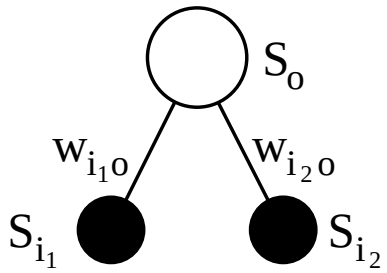

Figure 1.3: Perceptron structure.

The first quantitative representation of this logical relations was given the name of *perceptron* [Rosenblatt, 1961], and its first implementation was referred to as Mark I; this topology is depicted on Fig. 1.3. This structure considers an output unit S_o as the neuron that provides some response to the stimulation of two input S_{i_1} and S_{i_2} neurons. The output unit is connected to these by real-valued numbers known as *weights*, denoted w_{i_1o} and w_{i_2o}, respectively. Since biological neurons do only allow excited or inhibited state, units were originally conceived as *binary*, this is, they could only be 0 or 1. The state of

the output unit S_o depends on the values of the input units according to

$$S_o = \begin{cases} 1 & \text{if} \quad w_{i_1 o} S_{i_1} + w_{i_2 o} S_{i_2} \geq \theta \\ 0 & \text{if} \quad w_{i_1 o} S_{i_1} + w_{i_2 o} S_{i_2} < \theta \end{cases} \quad , \tag{1.1}$$

where θ is the threshold value that the electrical impulse has to reach in order to activate S_o. This expression can also be written in the form

$$S_o = \mathrm{U}\left(w_{i_1 o} S_{i_1} + w_{i_2 o} S_{i_2} - \theta\right) \quad , \tag{1.2}$$

$U(x)$ being the *step function*, defined as

$$U\left(x\right) = \begin{cases} 1 & \text{for} \quad x > 0 \\ 0 & \text{for} \quad x < 0 \end{cases} \quad . \tag{1.3}$$

Following the biological model, input units were defined as the external sensorial neurons that produce the input signals to be processed. However, and though this paradigm provided interesting results, it had some important drawbacks that prevented further use of the model and stopped the ANN research topic for years: this simple structure was unable to learn even some of the simplest binary relations, such as the XOR operation [Minsky and Papert, 1969]. An additional layer of neurons was later introduced to overcome this limitation. This layer processes the input signals and forwards them to the output units. This new layer could not be externally addressed and, therefore, it received the name of *hidden layer*. Input units would still be considered as an *external* excitation, hidden units were the core processor for the system and output units would show the result of the process -both input and output units are commonly referred to as visible units.

The multilayer perceptron

The multilayer perceptron has a structure that is defined by its ordering in layers; with an input layer, a set of hidden layers and an output layer; all of them possibly having different number of units. Though it is possible to define an arbitrary number of hidden

layers, a perceptron with a single layer is already able to approximate any continuous function [Cybenko, 1989, Hornik et al., 1989]. This structure is depicted on Fig. 1.4, where input units are represented as S_i, hidden units as S_h and output ones as S_o. Weights connecting input and hidden units are denoted as w_{hi} and those connecting the hidden layer to the output one are referred to as w_{oh}. It is possible to show that a perceptron with two hidden layers is able to approximate *any* function [Cybenko, 1988] with the required precision, hence it is of little use to define perceptrons with more than two hidden layers.

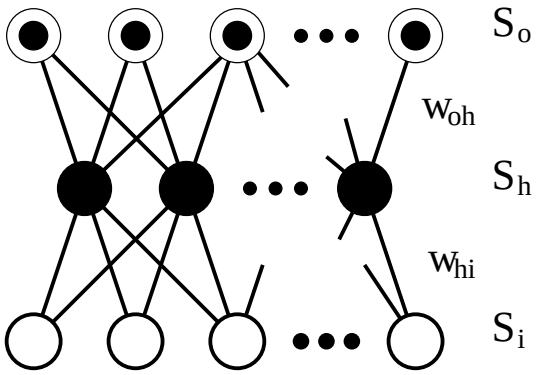

Figure 1.4: Perceptron with one hidden layer.

This model is inspired in the original, ANN model, as the signal propagation starts from the input units and flows through the neural network. The hidden layer will process the information from the input by setting the value of its units according to

$$S_{h_k} = \mathrm{U}\left(\sum_i w_{h_k i_j} S_{i_j} - \theta_{h_k}\right) \quad , \tag{1.4}$$

where S_{h_k} stands for the k-th unit of the hidden layer, θ_{h_k} being its bias term (which plays the role of the threshold θ in Eq. 1.1), S_{i_j} the j-th unit from the input layer, and $w_{h_k i_j}$ the weight connecting these units. Once this equation has been processed for every unit which belongs to the first hidden layer, this algorithm is repeated for the output units

$$S_{o_k} = \mathrm{U}\left(\sum_i w_{o_k h_j} S_{h_j} - \theta_{o_k}\right) \quad , \tag{1.5}$$

where S_{h_j} is the j-th hidden unit, S_{o_k} is the k-th output unit, θ_{o_k} is its bias term and, finally, $w_{o_k h_j}$ is the weight that connects these two units. Notice that, since neurons are

only connected to the next layer, units from the same layer do not communicate with each other; and thus the signal always propagates forward. When output units end their processing, there is no possibility for the neural network to send the information back.

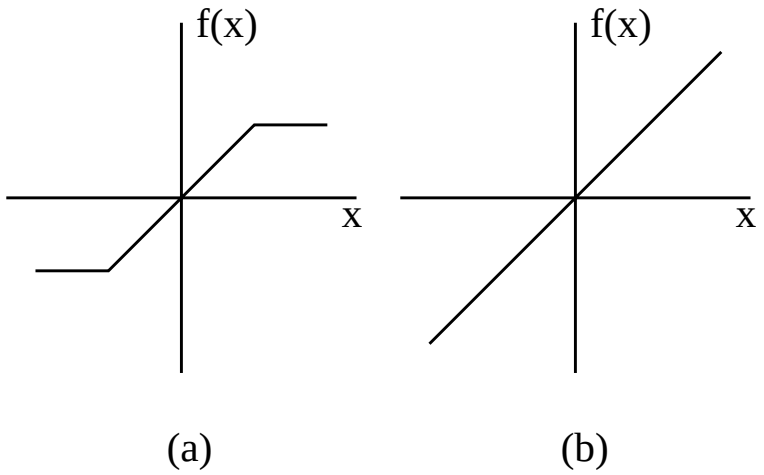

Figure 1.5: Piece-wise linear (a) and pure linear (b) functions.

In standard implementations, the step function $U(x)$ is commonly substituted by a monotonous function $f(x)$ which is known as the *activation function*. It can be linear [Hertz et al., 1991] -either piece-wise or pure, as depicted on figure 1.5- in its easiest version; but other implementations [The Mathworks^{TM}, 2008b] also use the sigmoids $1/\left(1+e^{x}\right)$ or $\tanh\left(x\right)$ and exponential-like $e^{-x^{2}}$ expressions, among others.

The Hopfield network

The Hopfield neural network [Hopfield, 1982] was originally conceived as a mathematical model for an auto-associative memory type. This device acts as a content addressable register, in the sense that it is able to recover previously learned information by providing an input signal that is similar to one of its stored patterns. This ability is specially interesting in image reconstruction and identification problems. The Hopfield model is characterized by the absence of hidden units and the fact that it is recursive, all its neurons are both used as input and output units.

The Hopfield network was taken as a simplified version of the biological neural network.

The main difference between this model and the perceptron is that the signal does not move forward on a single direction: the Hopfield neural network is in this sense fully recurrent. A set of differential equations is used to analyze its global behavior as a dynamical system [Hopfield, 1984]. Let S_i be a given unit from a Hopfield model with a total amount of N neurons. The dynamics is such that $S_i(t)$ is function of all units and is represented by

$$\frac{dS_i(t)}{dt} = -S_i(t) + \text{sgn}\left(\sum_{j=1}^{N} w_{ij} S_j(t) + h_i\right) \quad . \tag{1.6}$$

However, this expression is often solved under a discrete time reference

$$S_i(t) = \text{sgn}\left(\sum_{j=1}^{N} w_{ij} S_j(t-1) + h_i\right) \quad , \tag{1.7}$$

where $S_j(t-1)$ is the state of the other units in the previous instant and w_{ij} the weights linking units S_i and S_j. On the other hand, h_i stands for the bias term of unit S_i -thus being the equivalent θ_i from the perceptron-, and $sgn(x)$ is the sign function

$$sgn(x) = \frac{|x|}{x} \quad , \tag{1.8}$$

hence the units of the neural network are binary taking values $S_i \in [-1, +1]$.

It can be shown [Kosko, 1992] that this dynamic rule is governed by a cost function that is referred to as the *energy functional*

$$E = -\frac{1}{2}\sum_{i,j}^{N} w_{ij} S_i S_j - \sum_{i}^{N} h_i S_i \quad , \tag{1.9}$$

that has a set of global minima at some patterns $v^p = \{S_1^p, S_2^p, \ldots, S_N^p\}$, as seen in Refs. [Baldi, 1988, Abe, 1989]. The evolution in time of the network is such that, upon starting on an arbitrary vector $v^{p'} = \{S_1^{p'}, S_2^{p'}, \ldots, S_N^{p'}\}$ and arriving at a time where $S_i(t) = S_i(t-1) \ \forall i$, the system moves to the closest global minimum [Hertz et al., 1991] from Eq. 1.9. At this point, it gives as output the stored pattern v^p corresponding to that minimum.

Since the energy functional in Eq. 1.9 is a Lyapunov function [Boyd et al., 1994], it can also be shown that the system will reach convergence by running either in synchronous or asynchronous mode: synchronous dynamics imply that all units are updated at the same time, thus performing a parallel evaluation of Eq. 1.7. On the other hand, asynchronous transitions are performed by randomly selecting a unit S_i and updating its value according to the same equation until all the units remain stable.

1.3.2 Learning in ANNs

First paradigms

A biological neural network is able to learn a given pattern discerning the most relevant information from a training set of examples. The learning process increases the strength of connexions that are most used, the less used ones are weakened. The whole process is known as the *Hebb rule*, in honor to Donald O. Hebb [Hebb, 1949], and the first approaches to algorithmically simulate a learning process were inspired on this procedure. When applied to an ANN, a learning process refers to the system having some desired behaviour, which is often to reproduce a given function that is obtained through a set of vectors.

The Hebb rule was first applied as a learning rule to a perceptron model with only an input and an output layer. We will refer again to input and output units as S_i and S_o, respectively; these are connected through weights w_{io}, while the bias term for the output units is referred to as θ_o. The learning patterns are denoted $\{\xi_i^p\}$ for any input unit and $\{\xi_o^p\}$ for the output units, referred to the p-th vector of a P vectors learning set. The learning algorithm for this system begins initializing randomly the values of the weights and iteratively updating their value according to [Hertz et al., 1991]

$$w_{io}^{new} = w_{io}^{old} + \Delta w_{io} \ , \tag{1.10}$$

where

$$\Delta w_{io} = \begin{cases} 2\eta\xi_i^p\xi_o^p & \text{if } S_o^p \neq \xi_o^p \\ 0 & \text{otherwise} \end{cases} , \tag{1.11}$$

S_o^p reads as the value for a given output unit when the input units are fixed at a p vector, and η is the *learning rate*, which is tuned to carry out the learning process.

However, this learning rule can not be used on a hidden layer model, because the hidden layer has no known value *before* the learning process begins. This last issue forced the formal definition of the multilayer perceptron as a feed-forward neural network and its learning method; that is today known as *back-propagation* [Bryson, Jr. and Ho, 1969].

Learning on the Multilayer perceptron

The standard learning rule that is used nowadays on a perceptron is called *back-propagation* and was originally presented in Ref. [Bryson, Jr. and Ho, 1969]. In this algorithm a quadratic error function ε involving the learning pattern ξ_o^p and the current output state S_o^p of the network is defined

$$\varepsilon = \frac{1}{2}\sum_{p,o}\left(\xi_o^p - S_o^p\right)^2 . \tag{1.12}$$

Gradient descent is used to obtain an iterative learning procedure where weights and bias terms are randomly initialized and updated at each step of the algorithm. Weights are modified in the opposite direction of the gradient of the error

$$\Delta w_{io} \propto -\frac{\partial\varepsilon}{\partial w_{io}} , \tag{1.13}$$

$$\Delta\theta_o \propto -\frac{\partial\varepsilon}{\partial\theta_o} . \tag{1.14}$$

We first show how this rule is applied on a standard perceptron where output units S_o are related to the input layer units S_i by

$$S_o = f\left(\sum_i w_{io}S_i - \theta_o\right) , \tag{1.15}$$

where f is any monotonous function as piece-wise linear, pure linear, hyperbolic tangent, sigmoid function... as defined in Refs. [Hertz et al., 1991, The Mathworks^{TM}, 2008b].

For the w_{io} weights one has

$$\frac{\partial \varepsilon}{\partial w_{io}} = -\sum_{p,o} \left(\xi_o^p - S_o^p\right) \frac{\partial S_o^p}{\partial w_{io}}$$
$$= -\sum_{p,o} \left(\xi_o^p - S_o^p\right) \frac{\partial f}{\partial w_{io}} \ , \tag{1.16}$$

where the derivative of f will change according to the selected function. Weights are algorithmically modified according to

$$w_{io}^{new} = \eta \left(\sum_{p,o} \left(\xi_o^p - S_o^p\right) \frac{\partial f}{\partial w_{io}} \right) + w_{io}^{old} \ , \tag{1.17}$$

being η a convergence parameter to be tuned; the algorithm ends when all the weights change their value below an arbitrary small $\zeta = \left| w_{io}^{new} - w_{io}^{old} \right|$ value. When gradient descent is applied to the bias terms, one arrives at

$$\theta_o^{new} = \eta \left(\sum_{p,o} \left(\xi_o^p - S_o^p\right) \frac{\partial f}{\partial \theta_o} \right) + \theta_o^{old} \ , \tag{1.18}$$

with η being the same convergence constant as above.

This same concept is applied to find the weights connecting the different layers of a multilayer perceptron [Rumelhart et al., 1986]. However, the function must be derived with respect to the weights connecting the units from the separate layers

$$\Delta w_{ih} \propto -\frac{\partial \varepsilon}{\partial w_{ih}} \ , \tag{1.19}$$

$$\Delta w_{ho} \propto -\frac{\partial \varepsilon}{\partial w_{ho}} \ , \tag{1.20}$$

w_{ih} being the weight that links hidden and input units and w_{ho} the weights connecting hidden and output layers. This rule does also apply to bias terms

$$\Delta \theta_h \propto -\frac{\partial \varepsilon}{\partial \theta_h} \ , \tag{1.21}$$

$$\Delta \theta_o \propto -\frac{\partial \varepsilon}{\partial \theta_o} \ , \tag{1.22}$$

where θ_h and θ_o are the biases for hidden and output units, respectively.

Learning on the Hopfield neural network

The Hopfield model is a recurrent network where all units can be connected to each other, that has no hidden units and whose visible neurons act both as input and output cells. We define now a N units network and a pattern of P binary vectors $\vec{v}^p = \{\xi_1^p, \xi_2^p, \ldots \xi_N^p\}$ that the system has to learn. For this structure the Hebb learning rule [Hebb, 1949] solves entirely the problem at once, and can be directly implemented as follows

$$w_{ij} = \frac{1}{N} \sum_p \xi_i^p \xi_j^p \ , \tag{1.23}$$

$$h_i = \frac{1}{N} \sum_p \xi_i^p \ . \tag{1.24}$$

This expression grants stability for any pattern; the dynamics is expected to drive the neural network to a global state of equilibrium at $t \to \infty$ in Eq 1.7

$$S_i(t) = \text{sgn}\left(\sum_{j=1}^{N} w_{ij} S_j(t-1) + h_i \right) \ .$$

Equations 1.23 and 1.24 are compatible with the energy functional of the Hopfield model from Eq. 1.9, which is a Lyapunov function [Kosko, 1992]

$$E = -\frac{1}{2} \sum_{i,j}^{N} w_{ij} S_i S_j - \sum_i^N h_i S_i \ ,$$

this energy functional will reduce its value as the neural network evolves through time, arriving at one of the global minimum [Hertz et al., 1991]. Notice now that any previously learned vector $\vec{v}^p = \{\xi_1^p, \xi_2^p, \ldots \xi_N^p\}$ makes the energy become minimal once weights are replaced by their learning rule from Eqs. 1.23 and 1.24

$$\begin{aligned} E &= -\frac{1}{2} \sum_{i,j}^{N} \frac{1}{N} \sum_p \xi_i^p \xi_j^p S_i S_j - \sum_i^N \frac{1}{N} \sum_p \xi_i^p S_i \\ &= -\frac{1}{2N} \sum_p \left(\sum_i^N \xi_i^p S_i \right) \left(\sum_j^N \xi_j^p S_j \right) - \frac{1}{N} \sum_p \sum_i^N \xi_i^p S_i \ , \end{aligned} \tag{1.25}$$

since it is a quadratic function that is built by using the learned set of vectors. When there are too many vectors to learn for the system, it will generate spurious minimum; those

points are false solutions for the problem that differ from the real pattern. From [Baldi, 1988, Hertz et al., 1991] we know that the maximum number of vectors V_{max} that the Hopfield model can store is

$$V_{max} = \frac{N}{2 \ln N} \ , \tag{1.26}$$

this is also referred to as the *learning capacity*.

This expression is particularly interesting as it can be seen that the neural network has a relatively low capacity compared with the number of units it may have, at least if the Hebb rule is used when learning is carried out. A formal derivation of the learning capacity is found at [McEliece et al., 1987], with a full description of the capacity of the neural network for different kinds of data distributions across the learning patterns. However, final conclusions stick to Eq. 1.26 for unknown datasets. Though larger learning patterns will cause the existence of local minimum, there are some solutions that may be proposed in order to improve the learning capacity of the Hopfield model. The first one is a variation on its own learning algorithm, as proposed in [Storkey, 1997]; this maximizes ther distance between the minima of the system. However, -no matter which is the learning method used- if the Hopfield model is forced to learn a set with more vectors than units has the network -thus, $P > N$ on previous examples-, the system will be unable to achieve a stationary stable state [Abu-Mostafa and Jacques, 1985]. Another proposal is a complex definition of the states, using phasors to define the different possible states for the units [Jankowski et al., 1996, Muezzinoglu et al., 2003]. However, these kind of variations are far from the scope of this work, as we are interested on this one: weights can be modified to interconnect more than two units, even the whole network [Peretto and Niez, 1986], to create a *high order Hopfield model* with increased learning capabilities.

1.3.3 High order ANN models

A high order neural network is conceived as an evolution of a standard neural network where weights may connect more than two units at a time; even up to the total number

of units in the network. These kind of models have the advantage of using less hidden units to process the information in spite of a higher connectivity [Giles and Maxwell, 1987]. One of the first higher order models which were defined was the high order Hopfield memory [Peretto and Niez, 1986], though this topology has been also studied for perceptrons [Xiang et al., 1994] and other models of neural networks [Kosmatopoulos and Christodoulou, 1995, Kosmatopoulos et al., 1995, Pazienza et al., 2007].

From now on, we will represent these connections as a line joining the different units from the neural network. For the sake of simplicity, we shall adopt the notation from [Burshtein, 1998] and represent high order connections with the symbol $w_{\sigma}^{(n)}$; (n) stands for the number of units the weight is connecting, and σ stands for the set of indexes denoting the units that are connected by this weight. As an example, we show the high order Hopfield model, since its capacity has been deeply studied. The standard notation for one and two-body weights is

$$w_{ij}^{(2)} = w_{ij} \ , \tag{1.27}$$

$$w_{i}^{(1)} = h_{i} \ , \tag{1.28}$$

while a third order weight connecting units S_i, S_j and S_k would be referred as $w_{ijk}^{(3)}$, and depicted as in Fig. 1.6.

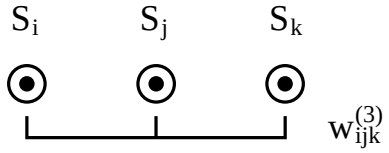

Figure 1.6: Third order weight linking units S_i, S_j and S_k.

In a Hopfield neural network, weights are invariant under permutation: second order weights satisfy $w_{ij}^{(2)} = w_{ji}^{(2)}$, third order connections satisfy $w_{ijk}^{(3)} = w_{ikj}^{(3)} = w_{jki}^{(3)} = \ldots$, and so on. The inclusion of these high order terms forces a variation on the energy functional [Peretto and Niez, 1986]

$$E = -\sum_{i} w_{i}^{(1)} S_i - \sum_{i<j} w_{ij}^{(2)} S_i S_j - \sum_{i<j<k} w_{ijk}^{(3)} S_i S_j S_k - \ldots - w_{ijk\ldots}^{(N)} \prod_{\forall \rho} S_{\rho} \ , \tag{1.29}$$

where the term $w_{ijk\ldots}^{(N)}$ links all the N units from the neural network. This expression is compacted [Burshtein, 1998] in the form

$$E = -\sum_{\sigma,n} w_{\sigma}^{(n)} \prod S_{\rho} \ . \tag{1.30}$$

The dynamic evolution of the units in the high order Hopfield model is defined as

$$\begin{aligned} S_i\left(t+1\right) = & \\ = \operatorname{sgn}\left(w_i^{(1)} + \sum_j w_{ij}^{(2)} S_j\left(t\right) + \sum_{j<k} w_{ijk}^{(3)} S_j\left(t\right) S_k\left(t\right) + \ldots + w_{ijk\ldots}^{(N)} \prod S_{\rho}\left(t\right)\right) & \ . \end{aligned} \tag{1.31}$$

The energy functional from Eq. 1.29 is still a Lyapunov function [Dembo et al., 1991] and the system will remain stable through time [Burshtein, 1998] when the following differential equation is used to simulate its behavior

$$\frac{dS_i\left(t\right)}{dt} = -S_i\left(t\right) + \operatorname{sgn}\left(w_i^{(1)} + \sum_j w_{ij}^{(2)} S_j\left(t\right) + \ldots + w_{ijk\ldots}^{(N)} \prod S_{\rho}\left(t\right)\right) \ . \tag{1.32}$$

We now prove this system to be stable; according to Lyapunov control theory [Slotine and Weiping, 1991], stability happens when the following two conditions are satisfied:

1. The energy functional $E\left(S\right)$ from Eq. 1.30, expressed as a function of the units, is bounded, and thus $E\left(S\right) \in [E_{inf}, E_{sup}]$.

2. The derivative of the energy functional $\frac{dE(S)}{dt}$ is negative or zero, hence $\frac{dE(S)}{dt} \leq 0$.

It can be readily seen that the energy functional is bounded: regardless of the values of the units, the lowest value E_{inf} that Eq. 1.30 can reach is $E_{inf} = -\sum_{\sigma,n} \left|w_{\sigma}^{(n)}\right|$, while its maximum value stands for $E_{sup} = \sum_{\sigma,n} \left|w_{\sigma}^{(n)}\right|$. We now differentiate the energy functional

$$\frac{dE\left(S\right)}{dt} = \nabla E\left(S\right) \frac{dS}{dt} = \sum_{i=1}^{N} \frac{\partial E\left(S\right)}{\partial S_i} \frac{dS_i}{dt} \ , \tag{1.33}$$

where

$$\frac{\partial E\left(S\right)}{\partial S_i} = -w_i^{(1)} - \sum_j w_{ij}^{(2)} S_j - \sum_{j<k} w_{ijk}^{(3)} S_j S_k - \ldots \ , \tag{1.34}$$

notice now that we can write Eq. 1.32 as

$$\begin{aligned}\frac{dS_i(t)}{dt} &= -S_i + \operatorname{sgn}\left(w_i^{(1)} + \sum_j w_{ij}^{(2)} S_j(t) + \ldots + w_{ijk\ldots}^{(N)} \prod S_\rho(t)\right) \\ &= -S_i - \operatorname{sgn}\left(\frac{\partial E(S)}{\partial S_i}\right) \quad , \end{aligned} \tag{1.35}$$

so

$$\frac{dE(S)}{dt} = \sum_{i=1}^{N} \left(-\frac{\partial E(S)}{\partial S_i}\right)\left(S_i + \operatorname{sgn}\left(\frac{\partial E(S)}{\partial S_i}\right)\right) \quad . \tag{1.36}$$

We now analyze the values of $\frac{\partial E(S)}{\partial S_i}\frac{dS_i}{dt}$ depending on the sign of $\frac{\partial E(S)}{\partial S_i}$. If $\frac{\partial E(S)}{\partial S_i} = -k < 0$ for $k \in \Re^+$, then

$$\frac{\partial E(S)}{\partial S_i}\frac{dS_i}{dt} = -k\left(S_i + \operatorname{sgn}(-k)\right) = -k(1 - S_i) \quad . \tag{1.37}$$

If $S_i = 1$, then $1 - S_i = 0$; on the other hand, if $S_i = -1$ then $1 - S_i = 2$ and, regardless of the value of S_i, $\frac{\partial E(S)}{\partial S_i}\frac{dS_i}{dt} \leq 0$. We conclude this proof by discussing the case where $\frac{\partial E(S)}{\partial S_i} = k > 0$ for $k \in \Re^+$. Now

$$\frac{\partial E(S)}{\partial S_i}\frac{dS_i}{dt} = -k\left(S_i + \operatorname{sgn}(k)\right) = -k(S_i + 1) \quad , \tag{1.38}$$

where $S_i + 1 = 0$ for $S_i = -1$ and $S_i + 1 = 2$ for $S_i = 1$, hence $\frac{\partial E(S)}{\partial S_i}\frac{dS_i}{dt} \leq 0$ always and $\frac{dE(S)}{dt} \leq 0$ irrespective of the value of S_i. Then, this system is stable.

This high order model increases the capacity of the Hopfield neural network [Burshtein, 1998], which for some patterns can reach

$$V_{max} = \frac{N^n}{2(n+1)\lambda_n \ln N} \quad , \tag{1.39}$$

where N stands for the units on the neural network, n is the maximum number of units that a weight is connecting and

$$\lambda_n = \frac{(2n)!}{n!2^n} \quad . \tag{1.40}$$

However, both the representation of the neural network and its learning process become more difficult as n increases, due to the large quantity of weights that the neural network

has. For an n-th order Hopfield network with N units, the total number of weights N_w becomes

$$N_w = \left(\begin{array}{c} N \\ 1 \end{array} \right) + \left(\begin{array}{c} N \\ 2 \end{array} \right) + \left(\begin{array}{c} N \\ 3 \end{array} \right) + \ldots + \left(\begin{array}{c} N \\ n \end{array} \right) \quad , \tag{1.41}$$

as for $n = N$, N_w becomes $N_w = 2^N - 1$. Hence, if we are interested in using a high order Hopfield network with a large number of units, it becomes a non-practical, though powerful, model.

Chapter 2

The Boltzmann Machine

2.1 Introduction

The Boltzmann Machine (BM) [Aarts and Korst, 1989] is a recurrent, stochastic neural network with the ability of learning and extrapolating probability distributions. Though it can be seen as an enhanced, probabilistic model that has evolved from the Hopfield [Hopfield, 1982] neural network, it was originally conceived as a parallel implementation of the Simulated Annealing (SA) [Kirkpatrick et al., 1983] optimization algorithm. In order to provide the system with an analytical learning expression, D. H. Ackley, T. J. Sejnowski and G. E. Hinton [Ackley et al., 1985] proposed a measure of the error between a probability distribution to learn and the BM own distribution -this value is known as the Kullback-Leibler [Kullback, 1959] distance.

The Simulated Annealing algorithm, whose dynamics describe the behavior of the Boltzmann Machine, is briefly explained in section 2.2. Section 2.3 is devoted to the analysis of the standard BM algorithm and its extension to the high order Boltzmann Machine (HOBM) [Sejnowski, 1987] model. Finally, the Boltzmann Machine learning equations, the main drawbacks of the BM and its standard learning solutions (as used nowadays) are discussed in section 2.4.

2.2 Simulated Annealing

The Simulated Annealing [Kirkpatrick et al., 1983] algorithm is a powerful, global stochastic optimization algorithm that numerically emulates the behavior of a given material under the process known as *annealing*. This process consists on heating it until liquid state is reached; this condition will lead to a random walk across all its feasible energetic states where it is equally possible to find its atoms on any spin direction. This material should be slowly cooled upon absolute zero, and so a perfect crystal structure would be obtained; at this point it would render on a global energetic minimum. However, since it is not possible to reach such temperature value, the quantity that is minimized instead is the Helmholtz free energy $\mathcal{F}$

$$\mathcal{F} = E - TS \ , \tag{2.1}$$

where T stands for the real temperature, E is the internal energy of the system and S its entropy.

In this section, the combinatorial optimization algorithm known as the Simulated Annealing algorithm is briefly described. The first part of this section presents the Metropolis algorithm as the original concept that led to the design of this method. It then proceeds by discussing the standard implementation for the SA algorithm.

2.2.1 The Metropolis algorithm

The Simulated Annealing was inspired by the Metropolis algorithm [Metropolis et al., 1953], which is a numerical method originally proposed as a way to simulate the behavior of a solid under a heat bath via Monte Carlo (MC) [Rubinstein, 1981] techniques: the Metropolis algorithm allows the simulation of *thermal equilibrium* situation for any ergodic physical system. It works by first proposing an initial random energetic state α with an associated energy value E_α. A transition to a new random state β is then generated; this new state will have an energy value E_β. If the quantity $\Delta E = E_\beta - E_\alpha$ is negative, state β is accepted as the new departing state. Otherwise, β is accepted with a certain

probability $p(\alpha \rightarrow \beta)$ such as

$$p(\alpha \rightarrow \beta) = \mathrm{e}^{\frac{E_\alpha - E_\beta}{k_B T}} \ , \tag{2.2}$$

where k_B is a physical constant known as the *Boltzmann constant* and T is the temperature of the heat bath. This algorithm can also be carried out if Eq. 2.2 is exchanged by

$$p(\alpha \rightarrow \beta) = \frac{1}{1 + \mathrm{e}^{\left(E_\beta - E_\alpha\right)/k_B T}} \ , \tag{2.3}$$

which is widely used in the BM literature [Freeman and Skapura, 1993]. However, it can be shown that this expression causes the algorithm to reach convergence slower [Metropolis et al., 1953]. Thermal equilibrium is reached once the average number of transitions from any given state α to any other β becomes the same [Itzykson and Drouffe, 1991], hence

$$p(\alpha \rightarrow \beta)\, p_\alpha = p(\beta \rightarrow \alpha)\, p_\beta \ . \tag{2.4}$$

Upon reaching thermal equilibrium, the probability of being on a given state α is given by the Boltzmann-Gibbs probability distribution

$$p(\alpha) = \frac{\mathrm{e}^{-\frac{E_\alpha}{k_B T}}}{\mathcal{Z}} \ , \tag{2.5}$$

where $\mathcal{Z}$ is known as the *partition function* and stands for

$$\mathcal{Z} = \sum_\gamma \mathrm{e}^{-\frac{E_\gamma}{k_B T}} \ . \tag{2.6}$$

2.2.2 The Simulated Annealing algorithm

The annealing process is a physical procedure which consists on heating a given material upon reaching a liquid state, to slowly cool it until becoming a solid structure. If the cooling process is slow enough, this solid state will have a crystal-like structure, thus reaching a state where the energy is minimum. The Simulated Annealing was born as a combinatorial optimization technique [Kirkpatrick et al., 1983] that emulated this process by multiple repetition of the Metropolis algorithm: temperature would be slowly decreased and thermal equilibrium reached at each Metropolis algorithm run.

The temperature value that is used at each Metropolis algorithm run is defined as a succession of monotonically decreasing K values. This succession of temperatures will simulate the temperature variations through the annealing process and is known as the *cooling schedule* [Aarts and Korst, 1989]. For each different temperature T_k from the cooling schedule, the algorithm must iterate until reaching thermal equilibrium. However, this is not often feasible in practical terms, and thereafter a number of iterations m_k is associated to each temperature of the cooling schedule.

Let a cost function f be a combinatorial function which depends on a given P variables vector

$$\vec{x}_i = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, \ldots, x_P^{(i)}\right) \ , \tag{2.7}$$

where

$$\vec{x}_i \neq \vec{x}_j \ , \ \forall i \neq j \ , \tag{2.8}$$

and

$$f_i = f\left(\vec{x}_i\right) = f\left(x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, \ldots, x_P^{(i)}\right) \ , \tag{2.9}$$

we will assume however that there are $N \to \infty$ possible instances of $\vec{x}_i$, and therefore it is not feasible to optimize f by exhaustive search. The Simulated Annealing algorithm will compute the Metropolis algorithm with probability

$$p = \mathrm{e}^{\frac{f_i - f_j}{T_k}} \ , \tag{2.10}$$

where j is the transition from i

$$f_j = f\left(\vec{x}_j\right) = f\left(x_1^{(j)}, x_2^{(j)}, \ldots, x_p^{(j)}, \ldots, x_P^{(j)}\right) \ . \tag{2.11}$$

Notice that the product $k_B T$ is replaced by the k-nth temperature from the cooling schedule T_k as a stochastic control parameter. This process is carried out m_k times for all the temperatures of the cooling schedule. It can be shown that for

$$T_{k+1} = T_k - \Delta T \ , \ \Delta T \to 0 \ , \tag{2.12}$$

and having achieved *real* thermal equilibrium, thus $m_k \to \infty$ at each T_k, the algorithm always finishes on a global minimum (this is, the smallest value that the function takes

in a point within its entire domain) [Aarts and Korst, 1989]. Since it is computationally exhausting to fulfill such conditions, the only statement that one can be made certain is that a good cooling schedule will lead to a minimum that is close to the global one.

2.3 The Boltzmann Machine as a Neural Network

The Simulated Annealing algorithm can also be defined as a parallel algorithm [Aarts and Korst, 1989, Younes, 1994] though in this case it is often known as the Boltzmann Machine [Ackley et al., 1985]. This is actually a stochastic neural network model whose dynamic is SA based, though we will *not* detail how the BM is used as a parallel optimization tool -modeling a given task to be suitable for a parallel BM processing is usually hard [Aarts and Korst, 1987, Koenig et al., 1992, Oyama, 1993]- but we will rather center on its behavior as neural network.

The first part of this section describes the topology of the BM and the notation that has been used in this work. The dynamics are detailed in the next section according to the two main algorithms that are used to simulate the behavior of the model: the Simulated Annealing and Mean Field (MF) algorithms. The first one is used when one is interested on reaching a given probability distribution with the neural network, because it can be used to attain a statistically exact estimation of its behavior. On the other hand, the Mean Field algorithm is an inexact, though a much faster approach. This section concludes with the introduction of the BM model known as the high order Boltzmann Machine, whose weights may connect more than two units and up to the whole network.

2.3.1 Topology of the BM

Standard units from a BM are commonly referred to as S_i, it is commonly accepted that such units may take either two valued $[-1, +1]$ [Hertz et al., 1991] or standard binary $[0, 1]$ [Freeman and Skapura, 1993] values. Though there are researchers who have developed four state complex units [Rager, 1992], quasi continuous [Lin and Lee, 1995]

or even continuous valued neurons [Beiu et al., 1992, Parra and Deco, 1993], we will stick to the standard $S_i = [-1, +1]$ definition. These units are distributed on a three layer recurrent topology, with input, hidden or output neurons that may be connected with no restrictions; notation as shown on Fig. 2.1 will be used through this work to represent them.

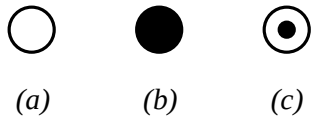

Figure 2.1: Notation for input (a), hidden (b) and output (c) units.

There are two possible structures for any BM: the *Termination BM* or the *Input-Output BM*, which are depicted on Fig. 2.2. The Termination BM is a model whose input units are also used as outputs, with the same topology as the Hopfield model with added hidden units. The Input-Output BM is a three layer structure with separate input, hidden and output layers. In this latter case, input units are assigned a value which can not change until the final output of the neural network is calculated; when a unit on a BM is not allowed to change its state it is commonly referred to as being a *clamped* unit.

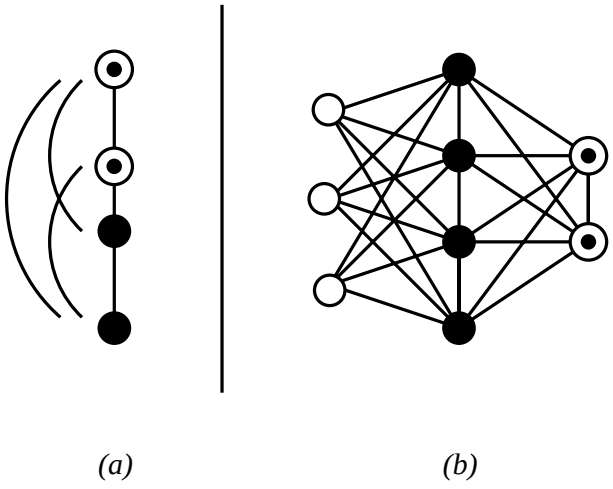

Figure 2.2: Termination BM (a) and Input-Output BM (b).

Each pair of units S_i, S_j is connected by a symmetric weight $w_{ij} = w_{ji}$, this is com-

monly depicted as a line linking both neurons. A full set of weights from a BM is numerically represented by using a real-valued symmetric matrix with zero diagonal -units can not receive feedback from themselves. It is possible for the neural network to have all its units connected to bias terms. These are written down as h_i, and they are represented as a short line starting from their respective units S_i. However, since we will be using densely connected BM models through this work, we will not use this notation. Weights linking units will be depicted as square lines that are separated from the units. In this sense, bias terms will also be considered as weights and thereafter they will be represented as short lines that will not begin at the unit. This notation is depicted in Fig. 2.3, where two biased, connected hidden units S_i, S_j are shown. Notice however that the label for each connection is written down at its side.

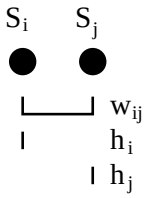

Figure 2.3: Two hidden units linked by weight w_{ij}, and biases h_i, h_j.

We finally show a two inputs neural network with a hidden and an output unit in Fig. 2.4. The input units have been labeled as S_{i_1} and S_{i_2}, while the hidden unit is referred as S_h and the output neuron as S_o. Weights linking the units are depicted in the same figure, notice then that there are no connections among the input units: since they are not allowed to change their state, these values are not needed.

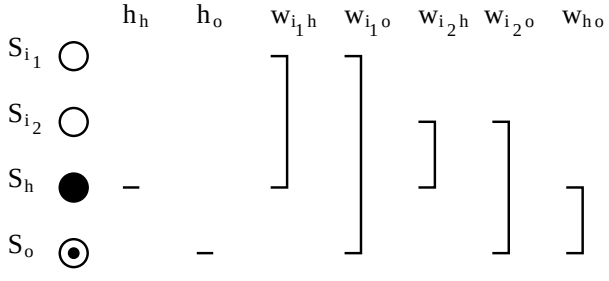

Figure 2.4: Two input units neural network with one hidden unit and an output unit.

2.3.2 Dynamics and algorithm for a BM

The dynamics of the Boltzmann Machine can be described in terms of an ergodic physical system. In this sense, the SA algorithm is used to simulate its behavior considering that the units of the neural network provide the orientation of the electrons from the system. The energy functional of the system is that of a real one

$$E = -\frac{1}{2}\sum w_{ij}S_iS_j - \sum h_iS_i \ , \tag{2.13}$$

this model is usually referred to as the Ising model [Hazewinkel and Vinogradov, 1995] from statistical physics [Itzykson and Drouffe, 1991]. This expression is dependent on the units, weights and biases of the neural network; this is the same functional as the Hopfield model. When the BM has to compute any input vector γ, input units are clamped to its values; they are unable to change their state until the simulation ends. The energy functional is then evaluated by using the SA algorithm, though it is not intended to achieve a global optimization solution: the cooling schedule is rather designed to achieve thermal equilibrium at each temperature. As a result, the states of the units change according to a stochastic dynamic until reaching a final state α for the output units, a state β for the hidden layer and the already known clamped state γ for the input neurons, thus reaching a state $\{\alpha, \beta \mid \text{Inputs} = \gamma\}$ that will result on an energy value $E_{\alpha,\beta|\gamma}$. The probability of finding the neural network in such state is given by the Boltzmann probability distribution

$$p\left(\alpha, \beta|\text{Inputs} = \gamma\right) = \frac{\mathrm{e}^{-\frac{E_{\alpha,\beta|\gamma}}{T_K}}}{\mathcal{Z}_\gamma} \ , \tag{2.14}$$

where T_K is the final temperature from the cooling schedule and $\mathcal{Z}_\gamma$ is the partition function with the input units clamped at a vector γ; this is actually a normalization constant which sums over all feasible energetic states

$$\mathcal{Z}_\gamma = \sum_{\mu,\nu} \mathrm{e}^{-\frac{E_{\mu,\nu,\gamma}}{T_K}} \ . \tag{2.15}$$

In the following, and for a clearer notation purpose, we will define

$$p\left(\alpha \mid \gamma\right) = p\left(\alpha \mid \text{Inputs}{=}\gamma\right) \ , \tag{2.16}$$

$$p\left(\alpha, \beta \mid \gamma\right) = p\left(\alpha, \beta \mid \text{Inputs}{=}\gamma\right) \ , \tag{2.17}$$

as conditioned probability distributions. Notice that the marginal probability distribution definition is used to relate $p(\alpha)$ with $p(\alpha, \beta)$

$$p(\alpha) = \sum_{\beta} p(\alpha, \beta) \quad , \tag{2.18}$$

where the sum is for all feasible combination of states that the hidden units may take.

We now describe the algorithm for a n_i input, n_h hidden and n_o output units BM. The process that is used to simulate the behavior of the BM can be described through the following steps [Freeman and Skapura, 1993]:

1. Set a given state γ to the input units.

2. Set the first temperature T_1 from the cooling schedule.

3. Select a random hidden or output unit S_i. This unit will change its state to $-S_i$ according to the SA probability

$$p(S_i \rightarrow -S_i) = \mathrm{e}^{\frac{E_{\alpha,\beta|\gamma}(S_i) - E_{\alpha,\beta|\gamma}(-S_i)}{T_1}} \quad , \tag{2.19}$$

   where $E_{\alpha,\beta|\gamma}(S_i)$ and $E_{\alpha,\beta|\gamma}(-S_i)$ correspond to $E_{\alpha,\beta|\gamma}$ when evaluated at S_i and $-S_i$, respectively. This step is carried out $n_h + n_o$ times.

4. Carry out the previous step m_1 times. Notice then that an iteration of the cooling schedule is accounted when all the units of the neural network that are not fixed at a certain value have had the opportunity of being selected.

5. Repeat this process for each T_k temperature from the cooling schedule, upon reaching the final T_K final temperature.

Since the neural network is expected to achieve thermal equilibrium, it is able to reproduce a probability distribution. The corresponding probability distribution function (p.d.f.) has to be estimated by carrying out enough iterations of the previous algorithm, because this is only a reaction to a given input instance. Since the system is on a thermal equilibrium situation, one could estimate the p.d.f. by following these steps:

6. Repeat step 3 at temperature T_K as many times as desired and store how many times a given state has been selected.

7. Estimate the vale of the probability distribution function according to the previous results.

This probability distribution reproducing ability can be used as a probability estimation tool [Kappen, 1993, Thathachar and Arvind, 1999], in the same sense that non-stochastic neural networks can extrapolate functions. Notice though that there are two probability values that the neural network is unable to reproduce, which are 0 and 1, as those would require $E_{\alpha,\beta,\gamma} \to \pm\infty$. Since it is not possible to reach such values for the energy functional, a Boltzmann Machine can not be asked to learn exact 0 nor 1 probability values, though there are techniques for approximate such patterns while learning is carried out [Hertz et al., 1991].

2.3.3 The mean field equations

The analysis of BM dynamics can however become complex when dealing with a high number of neurons. The behavior of the multiple units of the neural network can be approximated by using the mean field equations [Amit, 1989], where the BM is considered a physical system whose units are real electrons. This model then uses an interaction term w_{ij} between each pair of units; their orientation is written down as $S_i = [-1, +1]$, and an external influence is set up as h_i. These are the same terms as the biases from the neural network. The mean field equations provide the mean value of the neurons

$$\langle S_i \rangle = \tanh\left(\sum_{j=1}^{N} \frac{w_{ij}}{T} \langle S_j \rangle + \frac{h_i}{T}\right) \quad ,$$

at a certain T value which is the last temperature of the cooling schedule. This equation is used to generate a coupled system of equations that is solved by the fixed point iteration algorithm [Press et al., 1993]. Notice then that the result is always the same [Itzykson and Drouffe, 1991] for a given fixed $\{w_{ij}\}, \{h_i\}$ set of weights and biases. Due to this property,

this model it is often referred to as the *deterministic* Boltzmann Machine [Kappen, 1995]. The complete probability distribution is then computed by approximating the correlations of the system

$$\begin{aligned} &\langle S_i S_j \rangle \simeq \langle S_i \rangle \langle S_j \rangle \quad , \\ &\langle S_i S_j S_k \rangle \simeq \langle S_i \rangle \langle S_j \rangle \langle S_k \rangle \quad , \\ &\dots \end{aligned} \tag{2.20}$$

We now deduce the mean field equations, by using an approximation that is found by working out a Legendre transform [Arnold, 1997] of the Helmholtz free energy [Peterson and Anderson, 1987]. This quantity is minimal when the system reaches equilibrium at the last temperature of the cooling schedule

$$\mathcal{F} = E - TS = -\ln \mathcal{Z} \quad , \tag{2.21}$$

$\mathcal{Z}$ being the partition function of the Boltzmann probability distribution

$$\mathcal{Z} = \sum_{\mu,\nu} \mathrm{e}^{-E_{\mu,\nu}/T} \quad . \tag{2.22}$$

The Legendre transform of the Helmholtz free energy creates a new energy functional $\mathcal{G}$ that depends both on the bias terms h_i and the expected value of the given units $\langle S_i \rangle$. This new expression is known as the Gibbs free energy

$$\mathcal{G} = \mathcal{F} - \sum_i \frac{h_i}{T} \frac{\partial \mathcal{F}}{\partial h_i} \quad , \tag{2.23}$$

and we will use it to find some function that relates h_i with $\langle S_i \rangle$. The Gibbs energy can be approximated by an expansion that is known as the Plefka [Plefka, 1982] expansion when $h_i \gg w_{ij}$. We then define a constant term $\lambda \simeq 0$ such that $w_{ij} = \lambda w'_{ij}$ and then

$$E = -\sum_i h_i - \sum_{i<j} S_i S_j w_{ij} = -\sum_i h_i - \lambda \sum_{i<j} S_i S_j w'_{ij} \simeq -\sum_i h_i \quad , \tag{2.24}$$

so we now consider that the Gibbs free energy is function of $\langle S_i \rangle$ and λ, while $\mathcal{F}$ is function of h_i and λ. Then

$$\mathcal{G}\left(\langle S_i \rangle, \lambda\right) = \mathcal{F}\left(h_i, \lambda\right) - \sum_i \frac{h_i}{T} \frac{\partial \mathcal{F}}{\partial h_i} \quad . \tag{2.25}$$

We need to add this λ term because the Plefka expansion states that

$$\mathcal{G}\left(\langle S_i\rangle, \lambda\right)|_{\lambda\simeq 0} \simeq \mathcal{G}\left(\langle S_i\rangle, 0\right) + \lambda \frac{\partial \mathcal{G}\left(\langle S_i\rangle, 0\right)}{\partial \lambda} + \mathcal{O}\left(\lambda^2\right) \quad , \tag{2.26}$$

thus considering that $\mathcal{O}\left(\lambda^2\right) \to 0$ and therefore is negligible -this term however has been approximated in a more precise expansion in Ref. [Kuroki et al., 1999]. We now analyze both terms from the right hand side (rhs) of Eq. 2.26 to reach an expression that can be used to relate h_i with $\langle S_i\rangle$. The process begins by calculating the derivative $\left.\frac{\partial \mathcal{F}}{\partial h_i}\right|_{\lambda=0}$, where

$$E = -\sum_i h_i \quad , \tag{2.27}$$

is assumed. The Boltzmann probability distribution p is thus approximated by using only the bias terms of the units from the neural network

$$p = \frac{\mathrm{e}^{\sum_i \frac{h_i}{T} S_i^\alpha}}{\sum_{S_i} \mathrm{e}^{\sum_i \frac{h_i}{T} S_i}} \quad , \tag{2.28}$$

where the sum at the partition function $\mathcal{Z} = \sum_{S_i} \mathrm{e}^{\sum_i \frac{h_i}{T} S_i}$ is carried out for all the values that these units can take. Now, we proceed

$$\begin{aligned} \left.\frac{\partial \mathcal{F}}{\partial h_i}\right|_{\lambda=0} &= -\frac{1}{\mathcal{Z}} \frac{\partial \mathcal{Z}}{\partial h_i} \\ &= -\frac{1}{\sum_{S_i} \mathrm{e}^{\sum_i \frac{h_i}{T} S_i}} \sum_{S_i} S_i \mathrm{e}^{\sum_i \frac{h_i}{T} S_i} \\ &= -\tanh\left(\frac{h_i}{T}\right) = -\langle S_i\rangle \quad . \end{aligned} \tag{2.29}$$

This expression can be inverted to place $\langle S_i\rangle$ as function of h_i in Eq. 2.23

$$h_i = T \ln\left(\sqrt{\frac{1+\langle S_i\rangle}{1-\langle S_i\rangle}}\right) = T \ln\left(\sqrt{\frac{1+\langle S_i\rangle}{1-\langle S_i\rangle}}\right) \quad , \tag{2.30}$$

so we arrive at

$$\mathcal{G}\left(\langle S_i\rangle, \lambda\right) = \mathcal{F}\left(h_i, \lambda\right) + \sum_i \langle S_i\rangle \ln\left(\sqrt{\frac{1+\langle S_i\rangle}{1-\langle S_i\rangle}}\right) \quad . \tag{2.31}$$

We now replace the Helmholtz free energy for the expression of the partition function for $\lambda \simeq 0$, thus

$$\begin{aligned}
\mathcal{G}\left(\langle S_i\rangle, 0\right) &= -\ln \mathcal{Z}|_{\lambda=0} + \sum_i \langle S_i\rangle \ln\left(\sqrt{\frac{1+\langle S_i\rangle}{1-\langle S_i\rangle}}\right) \\
&= -\ln \sum_{S_i} \mathrm{e}^{\sum_i \ln\left(\sqrt{\frac{1+\langle S_i\rangle}{1-\langle S_i\rangle}}\right) S_i} + \sum_i \langle S_i\rangle \ln\left(\sqrt{\frac{1+\langle S_i\rangle}{1-\langle S_i\rangle}}\right) \\
&= \sum_i \left(\frac{1+\langle S_i\rangle}{2}\ln\frac{1+\langle S_i\rangle}{2} + \frac{1-\langle S_i\rangle}{2}\ln\frac{1-\langle S_i\rangle}{2}\right) \quad . \qquad (2.32)
\end{aligned}$$

We now need to calculate the derivative from the Plefka expansion at Eq. 2.26

$$\begin{aligned}
\frac{\partial \mathcal{G}}{\partial \lambda} &= -\frac{\partial \ln \mathcal{Z}}{\partial \lambda} + \frac{\partial\left(\sum_i \frac{h_i}{T}\langle S_i\rangle\right)}{\partial\lambda} = -\frac{1}{\mathcal{Z}}\frac{\partial\left(\sum_{S_i} \mathrm{e}^{-\sum_i \frac{h_i}{T}S_i + \lambda \sum_{i<j}\frac{w'_{ij}}{T}S_iS_j}\right)}{\partial\lambda} \\
&= -\frac{\sum_{S_i,S_j}\sum_{i<j} w'_{ij}S_iS_j \mathrm{e}^{\sum \frac{h_i}{T}S_i + \lambda \frac{\sum S_i,S_j \sum_{i<j} w'_{ij}S_iS_j}{T}}}{T\mathcal{Z}}\Bigg|_{\lambda=0} \\
&= -\frac{\sum_{S_i,S_j}\sum_{i<j} w'_{ij}S_iS_j \mathrm{e}^{\sum \frac{h_i}{T}S_i}}{T\mathcal{Z}} \\
&= -\sum_{i<j}\frac{w'_{ij}}{T}\langle S_i\rangle\langle S_j\rangle \quad , \qquad (2.33)
\end{aligned}$$

since

$$\frac{\partial\left(\sum_i \frac{h_i}{T}\langle S_i\rangle\right)}{\partial\lambda} = 0 \ , \qquad (2.34)$$

as it does not depend on λ. We now take Eq. 2.26 and undo the normalization of the weights $w_{ij} = \lambda w'_{ij}$ thus arriving at

$$\begin{aligned}
\mathcal{G}\left(\langle S_i\rangle, \lambda\right) &\simeq \sum_i \left(\frac{1+\langle S_i\rangle}{2}\ln\frac{1+\langle S_i\rangle}{2} + \frac{1-\langle S_i\rangle}{2}\ln\frac{1-\langle S_i\rangle}{2}\right) - \lambda\sum_{i<j}\frac{w'_{ij}}{T}\langle S_i\rangle\langle S_j\rangle \\
&= \sum_i \left(\frac{1+\langle S_i\rangle}{2}\ln\frac{1+\langle S_i\rangle}{2} + \frac{1-\langle S_i\rangle}{2}\ln\frac{1-\langle S_i\rangle}{2}\right) - \sum_{i<j}\frac{w_{ij}}{T}\langle S_i\rangle\langle S_j\rangle \quad , \qquad (2.35)
\end{aligned}$$

where $w_{ij} \ll h_i$. Notice now that the differentiation of the Gibbs energy functional will lead to a minimum of the Helmholtz free energy -it is a property of the Legendre transform.

Furthermore, this minimum value of $\mathcal{F}$ is only achieved at thermal equilibrium in the lowest temperature of the cooling schedule, therefore this is the point where we want to compute the mean values of the neurons. Due to the Legendre transform properties, one can state that

$$\frac{\partial \mathcal{G}}{\partial \langle S_i \rangle} = \frac{h_i}{T} \ , \tag{2.36}$$

so

$$\begin{aligned} \frac{\partial \mathcal{G}}{\partial \langle S_i \rangle} &= \frac{1}{2} \ln \left(\frac{1 + \langle S_i \rangle}{2} \right) + \frac{1}{2} - \frac{1}{2} \ln \left(\frac{1 - \langle S_i \rangle}{2} \right) - \frac{1}{2} + \frac{\partial \left(- \sum_{i<j} \frac{w_{ij}}{T} \langle S_i \rangle \langle S_j \rangle \right)}{\partial \langle S_i \rangle} \\ &= \frac{1}{2} \ln \left(\frac{1 + \langle S_i \rangle}{2} \right) - \frac{1}{2} \ln \left(\frac{1 - \langle S_i \rangle}{2} \right) + \frac{\partial \left(- \sum_{i<j} \frac{w_{ij}}{T} \langle S_i \rangle \langle S_j \rangle \right)}{\partial \langle S_i \rangle} \\ &= \operatorname{atanh} \left(\langle S_i \rangle \right) + \frac{\partial \langle H \rangle}{\partial \langle S_i \rangle} = \frac{h_i}{T} \ , \end{aligned} \tag{2.37}$$

and, finally

$$\langle S_i \rangle = \tanh \left(\frac{h_i}{T} - \frac{\partial \left(- \sum_{i<j} \frac{w_{ij}}{T} \langle S_i \rangle \langle S_j \rangle \right)}{\partial \langle S_i \rangle} \right) \ , \tag{2.38}$$

hence we arrive at the Mean Field equations

$$\langle S_i \rangle = \tanh \left(\frac{h_i}{T} + \sum_j \frac{w_{ij}}{T} \langle S_j \rangle \right) \ ,$$

which are a good approximation as far as $h_i \gg w_{ij}$. From this expression we can see that Hopfield neural network equations can be recalled as an Ising model where temperature has reached zero value [Hertz et al., 1991, Baldi and Venkatesh, 1993]. Notice then that

$$S_i = \lim_{T \to 0} \tanh \left(\frac{h_i}{T} + \sum_j \frac{w_{ij}}{T} \langle S_j \rangle \right) = \operatorname{sgn} \left(h_i + \sum_j w_{ij} S_j \right) \ . \tag{2.39}$$

Though the deterministic BM is faster to compute, standard Monte Carlo simulation must be used when the BM is either reproducing or extrapolating probability distributions, since it is unable to provide exact values for coupled correlations or the probability distribution for the output units.

2.3.4 The high order Boltzmann Machine

The high order Boltzmann Machine [Sejnowski, 1987] is an extension to the original model where weights may connect more than two units, even up to N units on an N units neural network. From now on, any weight from both standard BM and HOBM will be represented by denoting the number of units it connects as (n)

$$w_{\sigma}^{(n)} \ , \tag{2.40}$$

while σ stands for the label of the units that the weight links. Bias terms and standard weights now change their notation into

$$h_i = w_i^{(1)} \ , \tag{2.41}$$

$$w_{ij} = w_{ij}^{(2)} \ , \tag{2.42}$$

while a third order weight would be expressed as $w_{ijk}^{(3)}$. This connection, which links units S_i, S_j and S_k is depicted on Fig. 2.5.

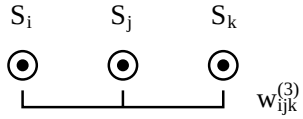

Figure 2.5: A third order weight connecting output units S_i, S_j and S_k.

The energy functional changes according to the newly introduced set of weights; for an N units Boltzmann Machine it reads as

$$E = -\sum_i w_i^{(1)} S_i - \sum_{i<j} w_{ij}^{(2)} S_i S_j - \sum_{i<j<k} w_{ijk}^{(3)} S_i S_j S_k - \ldots - w_{12\ldots N}^{(N)} S_1 S_2 \ldots S_N \ , \tag{2.43}$$

this expression is best written down [Albizuri et al., 1995, Burshtein, 1998] on a compact notation

$$E = -\sum_{n,\sigma} w_{\sigma}^{(n)} \prod_{\rho} S_{\rho} \ . \tag{2.44}$$

Despite the addition of these new, higher order connections, the HOBM has exactly the same dynamics as the standard BM [Sejnowski, 1987]: it still uses the Simulated

Annealing algorithm to simulate thermal equilibrium and to reach the Boltzmann probability distribution. Though the new energy functional is used instead of the standard one, the neural network is still able to learn and extrapolate a probability distribution [Kosmatopoulos and Christodoulou, 1994]. Mean Field equations are also defined on a HOBM basis [Tanaka, 1999], equations do now stand for

$$\langle S_i \rangle = \tanh\left(\frac{w_i^{(1)}}{T} + \sum_j^N \frac{w_{ij}^{(2)}}{T} \langle S_j \rangle + \sum_{j,k}^N \frac{w_{ijk}^{(3)}}{T} \langle S_j \rangle \langle S_k \rangle + \ldots + \frac{w_\sigma^{(N)}}{T} \prod \langle S_\rho \rangle \right) \ , \quad (2.45)$$

however they are still an approximation that works as far as $w_i^{(1)}$ is higher enough than the other values.

2.4 Learning on Boltzmann Machines

In this section, the learning process of the Boltzmann Machine is described. The learning equations that are used on a standard BM are analyzed in the first part of this section; this is used as an introduction to a next part where the learning process is briefly described. We then explain how this algorithm is applied to a high order Boltzmann Machine; the section is concluded by introducing an optional learning process based on the Mean Field approach, where the quantities needed to compute the weight updates are approximated.

2.4.1 Learning expression for a standard BM

The Boltzmann Machine has the feature of being able to learn and extrapolate probability distributions; this ability forces the usage of a metric that is able to relate both the pattern that we want the neural network to learn and its own (Boltzmann) distribution. The measure that describes this distance best is known as the Kullback-Leibler [Kullback, 1959] distance

$$G = \sum_\gamma p(\gamma) \sum_\alpha r(\alpha \mid \gamma) \ln \frac{r(\alpha \mid \gamma)}{p(\alpha \mid \gamma)} \ . \quad (2.46)$$

When G is applied to a BM, $r(\alpha \mid \gamma)$ is the probability distribution that we would

like the neural network to learn and $p(\alpha \mid \gamma)$ is the Boltzmann probability distribution. In this expression, $p(\alpha \mid \gamma)$ reads as the Boltzmann probability of finding an output state α when a state γ has been set in the input units and $r(\alpha \mid \gamma)$ as the desired probability distribution to be learned from the training set [Hertz et al., 1991]. It can be shown that $G > 0$ for any $p(\alpha \mid \gamma) \neq r(\alpha \mid \gamma)$ and that it reaches the global minimum $G = 0$ when $p(\alpha \mid \gamma) = r(\alpha \mid \gamma)$. In absence of hidden units, this function is a convex function [Albizuri et al., 1996]. If hidden units are added, the shape of G is uncertain, though the global minimum still happens at $p(\alpha \mid \gamma) = r(\alpha \mid \gamma)$. Gradient descent is commonly used to find the update rule for weights and biases [Ackley et al., 1985]

$$\Delta w_i^{(1)} = -\eta \frac{\partial G}{\partial w_i^{(1)}} \ , \tag{2.47}$$

$$\Delta w_{ij}^{(2)} = -\eta \frac{\partial G}{\partial w_{ij}^{(2)}} \ , \tag{2.48}$$

where η is an arbitrary constant. The Kullback distance can be best differentiated with the aid of the properties of the logarithm

$$G = \sum_{\gamma} p(\gamma) \sum_{\alpha} r(\alpha \mid \gamma) \ln \frac{r(\alpha \mid \gamma)}{p(\alpha \mid \gamma)} \tag{2.49}$$

$$= \sum_{\gamma} p(\gamma) \sum_{\alpha} r(\alpha \mid \gamma) \left(\ln r(\alpha \mid \gamma) - \ln p(\alpha \mid \gamma) \right) \ . \tag{2.50}$$

The inclusion of hidden units to the neural network is denoted by adding a term β to the probability distribution $p(\alpha \mid \gamma)$ for $p(\alpha, \beta \mid \gamma)$. Notice however that the probability distribution that the hidden units may reach is often worthless: they are used to increase the learning capacity of the BM, and we do not care about the values they take as far as the output units reproduce a given probability distribution. We calculate $p(\alpha \mid \gamma)$ as a marginal probability sum

$$p(\alpha \mid \gamma) = \sum_{\beta} p(\alpha, \beta \mid \gamma) = \sum_{\beta} \frac{e^{-\frac{1}{T} E_{\alpha, \beta \mid \gamma}}}{\mathcal{Z}_{\gamma}} \ , \tag{2.51}$$

the inclusion of these hidden units is applied to the partition function under a ν term

$$\mathcal{Z}_{\gamma} = \sum_{\mu, \nu} e^{-\frac{1}{T} E_{\mu, \nu \mid \gamma}} \ , \tag{2.52}$$

since the partition function sums over all feasible states; in this case input units are clamped at a state γ. We will work both bias and weight updating expressions by setting a generic order (n) on a standard weight $w_{\sigma}^{(n)}$; hence

$$\begin{aligned} \frac{\partial G}{\partial w_{\sigma}^{(n)}} &= -\sum_{\gamma} p(\gamma) \sum_{\alpha} r(\alpha \mid \gamma) \frac{\partial \ln p(\alpha \mid \gamma)}{\partial w_{\sigma}^{(n)}} \\ &= -\sum_{\gamma} p(\gamma) \sum_{\alpha} r(\alpha \mid \gamma) \frac{1}{p(\alpha \mid \gamma)} \frac{\partial p(\alpha \mid \gamma)}{\partial w_{\sigma}^{(n)}} \ , \end{aligned} \tag{2.53}$$

since $r(\alpha \mid \gamma)$ does not depend on $w_{\sigma}^{(n)}$ and thereafter $\frac{\partial r(\alpha|\gamma)}{\partial w_{\sigma}^{(n)}} = 0$. We now differentiate the energy functional

$$\frac{\partial E_{\alpha,\beta|\gamma}}{\partial w_{\sigma}^{(n)}} = -\prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} \ , \tag{2.54}$$

where $S_{\rho}^{\alpha,\beta|\gamma}$ stands for either a hidden or output unit whose state depends on the input units state γ. In this sense, notice that the output units state α from $S_{\rho}^{\alpha,\beta|\gamma}$ is fixed. We now calculate $\frac{\partial p(\alpha|\gamma)}{\partial w_{\sigma}^{(n)}}$, thus

$$\frac{\partial p(\alpha \mid \gamma)}{\partial w_{\sigma}^{(n)}} = \sum_{\beta} \left(\frac{\frac{1}{T} \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}} \mathcal{Z}_{\gamma} - e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}} \frac{\partial \mathcal{Z}_{\gamma}}{\partial w_{\sigma}^{(n)}}}{\mathcal{Z}_{\gamma}^{2}} \right) , \tag{2.55}$$

where differentiation of the partition function is required

$$\frac{\partial \mathcal{Z}_{\gamma}}{\partial w_{\sigma}^{(n)}} = \sum_{\mu,\nu} e^{-\frac{1}{T} E_{\mu,\nu|\gamma}} \frac{1}{T} \prod_{\rho \in \sigma} S_{\rho}^{\gamma} = \frac{1}{T} Z_{\gamma} \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle , \tag{2.56}$$

$E_{\mu,\nu|\gamma}$ being the energy functional value obtained when input units get clamped at a state

γ. Finally

$$\begin{aligned}
\frac{\partial G}{\partial w_{\sigma}^{(n)}} &= -\sum_{\gamma} p(\gamma) \sum_{\alpha} \frac{r(\alpha \mid \gamma)}{p(\alpha \mid \gamma)} \sum_{\beta} \left(\frac{\frac{1}{T} \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}} \mathcal{Z}_{\gamma} - e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}} \frac{\partial \mathcal{Z}_{\gamma}}{\partial w_{\sigma}^{(n)}}}{\mathcal{Z}_{\gamma}^{2}} \right) \\
&= -\frac{1}{T} \sum_{\gamma} p(\gamma) \sum_{\alpha} \frac{r(\alpha \mid \gamma)}{p(\alpha \mid \gamma)} \sum_{\beta} \left(\frac{\prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}}}{\mathcal{Z}_{\gamma}} - \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle \frac{e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}}}{\mathcal{Z}_{\gamma}} \right) \\
&= -\frac{1}{T} \sum_{\gamma} p(\gamma) \sum_{\alpha} \sum_{\beta} \left(\frac{\prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}}}{\mathcal{Z}_{\gamma}} - \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle \frac{e^{-\frac{1}{T} E_{\alpha|\gamma}}}{\mathcal{Z}_{\gamma}} \right) \\
&= -\frac{1}{T} \sum_{\gamma} p(\gamma) \left(\sum_{\alpha,\beta} \frac{\prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}}}{\mathcal{Z}_{\gamma}} - \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle \sum_{\alpha,\beta} \frac{e^{-\frac{1}{T} E_{\alpha,\beta|\gamma}}}{\mathcal{Z}_{\gamma}} \right) \\
&= -\frac{1}{T} \sum_{\gamma} p(\gamma) \left(\left\langle \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} \right\rangle - \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle \right) . \qquad (2.57)
\end{aligned}$$

These expectation values are computed for each different γ vector, therefore effectively calculating

$$\sum_{\gamma} p(\gamma) \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} \right\rangle = \overline{\left\langle \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} \right\rangle} , \qquad (2.58)$$

$$\sum_{\gamma} p(\gamma) \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle = \overline{\left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle} , \qquad (2.59)$$

now we arrive at

$$\begin{aligned}
\frac{\partial G}{\partial w_{\sigma}^{(n)}} &= -\frac{1}{T} \sum_{\gamma} p(\gamma) \left(\left\langle \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} \right\rangle - \left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle \right) \\
&= -\frac{1}{T} \left(\overline{\left\langle \prod_{\rho \in \sigma} S_{\rho}^{\alpha,\beta|\gamma} \right\rangle} - \overline{\left\langle \prod_{\rho \in \sigma} S_{\rho}^{\gamma} \right\rangle} \right) . \qquad (2.60)
\end{aligned}$$

This expression is commonly written down as

$$\frac{\partial G}{\partial w_{\sigma}^{(n)}} = -\frac{\eta}{T} \left(\left\langle \prod_{\rho \in \sigma} S_{\rho} \right\rangle^{*} - \left\langle \prod_{\rho \in \sigma} S_{\rho} \right\rangle \right) , \qquad (2.61)$$

where the $*$ term indicates that the correlations are computed for a fixed input and output pattern to both input and output units; notice then that only hidden units are

allowed to change. The units that are not allowed to change are referred to as being clamped, and the estimation of these correlations is known as *clamped phase*. On the other hand, the quantity with no $*$ is computed by setting a given input pattern to the input layer but allowing the remaining neurons from the neural network to change their state freely; according to the dynamics of the system. The process where these correlations are estimated is known as *free phase*. In essence, we can say that this expression compares the probability distribution that we want the neural network to learn (this is, the clamped phase) against its own Boltzmann distribution (the free phase). The final update rule for both bias and weights becomes

$$\Delta w_{\sigma}^{(n)} = \frac{\eta}{T}\left(\left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle^{*} - \left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle\right) \quad . \tag{2.62}$$

Since we are working with a standard BM, weights are restricted to second order -$n = 2$- and bias terms -$n = 1$-, and then

$$\Delta w_{i}^{(1)} = \frac{\eta}{T}\left(\langle S_i\rangle^{*} - \langle S_i\rangle\right) \quad , \tag{2.63}$$

$$\Delta w_{ij}^{(2)} = \frac{\eta}{T}\left(\langle S_i S_j\rangle^{*} - \langle S_i S_j\rangle\right) \quad . \tag{2.64}$$

2.4.2 Learning algorithm for a BM

Now that we have seen how the learning expressions of a Boltzmann Machine are deduced, we describe how these quantities are computed on a typical BM. Let us have a standard BM learning pattern $\{\Gamma, \mathcal{A}\}$ consisting of V vectors, which describes the input and output states

$$\{\Gamma, \mathcal{A}\} = \left(\{\gamma_1, \alpha_1\}\{\gamma_2, \alpha_2\}\ldots\{\gamma_V, \alpha_V\}\right) \quad , \tag{2.65}$$

for the input and output units, and the associated set of probabilities $\vec{p} = (p_1, p_2, \ldots, p_V)$ in which they happen. We will also consider that the neural network has n_i input units, n_h hidden units and n_o output neurons. The learning algorithm is carried out by following these steps [Freeman and Skapura, 1993]:

1. Get the first learning vector $\{\gamma_1, \alpha_1\}$ from the learning pattern. Fix the input units to a state γ_1 and the output neurons to a state α_1, this will begin the clamped phase.

2. Carry out the simulation process to the neural network until it reaches thermal equilibrium at the last temperature T_K from the cooling schedule, thus selecting *only* hidden units to carry out the SA process. Consider then that a single iteration of the cooling schedule will run n_h times, since these are the units that can change their state.

3. Carry out the probability estimation process to the BM for m iterations, as described in section 2.3.2. Calculate $\langle S_i S_j \rangle^*$ and $\langle S_i \rangle^*$ as

$$\langle S_i S_j \rangle^* = \sum_v \sum_{S_i, S_j} p_v S_i S_j \ , \tag{2.66}$$

$$\langle S_i \rangle^* = \sum_v \sum_{S_i} p_v S_i \ . \tag{2.67}$$

4. Get again the first learning vector $\{\gamma_1, \alpha_1\}$ from the learning pattern. Fix the input units to a state γ_1, this will begin the free phase.

5. Carry out the simulation process to the neural network until it reaches thermal equilibrium at the last temperature T_K from the cooling schedule, just as seen on the simulation process. Notice though that an iteration now runs $n_h + n_o$ times the cooling schedule.

6. Carry out the probability estimation process to the BM for m iterations. Calculate $\langle S_i S_j \rangle$ and $\langle S_i \rangle$.

7. Repeat this process from step 1 and compute the mean values and expectation values for the whole learning pattern.

8. Update the weights according to the learning expression from Eqs. 2.63 and 2.64.

9. Repeat this whole process until all $\Delta w_{ij}^{(2)} < \varepsilon$ and all $\Delta w_{i}^{(1)} < \varepsilon$, where ε is an arbitrarily small value that is selected for convergence means.

Notice now that this algorithm solves a standard MC integration [Press et al., 1993] algorithm when computing the correlations and expectation values. Thus, the relative error Err_R is about

$$Err_R \propto \frac{1}{\sqrt{m}} \ , \tag{2.68}$$

being m the number of samples. This is also the number of iterations that are carried out at the last vale of the cooling schedule, once the system has reached thermal equilibrium. The absolute error Err_A for a given correlation on a BM is calculated as

$$Err_A = 2\, Err_R \ , \tag{2.69}$$

since 2 is the spanning range of both correlations and expectation values. Therefore, the absolute error can be expressed as function of m

$$Err_A \propto \frac{2}{\sqrt{m}} \ . \tag{2.70}$$

This expression means that the Boltzmann Machine has to be simulated about $\frac{\sqrt{m}}{2}$ times in order to perform a weight update once with a precision proportional to Err_A. Learning process is typically carried out until $\Delta w_{\sigma}^{(n)} < \varepsilon$, where ε is an arbitrarily small value. Notice that Err_A and ε are closely related: if it happens that $Err_A > \varepsilon$ the learning algorithm will not be able to finish on a reliable solution, because we can not ensure that the same correlations have an implicit error bigger than this quantity. This has been so far the issue with Boltzmann Machines, as the associated computational cost for the learning algorithm prevents widespread usage of the neural network.

2.4.3 Learning on a HOBM

Learning on a high order Boltzmann Machine is also carried out by Kullback-Leibler distance optimization between the own probability distribution of the neural network and

the probability distribution that we want it to know. This operation results in

$$\Delta w_{\sigma}^{(n)} = \frac{\eta}{T}\left(\left\langle \prod_{\rho\in\sigma} S_{\rho}\right\rangle^{*} - \left\langle \prod_{\rho\in\sigma} S_{\rho}\right\rangle\right) \ ,$$

for a third order BM [Sejnowski, 1987] it would read as

$$\Delta w_{ijk}^{(3)} = \frac{\eta}{T}\left(\langle S_i S_j S_k\rangle^{*} - \langle S_i S_j S_k\rangle\right) \ . \tag{2.71}$$

However, higher order correlations become harder to estimate by Monte Carlo means [Graña et al., 1997] and though the inclusion of these terms provides an enhanced learning capability to the neural network [Albizuri et al., 1997], the algorithm does also greatly increase in complexity [Tanaka, 1999]. Now it becomes a compromise between many high order weights or many hidden units to reach similar capabilities on different neural networks. A valid solution is so far shown in Ref. [Albizuri et al., 1996]: the Kullback distance on an n-th order Boltzmann Machine with no hidden units is always a convex function, and therefore the algorithm will always reach a final solution. In order to decrease the complexity needed to compute the higher order correlations, the proposed Boltzmann Machine uses high order weights that connect any quantity of input units to either one or two output neurons. It is straightforward seen that, for a given set of n free units $[S_{i_1}, S_{i_2}, \ldots, S_{i_n}]$ and a given set of m clamped units $[S^*_{j_1}, S^*_{j_2}, \ldots, S^*_{j_m}]$, correlation for a weight with order $m+n$ can be computed as

$$\left\langle S_{i_1} S_{i_2} \cdots S_{i_n} S^*_{j_1} S^*_{j_2} \cdots S^*_{j_m}\right\rangle = \left\langle S_{i_1} S_{i_2} \cdots S_{i_n}\right\rangle S^*_{j_1} S^*_{j_2} \cdots S^*_{j_m} \ , \tag{2.72}$$

hence, any high order connection involving two free units will be computed as

$$\left\langle S_{i_1} S_{i_2} S^*_{j_1} S^*_{j_2} \cdots S^*_{j_m}\right\rangle = \left\langle S_{i_1} S_{i_2}\right\rangle S^*_{j_1} S^*_{j_2} \cdots S^*_{j_m} \ , \tag{2.73}$$

no matter the order of the weight.

The conclusion is that for a fully connected HOBM with no hidden units whose high order terms connect no more than two output units, we will have higher learning capabilities [Graña et al., 1997, Tanaka, 1999] than on a standard BM. Furthermore, the learning algorithm will always provide the best solution that this topology is able to learn from a

given dataset, being it convex, it is guaranteed that a global optimum of the Kullback-Leibler distance is found. However, the high number of weights that are used on this structure increases the complexity of this implementation.

2.4.4 The Mean Field learning solution

The Mean Field theory application to the Boltzmann Machine learning problem was first proposed as the *naive* mean field learning process in Ref. [Peterson and Anderson, 1987], with the approximation of the system coupled correlations by the product of their expectation values

$$\langle S_i S_j \rangle \simeq \langle S_i \rangle \langle S_j \rangle \quad , \tag{2.74}$$

where $\langle S_i \rangle$ is found by numerically solving the Mean Field equations, as seen in section 2.3.3

$$\langle S_i \rangle = \tanh\left(\frac{w_i^{(1)}}{T} + \sum \frac{w_{ij}^{(2)}}{T} \langle S_j \rangle \right) \quad .$$

The main point on using the mean field for the learning process is that correlations are approximated analytically and faster than using Monte Carlo methods. Mean field learning on a Boltzmann Machine is often referred to as learning on deterministic Boltzmann Machines [Hagiwara, 1992], and it is a standard solution for hardware implementations [Schneider and Card, 1993]. On the other hand, a more precise learning rule than the naive mean field method was proposed by Ref. [Kappen and Rodriguez, 1998]. This method is based in the Linear Response Theory [Parisi, 1988]

$$\langle S_i S_j \rangle = \langle S_i \rangle \langle S_i \rangle + A_{ij} \quad , \tag{2.75}$$

where A_{ij} stands for the ij position of the following $\mathcal{A}$ matrix

$$\mathcal{A} = \left(\frac{\delta_{ij}}{1 - \langle S_i \rangle^2} - \frac{1}{T} \mathcal{W} \right)^{-1} \quad , \tag{2.76}$$

$\mathcal{W}$ being the matrix that represents the weights connecting the units from the neural network. Notice that this matrix is symmetric with zero diagonal, since the weights of a

BM are bidirectional and units are not connected to themselves. On the other hand, δ_{ij} is the Kronecker delta

$$\delta_{ij} = \begin{cases} 1, & \text{if} \quad i = j \\ 0, & \text{if} \quad i \neq j \end{cases} .$$

The mean field learning method has been widely used as an *easy* BM implementation [Kappen and Wiegerinck, 2001], since the learning algorithm is performed faster than the standard MC based algorithm. As a consequence, it has impulsed research about third [Tanaka, 1999] and fourth [Leisink and Kappen, 2000] order correlations estimation for a MF, HOBM model. However, the relationship within weights and updates is not so direct as a matrix inverse. Furthermore, the model has not yet been able to overcome some serious drawbacks: if the neural network is reproducing a probability distribution, it needs an annealing process (which becomes harder due to the higher order weights), and the bias terms must still represent a significant value on the energy functional or the solution will loose accuracy.

Chapter 3

The process of Decimation

3.1 Introduction

Decimation is a technique that is used in statistical physics to reduce the size of the current system to another similar one, yet retaining most of its features [Cardy, 1996]. It can be used to focus on a fragment of the given material, hence reducing the complexity of the associated calculus. In terms of a Boltzmann Machine [Saul and Jordan, 1994], it becomes a procedure which allows us to make a transformation from a complex neural network to another smaller without loss of its properties. This means that the new network is an equivalent BM without one of its original units while the remaining ones yet retain the same behavior. This process is shown in Fig. 3.1, where a central unit S_d connected to units S_i, S_j and S_k is decimated, thus creating a new set of connections linking these neurons.

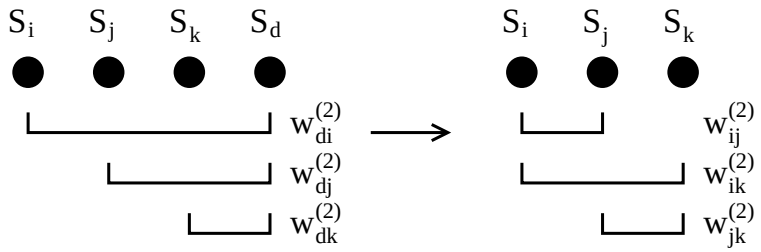

Figure 3.1: Applied example of decimation.

In practical terms, decimation is applied to a Boltzmann Machine when one is interested on analytically finding the quantities that are needed at the learning stage

$$\Delta w_{\sigma}^{(n)} = \frac{\eta}{T}\left(\left\langle \prod_{\rho=1}^{n} S_{\rho} \right\rangle^{*} - \left\langle \prod_{\rho=1}^{n} S_{\rho} \right\rangle\right) \quad , \tag{3.1}$$

which should otherwise be estimated by using Monte Carlo means. The decimation process as presented in Ref. [Saul and Jordan, 1994] was conceived to be applied iteratively for each pair of connected units. The topology of a BM where decimation could be applied was therefore referred to as *decimatable* [Rüger et al., 1996], an example of this structure is depicted in Fig. 3.2. This neural network would be decimated in order to update weight $w_{ij}^{(2)}$, thus computing the correlation value between S_i and S_j. The process would then be repeated for each pair of connected units in order to compute all the required correlations and expectation values.

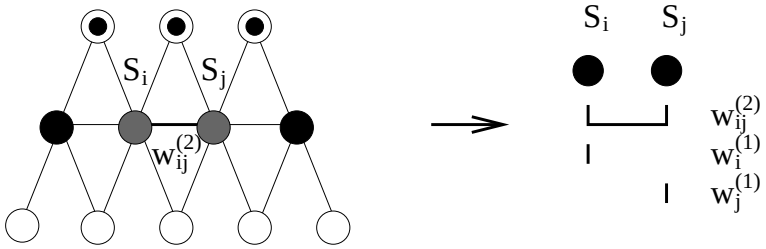

Figure 3.2: Decimatable structure and decimated model.

This chapter is organized as follows: section 3.2 presents the decimation process as proposed in Ref. [Saul and Jordan, 1994] and as further extended in Ref. [Rüger, 1997]. Section 3.3 is devoted to explaining how these methods are used to compute exact expectation values and correlations in the Boltzmann Machine. The main drawbacks of the standard decimation process, the high order Decimation (HOD) method [Farguell et al., 2008] and the way that it overcomes the problems that are found when applying decimation and a full discussion of its equations are analyzed in section 3.4. The chapter proceeds then with an extension to the high order Decimation that has been named as the Multiple Decimation process, thus allowing to algorithmically implement the HOD, and is concluded with some results of the HOD method applied to a set of learning problems.

3.2 Decimation applied to the BM

This section describes the application of the decimation process to a neural network such as the Boltzmann Machine. We start with an introduction to the equations that explain how standard decimation works, and why is it made possible on some given topologies of BM. The foregoing parts of this section are used to describe from the most basic to the most complex decimation procedures.

3.2.1 Main concepts from decimation

The basic idea behind the decimation procedure is to suppress a given unit S_d connected to its neighboring set of units $\mathcal{S}$ by a set of weights $\{w_d^{(1)}, w_{di}^{(2)}\}$, and substitute it with a new equivalent set of connections. This process is carried out at the last temperature from the cooling schedule, which is the equilibrium temperature of the BM; therefore this value is constant. The dependency on temperature can then be assimilated by the weights if they are normalized

$$J_{ij}^{(2)} = \frac{w_{ij}^{(2)}}{T} \ , \tag{3.2}$$

$$J_{i}^{(1)} = \frac{w_{i}^{(1)}}{T} \ , \tag{3.3}$$

T being the last temperature from the cooling schedule , $w_{ij}^{(2)}$ the weight connecting units S_i and S_j and $w_i^{(1)}$ the bias term from unit S_i. The Boltzmann probability distribution, the partition function and the energy functional now read as

$$p\left(\alpha\right) = \frac{\mathrm{e}^{-\mathcal{E}_\alpha}}{\mathcal{Z}} \ , \tag{3.4}$$

$$\mathcal{Z} = \sum_{\mu} \mathrm{e}^{-\mathcal{E}_\mu} \ , \tag{3.5}$$

$$\mathcal{E} = -\frac{1}{2} \sum J_{ij}^{(2)} S_i S_j - \sum J_i^{(1)} S_i \ . \tag{3.6}$$

The master equation of decimation stands for [Farguell et al., 2007]

$$\ln\left(\frac{1}{2}\sum_{S_d=-1}^{+1} \mathrm{e}^{\sum J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_\rho}\right) = G^{(0)} + \sum_{n=1,\ \sigma}^{2} G_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho \ , \tag{3.7}$$

where $J_{d\sigma}^{(n)}$ are the weights from the original neural network that connect any S_ρ unit with the one to decimate S_d and $G_\sigma^{(n)}$ are the weights of the resulting BM.

We now discuss this equation: we begin from a given neural network with a set of units $\{\mathcal{S}, S_d\}$; these can take a certain state α_d with an energy value $\mathcal{E}_{\alpha_d}$. Once the decimation process is carried out, the resulting neural network still keeps the set of units $\{\mathcal{S}\}$, but unit S_d is decimated. As a consequence, the resulting model has no weights linking unit S_d to any other neuron. Let α_d be then an energy state on a BM with associated energy value $\mathcal{E}_{\alpha_d}(\mathcal{S}, S_d)$. This state is due to a set of neurons $\{\mathcal{S}, S_d\}$ which have taken a certain combination of values where S_d is undefined

$$\mathcal{E}_{\alpha_d}(\mathcal{S}, S_d) = -\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho - \sum J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_\rho \ , \tag{3.8}$$

this set of neurons $\mathcal{S}$ is connected through a set of temperature normalized weights $\{J_d^{(1)}, J_{di}^{(2)}\}$ to the unit S_d, which is going to be decimated. The value of the energy depends on S_d, and thereafter we could reach $\mathcal{E}_{\alpha_d}(\mathcal{S}, +1)$ for $S_d = +1$ and $\mathcal{E}_{\alpha_d}(\mathcal{S}, -1)$ when $S_d = -1$. We now calculate the sum of the conditional probability distribution for all the possible values that unit S_d can take, which are -1 and $+1$. Thus

$$p(\alpha) = p(\alpha_d)|_{S_d=1} + p(\alpha_d)|_{S_d=-1} \ , \tag{3.9}$$

where the state α is a given combination of the set of units $\mathcal{S}$ and has an associated energy value $\mathcal{E}_\alpha(\mathcal{S})$, hence

$$p(\alpha) = \frac{\mathrm{e}^{-\mathcal{E}_\alpha(\mathcal{S})}}{\mathcal{Z}} \ , \tag{3.10}$$

notice then that $\mathcal{E}_\alpha(\mathcal{S})$ depends on a different set of weights $\left\{\tilde{J}_i^{(1)}, \tilde{J}_{ij}^{(2)}\right\}$, for $i, j \neq d$, because the decimation process creates a new set of connections. Furthermore, $\mathcal{E}_\alpha(\mathcal{S})$ also depends on the units from the set $\mathcal{S}$

$$\mathcal{E}_\alpha(\mathcal{S}) = -\sum \tilde{J}_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho \ , \ \ d \notin \sigma \ . \tag{3.11}$$

We now introduce $\Delta\mathcal{E}_d(\mathcal{S}, S_d)$ as the following quantity

$$\Delta\mathcal{E}_d(\mathcal{S}, S_d) = -\sum J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_\rho \ , \ \ S_\rho \in \mathcal{S} \ , \tag{3.12}$$

which is straightforward used in combination with Eqs. 3.8 and 3.9, thus leading to

$$\begin{aligned} p(\alpha) &= p(\alpha_d)|_{S_d=1} + p(\alpha_d)|_{S_d=-1} \\ &= \frac{\mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho - \Delta\mathcal{E}_d(\mathcal{S},+1)}}{\mathcal{Z}} + \frac{\mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho - \Delta\mathcal{E}_d(\mathcal{S},-1)}}{\mathcal{Z}} \\ &= \mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho} \left(\frac{\mathrm{e}^{-\Delta\mathcal{E}_d(\mathcal{S},+1)}}{\mathcal{Z}} + \frac{\mathrm{e}^{-\Delta\mathcal{E}_d(\mathcal{S},-1)}}{\mathcal{Z}} \right) \ . \end{aligned} \tag{3.13}$$

We now take from Ref. [Saul and Jordan, 1994] the original equation of decimation

$$\sum_{S_d=-1}^{+1} \mathrm{e}^{-\Delta\mathcal{E}_d(\mathcal{S},S_d)} = \sqrt{C}\mathrm{e}^{\sum G_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho} \ , \tag{3.14}$$

where $G_\sigma^{(n)}$ are the unknown weights directly resulting from the decimation operation. If we apply this expression to the one in Eq. 3.13 we arrive at

$$p(\alpha) = \mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho} \frac{\sqrt{C}\mathrm{e}^{\sum G_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho}}{\mathcal{Z}} \ . \tag{3.15}$$

Now it becomes necessary to work with the partition function in order to reach the following expression

$$\mathcal{Z} = \sqrt{C}\mathcal{Z}' \ , \tag{3.16}$$

thus assuming that

$$\mathcal{Z} = \sum_{\gamma=1}^{\Gamma} \mathrm{e}^{-\mathcal{E}_{\gamma_d}(\mathcal{S},S_d)} \ , \tag{3.17}$$

where the sum is carried out for all the possible states that $\{\mathcal{S}, S_d\}$ can reach. This expression also reads as

$$\mathcal{Z} = \sum_{\gamma=1}^{\Gamma} \mathrm{e}^{-\mathcal{E}_{\gamma}(\mathcal{S},S_d)} = \sum_{\mathcal{S},S_d} \mathrm{e}^{-\mathcal{E}_{\gamma_d}(\mathcal{S},S_d)} \ , \tag{3.18}$$

which leads us to

$$\begin{aligned} \mathcal{Z} &= \sum_{\mathcal{S},S_d} \mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho + \sum J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_\rho} \\ &= \sum_{\mathcal{S}} \mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho} \sum_{S_d} \mathrm{e}^{\sum J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_\rho} \\ &= \sum_{\mathcal{S}} \mathrm{e}^{\sum J_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho} \sqrt{C}\mathrm{e}^{\sum G_\sigma^{(n)} \prod_{\rho\in\sigma} S_\rho} = \sqrt{C}\mathcal{Z}' \ . \end{aligned} \tag{3.19}$$

We now take Eq. 3.15 and combine it with the previous expression

$$\begin{aligned} p(\alpha) &= \mathrm{e}^{\sum J_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}} \frac{\sqrt{C}\mathrm{e}^{\sum G_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}}}{\sqrt{C}\mathcal{Z}'} \\ &= \frac{\mathrm{e}^{\sum J_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho} + \sum G_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}}}{\mathcal{Z}'} \ , \end{aligned} \tag{3.20}$$

and recall the definition of $\mathcal{E}_{\alpha}(\mathcal{S})$ from Eq. 3.11

$$\mathcal{E}_{\alpha}(\mathcal{S}) = -\sum \tilde{J}_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho} \ , \ \ d \notin \sigma \ ,$$

thus arriving to the following equivalence

$$p(\alpha) = \frac{\mathrm{e}^{\sum J_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho} + \sum G_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}}}{\mathcal{Z}'} = \frac{\mathrm{e}^{\sum \tilde{J}_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}}}{\mathcal{Z}'} \ , \tag{3.21}$$

where

$$\mathcal{Z}' = \sum_{\mathcal{S}} \mathrm{e}^{\sum J_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}} \mathrm{e}^{\sum G_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho}} = \sum_{\mathcal{S}} \mathrm{e}^{\sum \left(J_{\sigma}^{(n)} + G_{\sigma}^{(n)} \right) \prod_{\rho\in\sigma} S_{\rho}} \ . \tag{3.22}$$

Therefore, a new set of connections

$$\tilde{J}_{\sigma}^{(n)} = J_{\sigma}^{(n)} + G_{\sigma}^{(n)} \ , \tag{3.23}$$

which link the set of units $\{\mathcal{S}\}$ is left in place of the decimated unit S_d. We have shown that the equality from Eq. 3.7 is used in order to decimate any structure, and that this expression is applied to a marginal probability sum of the unit that is currently being decimated. However, this equation is better written down if logarithm is applied at both sides [Farguell et al., 2007], and $\sqrt{C}/2$ is placed into the exponential term with a new notation

$$\ln\left(\frac{1}{2} \sum_{S_d=-1}^{+1} \mathrm{e}^{\sum J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_{\rho}} \right) = G^{(0)} + \sum_{n=1,\ \sigma}^{2} G_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_{\rho} \ .$$

3.2.2 Parallel association

Parallel association is a weight addition that is carried out once decimation has been used over a given structure and a new set of weights has been generated. Hence, it does

not eliminate any unit, but it is used to combine the remaining weights of the neural network with the new set of connections. It is explained by working out its temperature normalized energy expression

$$\tilde{J}_{ij}^{(2)} = J_{ij}^{(2)} + G_{ij}^{(2)} \quad , \tag{3.24}$$

where $J_{ij}^{(2)}$ and $G_{ij}^{(2)}$ are the weights connecting units S_i and S_j and $\tilde{J}_{ij}^{(2)}$ the resulting connection. We can graphically see how parallel association works in Fig. 3.3.

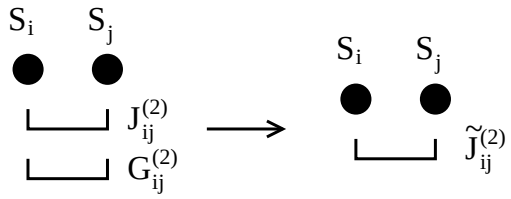

Figure 3.3: Parallel association.

We now analyze how Eq. 3.24 is generated. Let E be the energy functional of the structure depicted in Fig. 3.3

$$E = J_{ij}^{(2)} S_i S_j + G_{ij}^{(2)} S_i S_j = S_i S_j \left(J_{ij}^{(2)} + G_{ij}^{(2)} \right) \quad , \tag{3.25}$$

so the new weight $\tilde{J}_{ij}^{(2)}$ is the addition of the original terms.

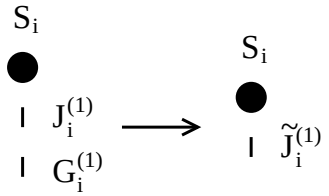

Figure 3.4: Parallel bias simplification.

An interesting fact of the parallel association is that one can add two bias terms by the same way or any set of clamped input units [Saul and Jordan, 1994], as depicted in Fig. 3.4. Notice then that the clamped units can be reduced to a set of parallel associated biases [DeGloria et al., 1993]: let S_{i_1} and S_{i_2} be two input units which are always clamped,

either when learning or on simulation process. These neurons are connected to a given biased unit S_j by weights $J^{(2)}_{i_1 j}$, $J^{(2)}_{i_2 j}$. We refer the bias term as $J^{(1)}_j$ and, since these units are clamped when the learning process is carried out, one can proceed as follows

$$\tilde{J}^{(1)}_j = S_{i_1} J^{(2)}_{i_1 j} + S_{i_2} J^{(2)}_{i_2 j} + J^{(1)}_j \quad , \tag{3.26}$$

for each vector of a given learning pattern set.

3.2.3 Serial association

Serial association was originally proposed in Ref. [Saul and Jordan, 1994] as the most basic decimation procedure used to suppress a unit. This association can only be carried out when there is a single, unbiased unit that is linked to another two neurons, the process is depicted in Fig. 3.5.

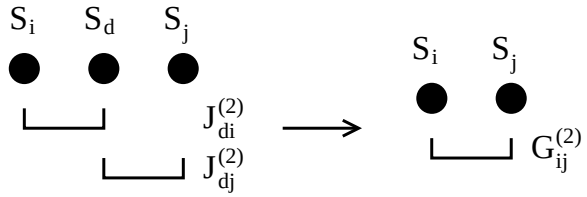

Figure 3.5: Serial association.

Therefore, the structure of the neural network that is being used must be sparsely connected, because we will repeat this process to isolate each pair of connected units. We can see an example of a neural network where this process is applied in Fig. 3.6, thus rendering two units to compute their correlation analytically.

Let S_d be a unit connected to units S_i and S_j by temperature normalized weights $J^{(2)}_{di}$ and $J^{(2)}_{dj}$, respectively; the equation to perform standard serial association over S_d is

$$G^{(2)}_{ij} = \frac{1}{2} \ln \left(\frac{\cosh\left(J^{(2)}_{di} + J^{(2)}_{dj} \right)}{\cosh\left(J^{(2)}_{di} - J^{(2)}_{dj} \right)} \right) \quad , \tag{3.27}$$

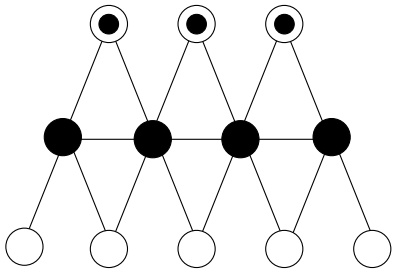

Figure 3.6: Typical structure where serial decimation is applied to find the correlations of the units.

where $G_{ij}^{(2)}$ is the resulting weight. Equation 3.27 can be proven by recalling Eq. 3.7, hence

$$\begin{aligned} \ln\left(\frac{1}{2}\sum_{S_d=-1}^{+1} \mathrm{e}^{J_{di}^{(2)}S_iS_d+J_{dj}^{(2)}S_jS_d}\right) &= G^{(0)}+G_{ij}^{(2)}S_iS_j \ , \\ \ln\cosh\left(J_{di}^{(2)}S_i+J_{dj}^{(2)}S_j\right) &= G^{(0)}+G_{ij}^{(2)}S_iS_j \ , \end{aligned} \tag{3.28}$$

where $G^{(0)}$ and $G_{ij}^{(2)}$are the weights that result from the decimation process.

S_i	S_j	$\ln\cosh\left(J_{di}^{(2)}S_i+J_{dj}^{(2)}S_j\right)=G^{(0)}+G_{ij}^{(2)}S_iS_j$
-1	-1	$\ln\cosh\left(J_{di}^{(2)}+J_{dj}^{(2)}\right)=G^{(0)}+G_{ij}^{(2)}$
-1	1	$\ln\cosh\left(J_{di}^{(2)}-J_{dj}^{(2)}\right)=G^{(0)}-G_{ij}^{(2)}$
1	-1	$\ln\cosh\left(J_{di}^{(2)}-J_{dj}^{(2)}\right)=G^{(0)}-G_{ij}^{(2)}$
1	1	$\ln\cosh\left(J_{di}^{(2)}+J_{dj}^{(2)}\right)=G^{(0)}+G_{ij}^{(2)}$

Table 3.1: Serial association equations.

When all possible combinations of values for S_i and S_j are written down we arrive to the set of equations from table 3.1; such equations read as

$$\ln\cosh\left(J_{di}^{(2)}-J_{dj}^{(2)}\right)=G^{(0)}-G_{ij}^{(2)} \ , \tag{3.29}$$

$$\ln\cosh\left(J_{di}^{(2)}+J_{dj}^{(2)}\right)=G^{(0)}+G_{ij}^{(2)} \ , \tag{3.30}$$

this leads to

$$G_{ij}^{(2)} = \frac{1}{2}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)}\right)}\right) \quad .$$

This expression may be worked out to match with the original one from Ref. [Saul and Jordan, 1994], which reads as follows

$$\tanh\left(G_{ij}^{(2)}\right) = \tanh\left(J_{di}^{(2)}\right)\tanh\left(J_{dj}^{(2)}\right) \quad , \tag{3.31}$$

hence

$$\begin{aligned}
G_{ij}^{(2)} &= \frac{1}{2}\ln\left(\frac{\cosh\left(J_{di}^{(2)}\right)\cosh\left(J_{dj}^{(2)}\right) + \sinh\left(J_{di}^{(2)}\right)\sinh\left(J_{dj}^{(2)}\right)}{\cosh\left(J_{di}^{(2)}\right)\cosh\left(J_{dj}^{(2)}\right) - \sinh\left(J_{di}^{(2)}\right)\sinh\left(J_{dj}^{(2)}\right)}\right) \\
&= \ln\left(\sqrt{\frac{1 + \frac{\sinh\left(J_{di}^{(2)}\right)\sinh\left(J_{dj}^{(2)}\right)}{\cosh\left(J_{di}^{(2)}\right)\cosh\left(J_{dj}^{(2)}\right)}}{1 - \frac{\sinh\left(J_{di}^{(2)}\right)\sinh\left(J_{dj}^{(2)}\right)}{\cosh\left(J_{di}^{(2)}\right)\cosh\left(J_{dj}^{(2)}\right)}}}\right) \\
&= \ln\left(\sqrt{\frac{1 + \tanh\left(J_{di}^{(2)}\right)\tanh\left(J_{dj}^{(2)}\right)}{1 - \tanh\left(J_{di}^{(2)}\right)\tanh\left(J_{dj}^{(2)}\right)}}\right) \\
&= \operatorname{atanh}\left(\tanh\left(J_{di}^{(2)}\right)\tanh\left(J_{dj}^{(2)}\right)\right) \quad .
\end{aligned}$$

Finally

$$\tanh\left(G_{ij}^{(2)}\right) = \tanh\left(J_{di}^{(2)}\right)\tanh\left(J_{dj}^{(2)}\right) \quad .$$

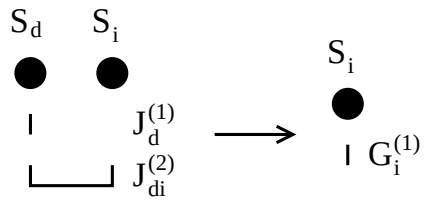

Figure 3.7: Serial association between a bias term and a weight.

Notice that this association can also be used to perform serial association between a bias term and a weight, as depicted in Fig. 3.7. If Eq. 3.28 is taken and unit S_j is clamped at $S_j = 1$ one arrives to

$$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} S_i\right) = G^{(0)} + G_i^{(1)} S_i \ , \tag{3.32}$$

the following equations are obtained

$$\begin{aligned} \ln\cosh\left(J_d^{(1)} - J_{di}^{(2)}\right) &= G^{(0)} - G_i^{(1)} \ , \\ \ln\cosh\left(J_d^{(1)} + J_{di}^{(2)}\right) &= G^{(0)} + G_i^{(1)} \ , \end{aligned}$$

which can finally be written down as

$$G_i^{(1)} = \frac{1}{2}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_d^{(1)}\right)}{\cosh\left(J_{di}^{(2)} - J_d^{(1)}\right)}\right) \ . \tag{3.33}$$

3.2.4 Star-triangle decimation

Finally, we describe the most complex structure that decimation is able to handle [Rüger, 1997] and the process that is carried out when it is decimated. This association, which is known as *star-triangle decimation*, transforms a non-biased unit S_d that is connected to units S_i, S_j and S_k by weights $J_{di}^{(2)}$, $J_{dj}^{(2)}$ and $J_{dk}^{(2)}$ to a new structure that is composed of units S_i, S_j, S_k and the weights $G_{ij}^{(2)}$, $G_{ik}^{(2)}$, $G_{jk}^{(2)}$ that link them; graphic for this conversion is shown in Fig. 3.8.

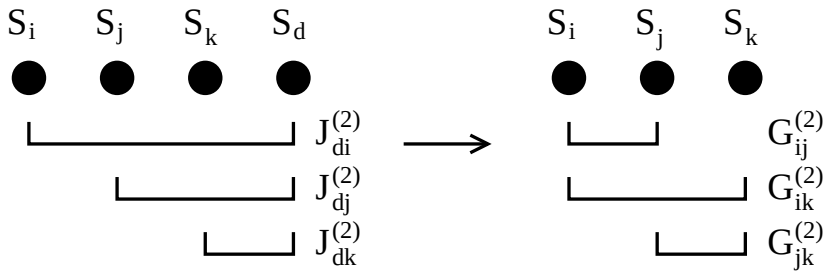

Figure 3.8: Star-triangle conversion.

The star-triangle association allows more complex structures to be decimated, thus increasing the number of connections that can be used in a given BM. The structures which allow usage of parallel, serial and star-triangle decimation are known as *decimatable* and a typical structure of this kind is depicted in Fig. 3.9. Notice that this structure is more densely connected than the one from Fig. 3.6, which was the one that allowed only serial association to be applied. The equations related to this process are written down as follows

$$\ln\left(\frac{1}{2}\sum_{S_d=-1}^{+1} \mathrm{e}^{J_{di}^{(2)}S_dS_i+J_{dj}^{(2)}S_dS_j+J_{dk}^{(2)}S_dS_k}\right) = G^{(0)} + G_{ij}^{(2)}S_iS_j + G_{ik}^{(2)}S_iS_k + G_{jk}^{(2)}S_jS_k \ ,$$
$$\ln\cosh\left(J_{di}^{(2)}S_i + J_{dj}^{(2)}S_j + J_{dk}^{(2)}S_k\right) = G^{(0)} + G_{ij}^{(2)}S_iS_j + G_{ik}^{(2)}S_iS_k + G_{jk}^{(2)}S_jS_k \ , \quad (3.34)$$

where S_d is the decimated unit, which is connected to units S_i, S_j and S_k by the temperature normalized weights $J_{di}^{(2)}$, $J_{dj}^{(2)}$ and $J_{dk}^{(2)}$.

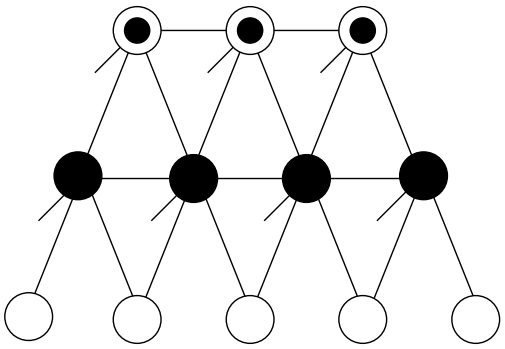

Figure 3.9: Decimatable structure using a number of connections that the star-triangle procedure can handle. Notice the bias terms and the weights linking the output units.

When one is willing to use the star-triangle decimation procedure, it is necessary to generate the system of equations by giving proper values to units S_i, S_j and S_k from Eq. 3.34. This leads to the system of equations that can be seen on table 3.2 and, since

S_i	S_j	S_k	$\ln\cosh\left(J_{di}^{(2)}S_i + J_{dj}^{(2)}S_j + J_{dk}^{(2)}S_k\right) =$ $= G^{(0)} + G_{ij}^{(2)}S_iS_j + G_{ik}^{(2)}S_iS_k + G_{jk}^{(2)}S_jS_k$
-1	-1	-1	$\ln\cosh\left(-J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)}$
-1	-1	1	$\ln\cosh\left(-J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)}$
-1	1	-1	$\ln\cosh\left(-J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)}$
-1	1	1	$\ln\cosh\left(-J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)}$
1	-1	-1	$\ln\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)}$
1	-1	1	$\ln\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)}$
1	1	-1	$\ln\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)}$
1	1	1	$\ln\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)}$

Table 3.2: Star-triangle transformation equations.

there are some repeated equations, to

$$\ln\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} \ , \tag{3.35}$$

$$\ln\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} \ , \tag{3.36}$$

$$\ln\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} \ , \tag{3.37}$$

$$\ln\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} \ , \tag{3.38}$$

their solution reads as

$$G_{ij}^{(2)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right)\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right)}\right), \tag{3.39}$$

$$G_{ik}^{(2)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right)\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right)}\right), \tag{3.40}$$

$$G_{jk}^{(2)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right)\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right)}\right). \tag{3.41}$$

This method does also work when any of the units is exchanged by a bias term [Rüger et al., 1996]. In such case, we arrive at a similar system of equations where $S_k = 1$,

$J_{dk}^{(2)} = J_d^{(1)}$, $G_{ik}^{(2)} = G_i^{(1)}$ and $G_{jk}^{(2)} = G_j^{(1)}$; this is

$$\ln\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_d^{(1)}\right) = G^{(0)} + G_{ij}^{(2)} - G_i^{(1)} - G_j^{(1)} \ , \tag{3.42}$$

$$\ln\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_d^{(1)}\right) = G^{(0)} - G_{ij}^{(2)} - G_i^{(1)} + G_j^{(1)} \ , \tag{3.43}$$

$$\ln\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_d^{(1)}\right) = G^{(0)} - G_{ij}^{(2)} + G_i^{(1)} - G_j^{(1)} \ , \tag{3.44}$$

$$\ln\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_d^{(1)}\right) = G^{(0)} + G_{ij}^{(2)} + G_i^{(1)} + G_j^{(1)} \ . \tag{3.45}$$

This system of equations is similar to the last one and its solution is

$$G_{ij}^{(2)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_d^{(1)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_d^{(1)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_d^{(1)}\right)\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_d^{(1)}\right)}\right) \ , \tag{3.46}$$

$$G_i^{(1)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_d^{(1)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_d^{(1)}\right)}{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_d^{(1)}\right)\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_d^{(1)}\right)}\right) \ , \tag{3.47}$$

$$G_j^{(1)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} - J_d^{(1)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} + J_d^{(1)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)} + J_d^{(1)}\right)\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)} - J_d^{(1)}\right)}\right) \ . \tag{3.48}$$

We finally show that serial association is a particular case of these equations, provided that $J_d^{(1)} = 0$. Then, $G_i^{(1)} = G_j^{(1)} = 0$ and

$$G_{ij}^{(2)} = \frac{1}{2}\ln\left(\frac{\cosh\left(J_{di}^{(2)} + J_{dj}^{(2)}\right)}{\cosh\left(J_{di}^{(2)} - J_{dj}^{(2)}\right)}\right) \ .$$

3.3 Correlations and expectation values

It has been shown that decimation is applied to reduce the size of a BM in order to analytically compute the quantities needed at the learning stage. We now explain how these correlations and expectation values

$$\Delta w_{ij}^{(2)} = \eta\left(\langle S_i S_j\rangle^* - \langle S_i S_j\rangle\right) \ ,$$
$$\Delta w_i^{(1)} = \eta\left(\langle S_i\rangle^* - \langle S_i\rangle\right) \ ,$$

are actually calculated.

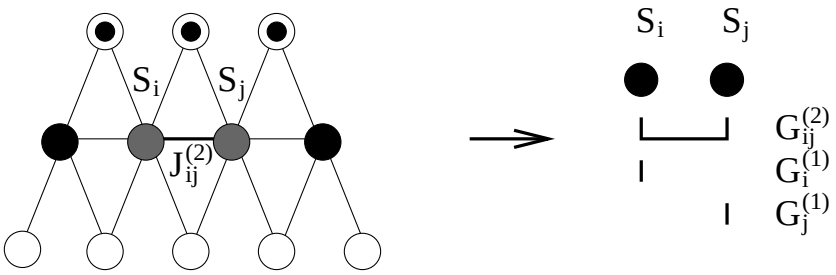

Figure 3.10: Applied example of decimation.

In this section, we will assume that there is a Boltzmann Machine model that has already been decimated, and thereafter we have a small set of neurons where correlations and expectation values have to be found. In this sense, the process shown in Fig. 3.10 has already been carried out, thus leading to a smaller structure where $\langle S_i S_j \rangle$ is computed. At this point, it is possible to analytically calculate $\langle S_i S_j \rangle$; the mean value for S_i is found by applying serial association as shown in Fig. 3.11.

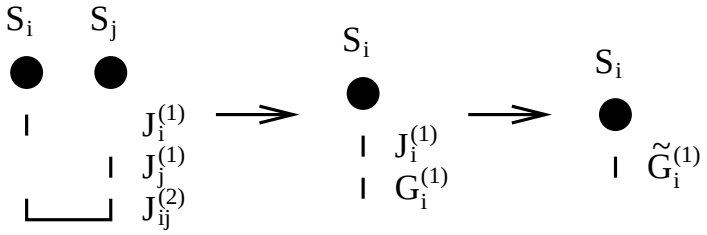

Figure 3.11: Decimation of a pair of units to a single one.

3.3.1 Expectation value for a single unit

We want now to calculate the expectation value for a single unit. This calculus is made once a pair of units has been decimated; one arrives then to the structure shown in Fig. 3.12, which is an isolated unit whose only connection is a first order weight $J_i^{(1)}$. We are going to find an analytical expression that provides the expectation value according to this connection, hence we begin with the mathematical expression for an expectation

value

$$\begin{aligned} \langle S_i \rangle &= \sum_{S_i=\pm 1} S_i \, p\left(S_i = \pm 1\right) \\ &= 1 \, p\left(S_i = 1\right) - 1 \, p\left(S_i = -1\right) \quad , \end{aligned} \tag{3.49}$$

where the probability distribution is the Boltzmann probability distribution

$$p\left(S_i = 1\right) = \frac{\mathrm{e}^{J_i^{(1)}}}{\mathrm{e}^{J_i^{(1)}} + \mathrm{e}^{-J_i^{(1)}}} \quad , \tag{3.50}$$

$$p\left(S_i = -1\right) = \frac{\mathrm{e}^{-J_i^{(1)}}}{\mathrm{e}^{J_i^{(1)}} + \mathrm{e}^{-J_i^{(1)}}} \quad , \tag{3.51}$$

we finally arrive at

$$\begin{aligned} \langle S_i \rangle &= \frac{\mathrm{e}^{J_i^{(1)}}}{\mathrm{e}^{J_i^{(1)}} + \mathrm{e}^{-J_i^{(1)}}} - \frac{\mathrm{e}^{-J_i^{(1)}}}{\mathrm{e}^{J_i^{(1)}} + \mathrm{e}^{-J_i^{(1)}}} \\ &= \frac{\mathrm{e}^{J_i^{(1)}} - \mathrm{e}^{-J_i^{(1)}}}{\mathrm{e}^{J_i^{(1)}} + \mathrm{e}^{-J_i^{(1)}}} \\ &= \tanh\left(J_i^{(1)}\right) \quad . \end{aligned} \tag{3.52}$$

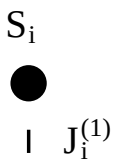

Figure 3.12: Single unit connected to bias term $J_i^{(1)}$.

3.3.2 Correlation of two free units

We now discuss the case where the correlation for two units that are set free is calculated, and provide the analytical expressions that are used to compute this value. We use here the term *free* to denote that these units would be able to change their state during the Monte Carlo learning process, regardless of whether we are on the learning free or clamped phase. Let S_i and S_j be two units linked by a temperature normalized weight $J_{ij}^{(2)}$ and two first order connections $J_i^{(1)}$, $J_j^{(1)}$ as depicted in Fig. 3.13. Notice then that

the decimation process has been carried out and that therefore, the other units of the neural network are represented by this final set of connections. It has already been shown that the remaining units from the neural network do still behave according to the same probability distribution, hence this two units will have the same correlation regardless than if we use the current connections to compute it or the whole neural network.

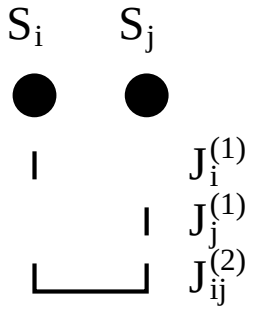

Figure 3.13: Two units structure connected by weight $J_{ij}^{(2)}$ and bias terms $J_i^{(1)}$, $J_j^{(1)}$.

The correlation value $\langle S_i S_j \rangle$ is calculated as follows

$$\begin{aligned} \langle S_i S_j \rangle &= \sum_{S_i, S_j = \pm 1} S_i \, S_j \, p\left(S_i, S_j = \pm 1\right) \\ &= \sum_{S_i, S_j = \pm 1} S_i \, S_j \frac{\mathrm{e}^{-\mathcal{E}}}{\mathcal{Z}} \ , \end{aligned} \tag{3.53}$$

where

$$\mathcal{E} = -S_i S_j J_{ij}^{(2)} - S_i J_i^{(1)} - S_j J_j^{(1)} \ , \tag{3.54}$$

and

$$\begin{aligned} \mathcal{Z} &= \sum_{\forall \gamma} \mathrm{e}^{-\mathcal{E}_\gamma} \\ &= \mathrm{e}^{J_{ij}^{(2)} + J_i^{(1)} + J_j^{(1)}} + \mathrm{e}^{J_{ij}^{(2)} - J_i^{(1)} - J_j^{(1)}} + \mathrm{e}^{-J_{ij}^{(2)} + J_i^{(1)} - J_j^{(1)}} + \mathrm{e}^{-J_{ij}^{(2)} - J_i^{(1)} + J_j^{(1)}} \\ &= 2\left(\mathrm{e}^{J_{ij}^{(2)}} \cosh\left(J_i^{(1)} + J_j^{(1)}\right) + \mathrm{e}^{-J_{ij}^{(2)}} \cosh\left(J_i^{(1)} + J_j^{(1)}\right)\right) \ . \end{aligned} \tag{3.55}$$

Hence, finally

$$\begin{aligned}\langle S_i S_j \rangle &= \frac{1}{\mathcal{Z}}\left(\mathrm{e}^{J_{ij}^{(2)}-J_i^{(1)}-J_j^{(1)}}+\mathrm{e}^{J_{ij}^{(2)}+J_i^{(1)}+J_j^{(1)}}-\mathrm{e}^{-J_{ij}^{(2)}+J_i^{(1)}-J_j^{(1)}}-\mathrm{e}^{-J_{ij}^{(2)}-J_i^{(1)}+J_j^{(1)}}\right) \\ &= \frac{\mathrm{e}^{J_{ij}^{(2)}}\cosh\left(J_i^{(1)}+J_j^{(1)}\right)-\mathrm{e}^{-J_{ij}^{(2)}}\cosh\left(J_i^{(1)}-J_j^{(1)}\right)}{\mathrm{e}^{J_{ij}^{(2)}}\cosh\left(J_i^{(1)}+J_j^{(1)}\right)+\mathrm{e}^{-J_{ij}^{(2)}}\cosh\left(J_i^{(1)}-J_j^{(1)}\right)} \ . \end{aligned} \tag{3.56}$$

3.3.3 Correlation of a free and a clamped connected units

We now discuss the situation where a unit that is set free and is then able to change its state during the Monte Carlo simulation process is connected to a clamped unit. Again, we refer to a clamped unit as the neuron that is not able to change its state. This would be the case, either for an input unit when the learning process is carried out or for an output unit when clamped phase at learning stage happens. We are interested in calculating the correlation between this pair of units, because this quantity is needed during the learning process. Let S_i and S_j be a pair of units as depicted in Fig. 3.14, where S_i is a free unit and $S_j = S_j^*$ remains clamped, herein the $*$ symbol. We will determine an analytical expression for $\langle S_i S_j^* \rangle$.

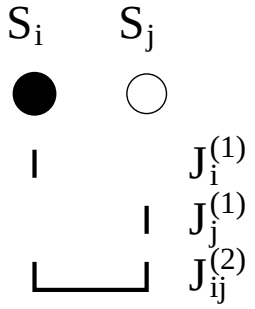

Figure 3.14: Correlation between a free and a clamped units.

If we apply the definition of expectation value

$$\langle S_i S_j^* \rangle = \sum_{S_i=\pm 1} S_i S_j^* \frac{\mathrm{e}^{J_{ij}^{(2)} S_i S_j^* + J_i^{(1)} S_i}}{\mathcal{Z}} , \tag{3.57}$$

the sum is only carried out accounting unit S_i, because S_j is clamped and its state can

not be changed. Thus

$$\begin{aligned} \left\langle S_i S_j^* \right\rangle &= S_j^* \left(\frac{\mathrm{e}^{J_{ij}^{(2)} S_j^* + J_i^{(1)}} - \mathrm{e}^{-J_{ij}^{(2)} S_j^* - J_i^{(1)}}}{\mathcal{Z}} \right) \\ &= S_j^* \tanh\left(J_{ij}^{(2)} S_j^* + J_i^{(1)} \right) , \end{aligned} \tag{3.58}$$

which is nothing else than a parallel association between the bias term $J_i^{(1)}$ and the product $J_{ij}^{(2)} S_j^*$.

3.4 High order Decimation

In this section, the high order Decimation method is presented and discussed in four parts: we first analyze the limits of the standard decimation process, while the second part proceeds with the concept that is used to overcome them, thus showing the master equation that is used for the high order Decimation process. The section is concluded with a numerical example that is carried out step by step, thus following all the calculus that are done on a simple, HOD process.

3.4.1 Biased star-triangle decimation

Decimation, as explained so far, is not able to handle some kind of topologies. We begin this discussion with the structure depicted in Fig. 3.15, where a central S_d unit is at the center of a triangle, connected to three standard units S_i, S_j, S_k by temperature normalized weights $J_{di}^{(2)}$, $J_{dj}^{(2)}$ and $J_{dk}^{(2)}$ and to an external unit by a temperature normalized bias term $J_d^{(1)}$. We will name this grouping as the biased star-triangle structure because it can be depicted as having this shape.

The structure proposed in such picture can not be decimated [Rüger et al., 1996]

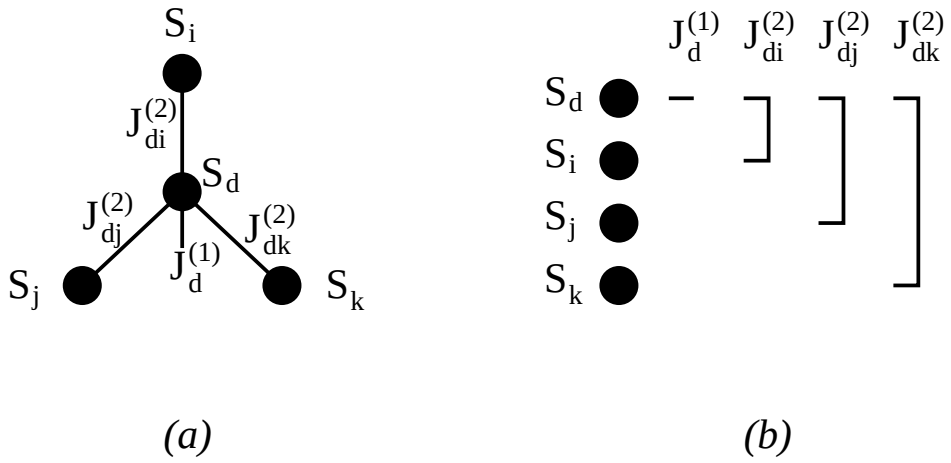

Figure 3.15: Non decimatable, biased star-triangle structure with typical notation (a) and our notation (b).

because the system of equations does not have enough degrees of freedom

$$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \tag{3.59}$$

$$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \tag{3.60}$$

$$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \tag{3.61}$$

$$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \tag{3.62}$$

$$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \tag{3.63}$$

$$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \tag{3.64}$$

$$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \tag{3.65}$$

$$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) = G^{(0)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \tag{3.66}$$

notice that for this system there are 8 equations but only 4 unknown terms. The system is not compatible because Eqs. 3.59, 3.60, 3.61 and 3.62 are the same expressions as Eqs. 3.66, 3.65, 3.64 and 3.63 respectively, but with different values in the left hand side (lhs) of the equations. We want to obtain a system of equations that can be solved, so we will enter as many variables as possible to reach 8 unknown terms and generate a system with 8 equations and 8 unknown terms. We begin by introducing the set of bias

terms that is added to the resulting structure

$$\begin{aligned}
\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) &= G^{(0)} - G_i^{(1)} - G_j^{(1)} - G_k^{(1)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) &= G^{(0)} - G_i^{(1)} - G_j^{(1)} + G_k^{(1)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) &= G^{(0)} - G_i^{(1)} + G_j^{(1)} - G_k^{(1)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) &= G^{(0)} - G_i^{(1)} + G_j^{(1)} + G_k^{(1)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) &= G^{(0)} + G_i^{(1)} - G_j^{(1)} - G_k^{(1)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) &= G^{(0)} + G_i^{(1)} - G_j^{(1)} + G_k^{(1)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) &= G^{(0)} + G_i^{(1)} + G_j^{(1)} - G_k^{(1)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} \quad , \\
\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) &= G^{(0)} + G_i^{(1)} + G_j^{(1)} + G_k^{(1)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} \quad ,
\end{aligned} \tag{3.67}$$

but this solution leads to a system of eight equations with seven unknowns, this is yet a non compatible system of equations. We need to introduce an eighth element which makes the system solvable: the only feasible solution is using a higher order term [Sejnowski, 1987, Farguell et al., 2006] that would at least grant a sufficient number of unknowns. Notice, however, that even by having such a drawback, decimation has been used so far for pattern recognition over a set of images [Nijman and Kappen, 1996] and for medical diagnosis [Rüger, 1997], thus proving than standard decimation is suitable for solving some learning problems.

3.4.2 The HOBM applied to decimation

The HOBM [Sejnowski, 1987] is an extension of the Boltzmann Machine where weights may connect more than two units. These are known as *high order weights* and the resulting BM model is typically referred to as a high order Boltzmann Machine. A typical high order connection is depicted in Fig. 3.16, though this one is linking three units such weights may connect up to N units on an N units neural network.

The energy functional is changed to allow the same dynamics with this new set of

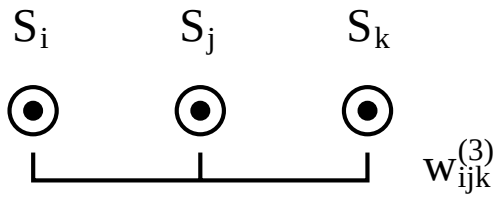

Figure 3.16: Third order weight linking three output units.

connections

$$E = -\sum_{n=1,\sigma}^{N} w_{\sigma}^{(n)} \prod_{\rho \in \sigma} S_{\rho} \ ,$$

and weights, regardless of their order, are again temperature normalized

$$J_{\sigma}^{(n)} = \frac{w_{\sigma}^{(n)}}{T} \ .$$

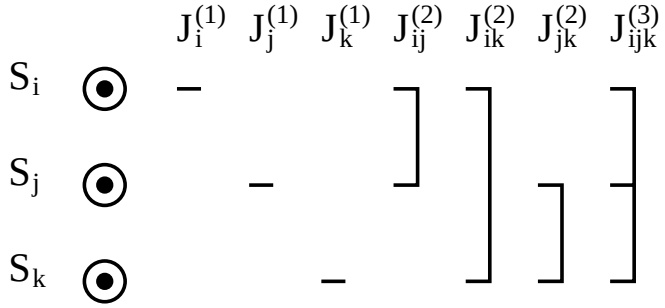

Figure 3.17: Third order smallest possible neural network.

A fully connected, three units neural network with temperature normalized weights is depicted in Fig. 3.17. Notice also that this is the minimal structure that allows inclusion of high order terms, as there are three units. We will consider bias terms as first order weights and standard connections as second order ones. Now, we recall the biased star-triangle association that had no solution in the previous section and add a third order element as an unknown. The transformation which is taking place is schematically described in

Fig. 3.18, where we decimate unit S_d. The related equation reads

$$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} S_i + J_{dj}^{(2)} S_j + J_{dk}^{(2)} S_k\right) = \tag{3.68}$$
$$= G^{(0)} + G_i^{(1)} S_i + G_j^{(1)} S_j + G_k^{(1)} S_k + G_{ij}^{(2)} S_i S_j + G_{ik}^{(2)} S_i S_k + G_{jk}^{(2)} S_j S_k + G_{ijk}^{(3)} S_i S_j S_k \ ,$$

this results on linear system of eight equations with eight unknowns, as represented in table 3.3.

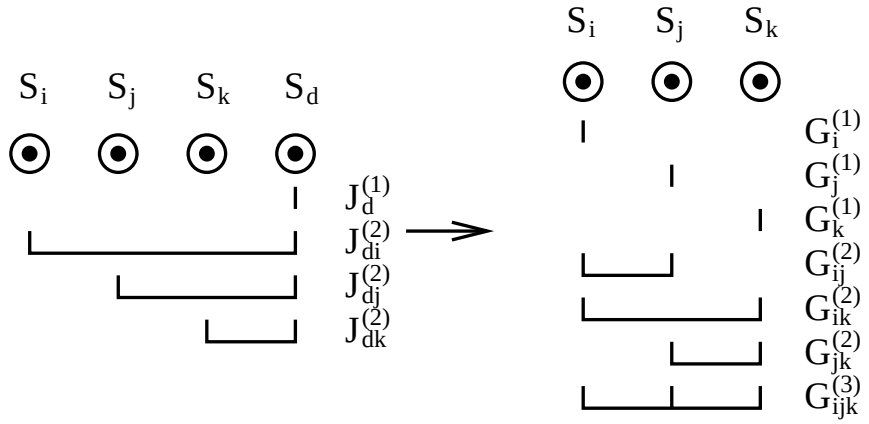

Figure 3.18: Third order star-triangle conversion.

When such system is solved, we obtain

$$\begin{aligned}
G_{ijk}^{(3)} &= \frac{1}{8}\ln\left(\frac{A_1A_2A_4A_7}{A_0A_3A_5A_6}\right) \ , \\
G_{ij}^{(2)} &= \frac{1}{8}\ln\left(\frac{A_0A_1A_6A_7}{A_2A_3A_4A_5}\right) \ , \\
G_{ik}^{(2)} &= \frac{1}{8}\ln\left(\frac{A_0A_2A_5A_7}{A_1A_3A_4A_6}\right) \ , \\
G_{jk}^{(2)} &= \frac{1}{8}\ln\left(\frac{A_0A_3A_4A_7}{A_1A_2A_5A_6}\right) \ , \\
G_{i}^{(1)} &= \frac{1}{8}\ln\left(\frac{A_4A_5A_6A_7}{A_0A_1A_2A_3}\right) \ , \\
G_{j}^{(1)} &= \frac{1}{8}\ln\left(\frac{A_2A_3A_6A_7}{A_0A_1A_4A_5}\right) \ , \\
G_{k}^{(1)} &= \frac{1}{8}\ln\left(\frac{A_1A_3A_5A_7}{A_0A_2A_4A_6}\right) \ ,
\end{aligned} \tag{3.69}$$

S_i	S_j	S_k	$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} S_i + J_{dj}^{(2)} S_j + J_{dk}^{(2)} S_k\right)$
-1	-1	-1	$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) =$ $= -G_{ijk}^{(3)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} - G_i^{(1)} - G_j^{(1)} - G_k^{(1)} + G^{(0)} = \ln A_0$
-1	-1	1	$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) =$ $= G_{ijk}^{(3)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} - G_i^{(1)} - G_j^{(1)} + G_k^{(1)} + G^{(0)} = \ln A_1$
-1	1	-1	$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) =$ $= G_{ijk}^{(3)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} - G_i^{(1)} + G_j^{(1)} - G_k^{(1)} + G^{(0)} = \ln A_2$
-1	1	1	$\ln\cosh\left(J_d^{(1)} - J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) =$ $= -G_{ijk}^{(3)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} - G_i^{(1)} + G_j^{(1)} + G_k^{(1)} + G^{(0)} = \ln A_3$
1	-1	-1	$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} - J_{dj}^{(2)} - J_{dk}^{(2)}\right) =$ $= G_{ijk}^{(3)} - G_{ij}^{(2)} - G_{ik}^{(2)} + G_{jk}^{(2)} + G_i^{(1)} - G_j^{(1)} - G_k^{(1)} + G^{(0)} = \ln A_4$
1	-1	1	$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} - J_{dj}^{(2)} + J_{dk}^{(2)}\right) =$ $= -G_{ijk}^{(3)} - G_{ij}^{(2)} + G_{ik}^{(2)} - G_{jk}^{(2)} + G_i^{(1)} - G_j^{(1)} + G_k^{(1)} + G^{(0)} = \ln A_5$
1	1	-1	$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} + J_{dj}^{(2)} - J_{dk}^{(2)}\right) =$ $= -G_{ijk}^{(3)} + G_{ij}^{(2)} - G_{ik}^{(2)} - G_{jk}^{(2)} + G_i^{(1)} + G_j^{(1)} - G_k^{(1)} + G^{(0)} = \ln A_6$
1	1	1	$\ln\cosh\left(J_d^{(1)} + J_{di}^{(2)} + J_{dj}^{(2)} + J_{dk}^{(2)}\right) =$ $= G_{ijk}^{(3)} + G_{ij}^{(2)} + G_{ik}^{(2)} + G_{jk}^{(2)} + G_i^{(1)} + G_j^{(1)} + G_k^{(1)} + G^{(0)} = \ln A_7$

Table 3.3: Third order equations for the star-triangle conversion.

arriving to a general equivalence for the star-triangle system with a central biased unit. Notice though that standard star-triangle equations can be found by setting $J_d^{(1)} = 0$, since it will make the system *loose* four equations. This is a starting point to obtain the serial association expressions. However, this method allows us to reduce a second order Boltzmann Machine structure that was considered as non-decimatable [Rüger et al., 1996] to a third order equivalent neural network.

Now that we know how to solve the biased star-triangle transformation, we can inquiry if adding high order terms would always provide a solution to decimate any given topology of a Boltzmann Machine. This concept of adding high order weights to the decimated

network constitutes the basic idea behind the *high order Decimation* procedure. In this sense, we now consider S_d to be a biased neuron from a BM model which is connected to other units S_1 to S_N. Weights up to order N are added in the decimation expression and, as a result, a fully connected N-th order neural network is obtained. The corresponding equations have the following form

$$\ln\cosh\left(J_d^{(1)} + \sum_i J_{di}^{(2)} S_i\right) = \\ = G^{(0)} + \textstyle\sum_i G_i^{(1)} S_i + \sum_{j<i} G_{ij}^{(2)} S_i S_j + \sum_{k<j<i} G_{ijk}^{(3)} S_i S_j S_k + \ldots \ . \tag{3.70}$$

We now count the total number of equations required to perform high order decimation, as there are

- $\begin{pmatrix} N \\ 0 \end{pmatrix} = 1$ normalization constants $G^{(0)}$,
- $\begin{pmatrix} N \\ 1 \end{pmatrix} = N$ biases $G_i^{(1)}$,
- $\begin{pmatrix} N \\ 2 \end{pmatrix} = \frac{N(N-1)}{2}$ second order weights $G_{ij}^{(2)}$,
- $\begin{pmatrix} N \\ 3 \end{pmatrix} = \frac{N(N-1)(N-2)}{6}$ third order terms $G_{ijk}^{(3)}$,
- ...
- $\begin{pmatrix} N \\ N \end{pmatrix} = 1$ N-th order $G_{12\ldots N}^{(N)}$ weight.

However, this is a known identity

$$\begin{pmatrix} N \\ 0 \end{pmatrix} + \begin{pmatrix} N \\ 1 \end{pmatrix} + \begin{pmatrix} N \\ 2 \end{pmatrix} + \ldots + \begin{pmatrix} N \\ N \end{pmatrix} = 2^N \ ,$$

making it for a total of 2^N variables. Since there are N units, these can take 2^N combinations, hence there are 2^N equations and a 2^N unknown terms. The HOD process

is schematically shown in Fig. 3.19, where the originally second order neural network of Fig. 3.19a is decimated to produce the result of Fig. 3.19b. It is shown in the appendix that the matrix associated to the resulting system of equations is a $2^N \times 2^N$ Hadamard type [Sylvester, 1867] and that therefore its determinant is always different from zero. Hadamard matrices are a family of square matrices which are widely used in the communication area and have some interesting properties: let $H_{2^N \times 2^N}$ be a Hadamard matrix of size 2^N, it is shown in the appendix that

$$\det \{H_{2^N \times 2^N}\} \neq 0 \ , \tag{3.71}$$

$$H_{2^N \times 2^N} \cdot H^T_{2^N \times 2^N} = H^T_{2^N \times 2^N} \cdot H_{2^N \times 2^N} = 2^N I \ , \tag{3.72}$$

$$H^{-1}_{2^N \times 2^N} = \frac{1}{2^N} H^T_{2^N \times 2^N} \ . \tag{3.73}$$

In this way, the system of equations has always a solution that is unique because a Hadamard matrix does always have an inverse.

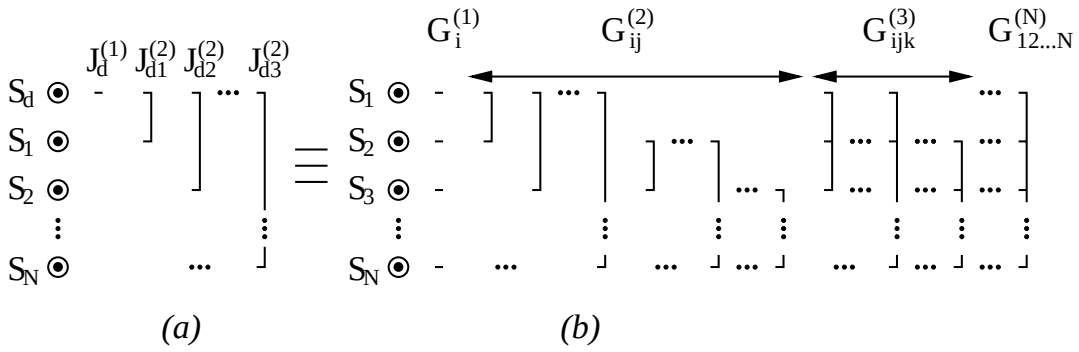

Figure 3.19: Original (a) and decimated (b) structures.

When the original network is already of high order, all weights connected to the unit to be decimated must be taken into account in the lhs of the previous expression, leading to the HOD master equation

$$\ln \cosh \left(J^{(1)}_d + \sum_i J^{(2)}_{di} S_i + \sum_{j<i} J^{(3)}_{dij} S_i S_j + ... + J^{(N+1)}_{d123...N} S_1 S_2 S_3 \ldots S_N \right) =$$
$$= G^{(0)} + \textstyle\sum_i G^{(1)}_i S_i + \sum_{j<i} G^{(2)}_{ij} S_i S_j + \sum_{k<j<i} G^{(3)}_{ijk} S_i S_j S_k + ... \ . \tag{3.74}$$

Notice that the equations for the standard BM, where only two-body weights are considered, can be recovered from this expression by setting $J_\sigma^{(n>2)} = 0$. Once the system is solved, a new Boltzmann Machine with one less unit is left, although the resulting network is highly connected due to the inclusion of the new, high order weights. In order to compute the needed n-th order correlations appearing in the weight update rule for the Boltzmann Machine learning algorithm, the process is iterated until all the required units are decimated.

3.4.3 HOD numerical example

We have already seen that HOD can be applied to any Boltzmann Machine structure, regardless of its order. A numerical example over the neural network depicted in Fig. 3.20, which is a fourth order HOBM, is now carried out. In this example, the value of the correlation $< S_2 S_3 >$ is calculated. Notice however that a learning process would require to repeat this same algorithm for each connection from the neural network.

We initialize the normalized weights $\left\{ J_\sigma^{(n)} \right\}$ of this model randomly and in the range $[-1, +1]$, hence

$$
\begin{array}{llll}
J_1^{(1)} = 0.86 & J_2^{(1)} = -0.068 & J_3^{(1)} = -0.163 & J_4^{(1)} = 0.69 \\
J_{12}^{(2)} = 0.050 & J_{13}^{(2)} = -0.60 & J_{14}^{(2)} = 0.34 & \\
J_{23}^{(2)} = 0.68 & J_{24}^{(2)} = -0.96 & J_{34}^{(2)} = 0.36 & \\
J_{123}^{(3)} = -0.24 & J_{124}^{(3)} = 0.66 & J_{134}^{(3)} = 0.0056 & J_{234}^{(3)} = 0.42 \\
J_{1234}^{(4)} = -0.14 & & &
\end{array}
$$

we now apply these values to Eq. 3.74 which, for this example, reads as

$$
\begin{aligned}
&\ln\cosh\Big(J_4^{(1)} S_4 + J_{14}^{(2)} S_1 S_4 + J_{24}^{(2)} S_2 S_4 + J_{34}^{(2)} S_3 S_4 + \\
&\qquad J_{124}^{(3)} S_1 S_2 S_4 + J_{134}^{(3)} S_1 S_3 S_4 + J_{234}^{(3)} S_2 S_3 S_4 + J_{1234}^{(4)} S_1 S_2 S_3 S_4 \Big) = \\
&= G^{(0)} + G_1^{(1)} S_1 + G_2^{(1)} S_2 + G_3^{(1)} S_3 + \\
&\qquad + G_{12}^{(2)} S_1 S_2 + G_{13}^{(2)} S_1 S_3 + G_{23}^{(2)} S_2 S_3 + G_{123}^{(3)} S_1 S_2 S_3 \ .
\end{aligned} \tag{3.75}
$$

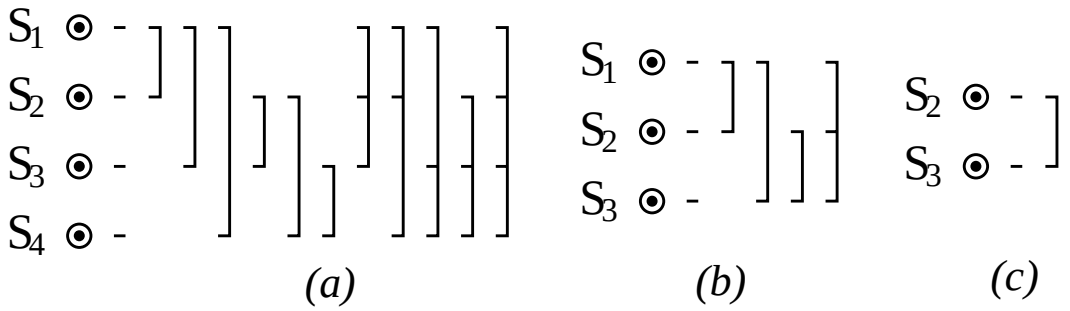

Figure 3.20: Decimation process to compute correlation $< S_2 S_3 >$.

When the system of equations is solved, one arrives to these values

$$\begin{aligned} &G^{(0)} = 1.1 \\ &G_1^{(1)} = -0.45 \quad G_2^{(1)} = 0.077 \quad G_3^{(1)} = -0.79 \\ &G_{12}^{(2)} = -0.30 \quad G_{13}^{(2)} = 0.22 \quad G_{23}^{(2)} = -0.045 \\ &G_{123}^{(3)} = 0.31 \end{aligned}$$

which are added to the original connections. This leads then to a new set of weights $\left\{ J_\sigma^{(n)'} \right\}$, as shown in Fig. 3.20b. These are

$$\begin{aligned} &J_1^{(1)'} = 0.41 \quad J_2^{(1)'} = 0.010 \quad J_3^{(1)'} = -0.86 \\ &J_{12}^{(2)'} = 0.20 \quad J_{13}^{(2)'} = -0.38 \quad J_{23}^{(2)'} = 0.63 \\ &J_{123}^{(3)'} = 0.070 \end{aligned}$$

that are again set into the HOD equation. We decimate now unit S_1 and Eq. 3.74 reads then as

$$\begin{aligned} &\ln \cosh \left(J_1^{(1)'} S_1 + J_{12}^{(2)'} S_1 S_2 + J_{13}^{(2)'} S_1 S_3 + J_{23}^{(2)'} S_2 S_3 + J_{123}^{(3)'} S_1 S_2 S_3 \right) = \\ &= G^{(0)'} + G_2^{(1)'} S_2 + G_3^{(1)'} S_3 + G_{23}^{(2)'} S_2 S_3 \ . \end{aligned} \tag{3.76}$$

The set $\left\{ G_\sigma^{(n)'} \right\}$ that is obtained is the following

$$\begin{aligned} &G^{(0)'} = 0.65 \\ &G_2^{(1)'} = 0.19 \quad G_3^{(1)'} = -0.51 \\ &G_{23}^{(2)'} = -0.15 \end{aligned}$$

we arrive then to the weights that are depicted in Fig. 3.74c, which are referred to as $\left\{J_{\sigma}^{(n)''}\right\}$. These values are

$$J_2^{(1)''} = 0.20 \qquad J_3^{(1)''} = -1.37$$
$$J_{23}^{(2)''} = -0.08$$

the value of the correlation $< S_2 S_3 >$ is then calculated by using Eq. 3.56. The final result is then

$$< S_2 S_3 >= -0.25 \ , \tag{3.77}$$

notice again that this process has to be carried out for each pair of connected units, in order to calculate the quantities that are needed to carry out the learning process.

3.5 Multiple unit decimation process

In this section an extension of the high order Decimation method is proposed by decimating a given number of units at once. The final result of the neural network is yet the same, though in this sense the intermediate steps where high order weights are added in parallel structures are not used anymore.

In the first part of this section, we calculate the marginal probability distribution sum for a multiple decimation process and show it to be the same calculus that is used on an iterative HOD process. We then proceed on a simple case for a BM with two hidden units whose decimated results are numerically proven to be the same as per standard decimation. This section concludes then with the generalization of the multiple decimation instance, thus allowing further analysis of the decimation process and a comparative analysis on the results that are obtained by following an iterative decimation procedure and the multiple decimation algorithm.

3.5.1 Iterative HOD and the Multiple Decimation equivalence

The high order Decimation process carries out decimation for a high order energy functional. In this sense, Eq. 3.7 is applied to a HOBM model and a sum over the possible states of the unit that is being decimated is carried out. Consider now a given neural network with $N+1$ units, where the decimation process leads to an equivalent model with N units. The HOD master equation reads as

$$\ln\left(\frac{1}{2}\sum_{S_d=-1}^{+1}\mathrm{e}^{\sum J_{d\sigma}^{(n)}S_d\prod_{\rho\in\sigma}S_\rho}\right)=G^{(0)'}+\sum_{n=1,\ \sigma}^{n=N}G_\sigma^{(n)}\prod_{\rho\in\sigma}S_\rho\ . \tag{3.78}$$

where S_d is the unit to decimate and σ represents the labels that weight $G_\sigma^{(n)}$ connects; notice also that $G^{(0)'}$ has been explicitly separated from the rest of the weights. This expression is used within the marginal probability sum

$$p\left(\alpha\right)=p\left(\alpha_d\right)|_{S_d=1}+p\left(\alpha_d\right)|_{S_d=-1}\ \ , \tag{3.79}$$

where α_d is an energy state which depends on both a set of units $\mathcal{S}$ and a certain unit S_d that is being decimated. Decimating two units S_{d_1} and S_{d_2} at the same time would therefore be equivalent to

$$p\left(\alpha\right)= \tag{3.80}$$
$$p\left(\alpha_D\right)|_{S_{d_1}=1,S_{d_2}=1}+p\left(\alpha_D\right)|_{S_{d_1}=1,S_{d_2}=-1}+p\left(\alpha_D\right)|_{S_{d_1}=-1,S_{d_2}=1}+p\left(\alpha_D\right)|_{S_{d_1}=-1,S_{d_2}=-1}\ \ ,$$

where α_D depends on a given set of units $\mathcal{S}$ that are connected to the two units that are being decimated. We consider now the decimation process for a set of M units $\mathcal{S}_d$ on a given neural network that originally had a total of $N+M$ neurons. In this sense, the previous expression now becomes

$$p\left(\alpha\right)=\sum_{\mathcal{S}_d}p\left(\alpha_D\right)\ \ , \tag{3.81}$$

and the sum is carried out for all the values that the units from $\mathcal{S}_d$ can take. The HOD master expression from Eq. 3.78 becomes then

$$\ln\left(\frac{1}{2}\sum_{\mathcal{S}_d}\mathrm{e}^{\sum_{n=1,\ \sigma}^{n=N+M}J_{d\sigma}^{(n)}S_d\prod_{\rho\in\sigma}S_\rho}\right)=G^{(0)'}+\sum_{n=1,\ \sigma}^{n=N}G_\sigma^{(n)}\prod_{\rho\in\sigma}S_\rho\ . \tag{3.82}$$

where $\left\{G_{\sigma}^{(n)}\right\}$ is the new set of weights that connects all the units from set $\mathcal{S}$. We do also introduce the term $G^{(0)}$ instead of $G^{(0)'}$, which absorbs $-\ln\frac{1}{2}$, and put this term into the sum

$$\ln\left(\sum_{\mathcal{S}_d} \mathrm{e}^{\sum_{n=1,\ \sigma}^{n=N+M} J_{d\sigma}^{(n)} S_d \prod_{\rho\in\sigma} S_\rho}\right) = \sum_{n=0,\ \sigma}^{n=N} G_{\sigma}^{(n)} \prod_{\rho\in\sigma} S_\rho \ , \tag{3.83}$$

thus becoming the multiple decimation master equation.

3.5.2 Two units decimation

We have seen that a certain structure can be decimated on a single step by carrying out a sum over all the units that are being decimated. In this sense, it becomes a simpler process than carrying out a multiple set of HOD processes. We now propose a simple, second order structure with two output units and a hidden layer with two neurons that is represented in Fig. 3.21. This structure will be first decimated by using HOD, since this method will generate a third order weight for this topology. This weight will then be decimated to reach a simpler structure only with the output units. The results that are obtained by carrying out this process will then be compared to the ones that are found by carrying out Multiple Decimation.

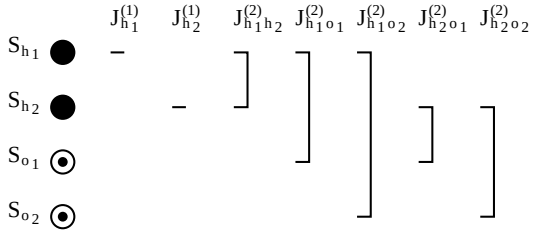

Figure 3.21: Two hidden and two output neurons BM.

We now proceed to write down the equations for a multiple decimation process assuming that the conditioned probability distribution of the output neurons remains unchanged

$$\sum_{S_{h_1}S_{h_2}} \mathrm{e}^{S_{h_1}\left(J_{h_1}^{(1)}+J_{h_1o_1}^{(2)}S_{o_1}+J_{h_1o_2}^{(2)}S_{o_2}\right)+S_{h_2}\left(J_{h_2}^{(1)}+J_{h_1o_1}^{(2)}S_{o_1}+J_{o_2h_2}^{(2)}S_{o_2}\right)+J_{h_1h_2}^{(2)}S_{h_1}S_{h_2}} =$$

$$= \mathrm{e}^{G^{(0)}+G_{o_1}^{(1)}S_{o_1}+G_{o_2}^{(1)}S_{o_2}+G_{o_1o_2}^{(2)}S_{o_1}S_{o_2}} \ , \tag{3.84}$$

where the set of weights $\left\{J_{\sigma}^{(n)}\right\}$ is known and $\left\{G_{\sigma}^{(n)}\right\}$ is the unknown set to be found. When logarithm is taken at both sides of the equation we arrive at

$$\ln\left[\sum_{S_{h_1}S_{h_2}} \mathrm{e}^{S_{h_1}\left(J_{h_1}^{(1)}+J_{h_1o_1}^{(2)}S_{o_1}+J_{h_1o_2}^{(2)}S_{o_2}\right)+S_{h_2}\left(J_{h_2}^{(1)}+J_{h_1o_1}^{(2)}S_{o_1}+J_{o_2h_2}^{(2)}S_{o_2}\right)+J_{h_1h_2}^{(2)}S_{h_1}S_{h_2}}\right] = \\ = G^{(0)}+G_{o_1}^{(1)}S_{o_1}+G_{o_2}^{(1)}S_{o_2}+G_{o_1o_2}^{(2)}S_{o_1}S_{o_2} \ , \tag{3.85}$$

we can now reach the same system of equations that we already knew from the high order Decimation method

$$\ln A_{\gamma} = \sum_{n=0,\sigma}^{n=2} G_{\sigma}^{(n)} \prod S_{\rho} \ , \tag{3.86}$$

where the term A_{γ} is used to describe the lhs from Eq. 3.85 and $\left\{G_{\sigma}^{(n)}\right\}$ is the new set of connections that links the remaining units. Again, this equation can be expressed as a system of equations with a Hadamard matrix

$$H_{2^2\times 2^2} = \begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \ , \tag{3.87}$$

notice however that the resulting neural network does only have the two output units, and therefore the Hadamard matrix is $2^2 \times 2^2$.

We now carry out an example with this topology. We shall use the following numerical values for the weights, which have been set up randomly within a $[-1,+1]$ range as

$$\begin{array}{lll} J_{h_1}^{(1)} = 0.90026 \ , & J_{h_1h_2}^{(2)} = -0.087065 \ , & J_{h_1o_1}^{(2)} = -0.53772 \ , \quad J_{h_1o_2}^{(2)} = 0.21369 \ , \\ J_{h_2}^{(1)} = -0.028035 \ , & J_{h_2o_1}^{(2)} = 0.7826 \ , & J_{h_2o_2}^{(2)} = 0.52419 \ , \\ J_{o_1}^{(1)} = 0.78730 \ , & J_{o_1o_2}^{(2)} = -0.88422 \ , & \\ J_{o_2}^{(1)} = -0.29426 \ , & & \end{array}$$

this neural network is being both decimated as per multiple decimation and standard high order Decimation, as shown in Fig. 3.22. Applying multiple decimation leads (directly)

to the following values

$$G^{(1)}_{o_1} = -0.4072 \ ,$$
$$G^{(1)}_{o_2} = 0.1042 \ ,$$
$$G^{(2)}_{o_1 o_2} = 0.2677 \ ,$$

which are added to the original $J^{(1)}_{o_1}$, $J^{(1)}_{o_2}$ and $J^{(2)}_{o_1 o_2}$

$$\tilde{G}^{(1)}_{o_1} = J^{(1)}_{o_1} + G^{(1)}_{o_1} = 0.3801 \ ,$$
$$\tilde{G}^{(1)}_{o_2} = J^{(1)}_{o_1} + G^{(1)}_{o_1} = -0.1900 \ ,$$
$$\tilde{G}^{(2)}_{o_1 o_2} = J^{(2)}_{o_1 o_2} + G^{(2)}_{o_1 o_2} = -0.6165 \ .$$

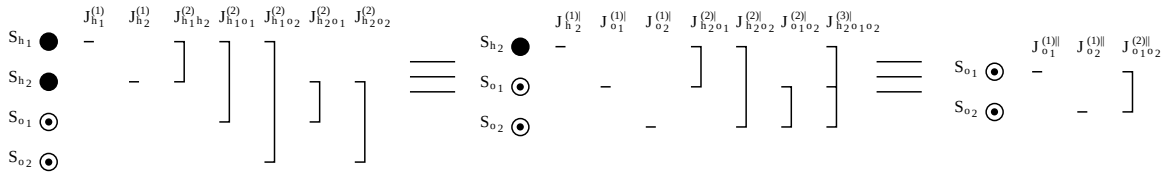

Figure 3.22: High order Decimation process.

On the other hand, we apply HOD to the same randomly initialized weights through this expression

$$\begin{aligned} &\ln\cosh\left(J^{(1)}_{h_1} + J^{(2)}_{h_1 h_2} S_{h_2} + J^{(2)}_{h_1 o_1} S_{o_1} + J^{(2)}_{h_1 o_2} S_{o_2}\right) = \\ &\quad = G^{(0)} + G^{(1)}_{h_2} S_{h_2} + G^{(1)}_{o_1} S_{o_1} + G^{(1)}_{o_2} S_{o_2} + \\ &\qquad + G^{(2)}_{h_2 o_1} S_{h_2} S_{o_1} + G^{(2)}_{h_2 o_2} S_{h_2} S_{o_2} + G^{(2)}_{o_1 o_2} S_{o_1} S_{o_2} + G^{(3)}_{h_2 o_1 o_2} S_{h_2} S_{o_1} S_{o_2} \ , \end{aligned} \tag{3.88}$$

where we first decimate unit S_{h_1}. We reach the following intermediate values

$$G^{(1)}_{h_2} = -0.05301 \ , \quad G^{(2)}_{h_2 o_1} = 0.024002 \ , \quad G^{(2)}_{h_2 o_2} = -0.010013 \ , \quad G^{(3)}_{h_2 o_1 o_2} = -0.0061618 \ ,$$
$$G^{(1)}_{o_1} = -0.3588 \ , \quad G^{(2)}_{o_1 o_2} = -0.058601 \ ,$$
$$G^{(1)}_{o_2} = 0.1314 \ ,$$

which are then added to previously existing ones

$$J^{(n)\prime}_{\sigma} = J^{(n)}_{\sigma} + G^{(n)}_{\sigma} \ , \tag{3.89}$$

thus arriving to

$$J_{h_2}^{(1)'} = -0.081045 \ , \quad J_{h_2o_1}^{(2)'} = 0.8066 \ , \qquad J_{h_2o_2}^{(2)'} = 0.51418 \ , \quad J_{h_2o_1o_2}^{(3)'} = -0.0061618 \ ,$$
$$J_{o_1}^{(1)'} = 0.4285 \ , \qquad J_{o_1o_2}^{(2)'} = -0.94282 \ ,$$
$$J_{o_2}^{(1)'} = -0.16286 \ .$$

We apply again HOD to suppress unit S_{h_2}. The following equation is used

$$\begin{aligned} &\ln\cosh\left(J_{h_2}^{(1)'} + J_{h_2o_1}^{(2)'} S_{o_1} + J_{h_2o_2}^{(2)'} S_{o_2} + J_{h_2o_1o_2}^{(3)'} S_{o_1} S_{o_2}\right) \\ &= G^{(0)''} + G_{o_1}^{(1)''} S_{o_1} + G_{o_2}^{(1)''} S_{o_2} + G_{o_1o_2}^{(2)''} S_{o_1} S_{o_2} \ , \end{aligned} \tag{3.90}$$

thus leading to

$$G_{o_1}^{(1)''} = -0.048408 \ , \quad G_{o_1o_2}^{(2)''} = 0.32634 \ ,$$
$$G_{o_2}^{(1)''} = -0.027151 \ ,$$

these are again added to the original connections

$$J_{\sigma}^{(n)''} = J_{\sigma}^{(n)'} + G_{\sigma}^{(2)''} \ , \tag{3.91}$$

and we obtain the following quantities

$$J_{o_1}^{(1)''} = 0.38009 \ , \qquad J_{o_1o_2}^{(2)''} = -0.61649 \ ,$$
$$J_{o_2}^{(1)''} = -0.19001 \ ,$$

which are exactly the same as per multiple decimation process. Finally, table 3.4 shows a five instances multiple decimation trial: multiple decimation has been run over five different randomly generated sets of weights. The values shown in the table are the initial $J_{\sigma}^{(n)}$, intermediate $J_{\sigma}^{(n)'}$ and final values $J_{\sigma}^{(n)''}$. These last ones can be compared to the ones obtained by the multiple decimation process $\tilde{G}_{\sigma}^{(n)}$, notice however that both process provide the same results.

3.5.3 Multiple unit decimation for a 10 units BM

Finally, we propose a numerical example that is used to compare the Multiple Decimation process with the high order Decimation method. We will decimate a second order standard

Trial number	1	2	3	4	5
$J_{h_1}^{(1)}$	0.90026	0.23086	-0.88422	-0.96945	0.67624
$J_{h_1h_2}^{(2)}$	-0.087065	0.87094	-0.60256	0.69244	0.41894
$J_{h_1o_1}^{(2)}$	-0.53772	0.58387	-0.29426	0.49357	-0.96072
$J_{h_1o_2}^{(2)}$	0.21369	0.84363	0.62633	-0.10981	0.36255
$J_{h_2}^{(1)}$	-0.028035	0.47641	-0.98028	0.86363	-0.24104
$J_{h_2o_1}^{(2)}$	0.7826	-0.64747	-0.72222	-0.068011	0.66359
$J_{h_2o_2}^{(2)}$	0.52419	-0.18859	-0.59447	-0.1627	0.0056258
$J_{o_1}^{(1)}$	-0.96299	0.83381	0.20758	0.050305	-0.14222
$J_{o_1o_2}^{(2)}$	-0.11059	0.7873	-0.60237	0.34427	-0.62069
$J_{o_2}^{(1)}$	0.64281	-0.17946	-0.45562	-0.59471	-0.39077
$J_{h_2}^{(1)'}$	-0.081045	0.58175	-0.6532	0.42829	-0.10205
$J_{h_2o_1}^{(2)'}$	0.8066	-0.37293	0.35786	0.097147	0.42525
$J_{h_2o_2}^{(2)'}$	0.51418	0.26143	-0.79883	-0.20011	0.083138
$J_{h_2o_1o_2}^{(3)'}$	-0.0061618	-0.04987	-0.62887	-0.019423	0.052702
$J_{o_1}^{(1)'}$	-1.3218	0.90365	-0.7926	-0.23783	-0.55941
$J_{o_1o_2}^{(2)'}$	-0.16919	1.048	-0.69926	0.31658	-0.82509
$J_{o_2}^{(1)'}$	0.77421	-0.078672	0.055192	-0.53351	-0.2701
$J_{o_1}^{(1)''}$	-1.3702	0.71426	0.56755	-0.19677	-0.59627
$J_{o_1o_2}^{(2)''}$	0.15714	0.056605	-0.48987	0.29292	-0.79664
$J_{o_2}^{(1)''}$	0.74706	0.95569	-0.40708	-0.61435	-0.25636
$\tilde{G}_{o_1}^{(1)}$	-1.3702	0.71426	0.56755	-0.19677	-0.59627
$\tilde{G}_{o_1o_2}^{(2)}$	0.15714	0.056605	-0.48987	0.29292	-0.79664
$\tilde{G}_{o_2}^{(1)}$	0.74706	0.95569	-0.40708	-0.61435	-0.25636

Table 3.4: Multiple vs. standard decimation trial example.

topology with ten units in order to compare the correlations that should be used for all the weights. Notice that we are only working with a part of a whole learning process, which would be a free correlation calculus for a set of ten units. In this structure, there are 45 second order weights and 10 bias terms, since units are not connected to themselves

$$\mathcal{W} = \begin{pmatrix} 0.0 & w^{(2)}_{1,2} & w^{(2)}_{1,3} & \cdots & w^{(2)}_{1,9} & w^{(2)}_{1,10} \\ w^{(2)}_{1,2} & 0.0 & w^{(2)}_{1,3} & \cdots & w^{(2)}_{2,9} & w^{(2)}_{2,10} \\ w^{(2)}_{1,2} & w^{(2)}_{1,3} & 0.0 & \cdots & w^{(2)}_{3,9} & w^{(2)}_{3,10} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ w^{(2)}_{1,10} & w^{(2)}_{2,10} & w^{(2)}_{3,10} & \cdots & 0.0 & w^{(2)}_{1,9} \\ w^{(2)}_{1,10} & w^{(2)}_{2,10} & w^{(2)}_{3,10} & \cdots & w^{(2)}_{1,9} & 0.0 \\ w^{(1)}_{1} & w^{(1)}_{2} & w^{(1)}_{3} & \cdots & w^{(1)}_{9} & w^{(1)}_{10} \end{pmatrix} . \tag{3.92}$$

We now compute all the expectation values and correlations that are involved in the process for a 1000 randomly generated set of $\mathcal{W}$ instances, decimated by HOD and Multiple Decimation. These will be referred as $\langle \prod S_\rho \rangle_{HOD}$ and $\langle \prod S_\rho \rangle_{MDec}$, respectively. We have calculated the following statistics to measure how the correlations found by HOD differ from the ones computed by multiple decimation:

- Mean absolute difference μ between the two results, defined as

$$\mu = \frac{1}{100 \cdot 1000} \sum \left| \left\langle \prod S_\rho \right\rangle_{HOD} - \left\langle \prod S_\rho \right\rangle_{MDec} \right| = 0.000854 \ . \tag{3.93}$$

- Standard deviation σ associated to this mean value

$$\sigma = \frac{1}{100 \cdot 1000} \left(\sum \left(\left| \left\langle \prod S_\rho \right\rangle_{HOD} - \left\langle \prod S_\rho \right\rangle_{MDec} \right| - \mu \right)^2 \right)^{\frac{1}{2}} = 0.0064247 \ . \tag{3.94}$$

- Maximum and minimum values associated to the same operation

$$max_S = max \left\{ \left| \left\langle \prod S_\rho \right\rangle_{HOD} - \left\langle \prod S_\rho \right\rangle_{MDec} \right| \right\} = 0.30578 \ , \tag{3.95}$$

$$min_S = min \left\{ \left| \left\langle \prod S_\rho \right\rangle_{HOD} - \left\langle \prod S_\rho \right\rangle_{MDec} \right| \right\} = 0.0 \ . \tag{3.96}$$

By using statistical theory, it can be shown that about 99.73% of the values will have an error smaller than $3\sigma = 0.0193$, and that in the 99.994% of the cases it will be $4\sigma = 0.0257$. Notice however that this example is working with a case that involves solving the HOD equations for 10 units, hence the algorithm has to solve a total of $1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 = 2044$ equations. Notice also that the previous example resulted in more similar values, as far as decimating a four units neural network does not imply as many calculus as the ten units neural network decimation example.

3.6 Simulations and results applying HOD

In this section we solve some problems through the HOD method to show the effectiveness of the algorithm. In the first part of this section we describe a problem that was created as a *toy* problem: toy problems are non-real in the sense that they are used to test if a given algorithm could be used on a real life, commercial application. The configuration of the neural network, its learning parameters and a comparison with the multilayer perceptron are then described, as the results are discussed. The second part of this section proceeds with the BM applied to some problems that were once solved as real problems, and that now stand in a benchmarking repository [Newman et al., 1998, Prechelt, 1994]. We finally conclude this section by solving a toy problem that was specifically created as a benchmark for multiple learning algorithms: the Monk problem [Thrun et al., 1991].

However, we will first describe a modification to the standard gradient descent learning process: the BM learning problem is solved by performing gradient descent over the Kullback-Leibler distance, thus resulting in the following expression

$$\Delta w_{\sigma}^{(n)} = \frac{\eta}{T}\left(\left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle^{*} - \left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle\right) \ ,$$

where the values of the correlations are computed by using either the MC based algorithm or either the high order Decimation method. However, the learning process is better carried out when a variation of the gradient descent algorithm, which is known as *conjugate*

gradient [Duda et al., 2001], is applied to this expression. This new algorithm changes the previous equation into

$$\Delta w_{\sigma}^{(n)}\big|_k = (1-\alpha)\left[\frac{\eta}{T}\left(\left\langle\prod_{\rho\in\sigma} S_\rho\right\rangle^* - \left\langle\prod_{\rho\in\sigma} S_\rho\right\rangle\right)\right] + \alpha\,\Delta w_{\sigma}^{(n)}\big|_{k-1} \quad , \tag{3.97}$$

where k is the current algorithm iteration and α is a new parameter that has to be properly tuned to reach convergence and that can not be greater than 1; notice that the increment of the weights at the previous iteration is being used to update their current value.

3.6.1 The letter recognition problem: a toy problem

The high order decimation method has been tested against a perceptron and a traditional BM in a letter recognition dataset context, where the network has to recognize characters from a noisy source. A system of 24 letters written with a *Times New Roman* font is used, and each letter is represented by a 50x50 pixels binary image. While these neat characters are the ones to be learned, a set of 100 different images for each letter is generated by adding random noise which is implemented via bit negation. The amount of noise present is characterized by a parameter γ, which is proportional to the percentage of negated bits. Thus for example, 10% of bits are reversed when $\gamma = 10$. Once the learning using the previous patterns has been carried out, a new set of noisy characters is generated using the same procedure to test the network.

Table 3.5 shows the amount of time and epochs (equal to the number of times weights are updated in a complete run of the learning algorithm) required by a Boltzmann Machine trained with the high order Decimation method compared with results for the same network trained with the standard Monte Carlo algorithm using the above stated patterns for $\gamma = 20$. All calculations have been performed on a DELL workstation mounting a Pentium Xeon EMT64 with 2Mb of cache processor working at 3.0 GHz and equipped with 1.0 Gb DDR2 ECC RAM memory. As it can be seen, not only the high order Decimation performs *faster* but also requires less epochs to reach the desired result. This is due to the fact that every Monte Carlo simulation has an associated statistical error,

Algorithm	Mean epochs	Mean time/epoch (seconds)
Monte Carlo	45	586.68
high order Decimation	11	4.39

Table 3.5: Decimation method against Monte Carlo implementation.

and bringing that below a certain limit (imposed by the accuracy to be achieved) can be very expensive in computational terms. The network used in this calculation has 2500 input units (corresponding to the 50x50 pixel images used as input), 1 hidden and 5 output units. The training parameters were $\eta = 0.2$, $\alpha = 0.1$, maximum absolute error $\mid \partial\mathcal{E}/\partial w \mid = 0.05$ and maximum absolute initial random value for the weights $\mid w_0 \mid = 1.0$.

The relation *time/epoch* describes how long does it take to run a complete epoch in the simulation. As it can be seen from the table, Decimation performs considerably better in both aspects. A Decimation epoch is faster because there is no need to run a Simulated Annealing but only to solve a system of equations. On the other hand, it needs less epochs to end because it does not suffer from statistical errors as does a Monte Carlo simulation.

Finally, a comparison between the performance of the BM trained with the high order Decimation method and a dual layer perceptron is presented. The comparison is made on the basis that both networks can provide a *full* solution to the problem at hand if enough learning instances are allowed. In fact both networks have been trained many times and its efficiency tested at the end of each learning process, finding that both systems can be 100% efficient in many cases. Taking into account this fact, the mean efficiency over a batch of instances of the same problem has been measured, and this parameter used to decide which network performs better in a statistical sense.

The BM used in the comparison is fully connected, with five output units and a variable number of hidden neurons ranging from zero to two. Learning parameters are once again $\eta = 0.2$, $\alpha = 0.1$, maximum absolute error $\mid \partial\mathcal{E}/\partial w \mid = 0.05$ and maximum absolute initial random value for weights $\mid w_0 \mid = 1.0$. On the other hand, the topology of the perceptron employed has been optimized to get best results. The experiment has

γ	Mean eff. for BM	Mean eff. for perceptron
0.10	97.25	60.38
0.15	96.87	68.77
0.20	94.58	83.49
0.25	94.05	88.55
0.30	86.90	84.87

Table 3.6: Decimation method against perceptron.

been repeated using a number of hidden units spanning the range from 5 to 2500, using both lineal and hyperbolic tangent transfer functions, and a momentum α between 0.0 and 0.2 with an adaptive η learning rate. Results on the performance are presented in Table 3.6.

It can be seen from the table that the Boltzmann Machine performs slightly *better* than the perceptron. This can be understood when the problem is carefully analyzed, as it has an original discrete nature. Since the Boltzmann Machine is a binary neural network and the perceptron is a continuous one, the BM is better suited to solve the problem. However, the perceptron is a widely used multi purpose network, and it can perform very well on problems where other continuous models fail. In any case and although the Boltzmann Machine does a better job, the perceptron still provides solutions that are more than satisfactory. Still when high order Decimation is employed, the Boltzmann Machine not only outperforms the perceptron but also gets the solution in a similar period of time.

3.6.2 Problems from a benchmarking repository

The efficiency of the HOD method has been tested against three classification problems drawn from the *UCI* [Newman et al., 1998] and *Proben1* [Prechelt, 1994] repositories. The following tasks were selected in order to establish a comparison between the performance of the BM and the Perceptron:

- *Balance* problem. This dataset belongs to the *UCI* repository and was generated to model psychological experimental results [Klahr and Siegler, 1978]. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (left-distance $\times$ left-weight) and (right-distance $\times$ right-weight). If they are equal, it is balanced.

- *Tic-tac-toe* problem. This database has been extracted from the *UCI* repository. It encodes the complete set of possible board configurations at the end of tic-tac-toe games, and the target concept is *win for x* (hence, it is true when x has one of the 8 possible ways to create a *three-in-a-row*), where x is assumed to have played first. This dataset was first used in [Matheus and Rendell, 1989].

- *Gene* problem. The *Primate splice-junction gene sequences* problem, which will be referred to as *Gene* problem, comes from the *Proben1* database and was first used in [Noordewier et al., 1991]. Splice junctions are points on a DNA sequence where *superfluous* DNA is removed during the process of protein creation. The problem posed in this dataset is to recognize, given a DNA sequence, the boundaries between exons (the parts that must be retained after splicing), introns (the parts that must be spliced out) and the ones that are neither exons nor introns (that is, parts that can be kept or not without an apparent impact on the result).

All three problems were originally conceived as classification tasks with a discrete set of inputs. Since the Boltzmann Machine is a discrete neural network, they are presumably well suited for it. The comparison has been carried out using a standard *ten-fold cross validation* method [Stone, 1977], where the data is divided in ten different random, uniformly distributed, test sets. These sets are combined to generate ten separate training patterns, and the final efficiency is calculated as the mean efficiency on solving each pattern separately. The training parameters and topologies for both the Perceptron and

the Boltzmann Machine were systematically tuned until the best possible results were achieved. These are summarized in Table 3.7.

Problem	Perceptron efficiency	BM efficiency
Balance	93.90	96.49
Tic-tac-toe	100.00	98.44
Gene	96.77	97.80

Table 3.7: Decimation method versus perceptron.

The convergence time for both algorithms is similar, taking only a few seconds to finish on a 2.6 GHz standard Pentium IV platform. Both methods solve the problems efficiently: the perceptron outperforms the Boltzmann Machine on the *Tic-tac-toe*, while the BM wins on the other two tasks. In any case, both algorithms perform remarkably well when dealing with any of these problems.

3.6.3 The Monk Problem

The Monk problem [Thrun et al., 1991] was originally proposed as a benchmarking comparative between many data classification methods. This problem was given to several different research groups who where either creators or experts in the use of these algorithms. Back in 1992, M. Graña et al. solved this problem with a good overall efficiency by using a HOBM [Graña et al., 1997], which indicates that it can be a good starting point to test the high order Decimation method presented in this work.

The Monk Problem is characterized by a space $\mathcal{M}$ containing 432 different vectors used to compute three different tasks, referred to as M_1, M_2 and M_3, respectively. Each input vector has six discrete variables x_0 to x_5, which can only take integer values in the ranges $x_0 \in [1,2,3], x_1 \in [1,2,3], x_2 \in [1,2], x_3 \in [1,2,3], x_4 \in [1,2,3,4]$ and $x_5 \in [1,2]$.

Every task in the Monk Problem is evaluated independently of the other two. They are described in terms of the following boolean logical functions:

- M_1 classifies *true* according to the logical operation $(x_0 = x_1) + (x_4 = 1)$. For this

task, 124 specific vectors were selected as the training set and all 432 were used for testing purposes. The 124 training vectors were randomly selected from the whole space by the authors of the problem.

- M_2 classifies *true* if *exactly* two of the six inputs are set to 1. As before, 169 vectors specified by the authors are used for training and the whole space $\mathcal{M}$ for testing.

- M_3 classifies *true* if $[(x_4 = 3) \cdot (x_3 = 1)] + [(x_4 \neq 4) \cdot (x_1 \neq 3)]$. In this case, 122 vectors were randomly selected as the training set. However, 5% of them were misclassified, in an attempt to simulate the effects induced by noise on the patterns.

The results obtained by the Boltzmann Machine with the high order Decimation method are compared against other classification techniques in Table 3.9, and are expressed as the percentage of correctly classified test vectors for each task. These values have been obtained using the same learning and test vectors employed in Refs. [Graña et al., 1997] and [Thrun et al., 1991]. As it can be seen from the table, the results obtained with the decimated Boltzmann Machine are good when compared to the other methods.

Task	# hidden units	η	α
M_1	4	0.8	0.2
M_2	3	0.4	0.4
M_3	4	0.8	0.2

Table 3.8: BM topology and learning parameters.

The learning parameters η and α from Eq. 3.97 and the topology employed are reported in Table 3.8. Notice that the table only shows the number of hidden units, as in all cases a total of 10 input and 1 output units were used, and the networks employed were fully connected.

The weights were initialized at random in the range $[-1, +1]$ and the learning algorithm was considered to have finished when $|\varepsilon_{err}| \leq 0.02$, as we found this value to be a suitable bound to achieve the accuracy reported on Table 3.9.

Method	M_1	M_2	M_3
AQ17-DCI	100	100	94.2
AQ17-HCI	100	93.1	100
AQ17-FCLS	-	92.6	97.2
AQ14-NT	-	-	100
AQ15-GA	100	86.8	100
Assistant professional	100	81.3	100
mFOIL	100	69.2	100
ID5R	81.7	69.2	95.2
IDL	97.2	66.2	-
ID5R-hat	90.3	65.7	-
TDIDT	75.7	66.7	-
ID3	98.6	67.9	94.4
ID3, no windowing	83.2	69.1	95.6
AQR	95.9	79.7	87.0
CN2	100	69.0	89.1
CLASSWEB 0.10	71.8	64.8	80.8
CLASSWEB 0.15	65.7	61.6	85.4
CLASSWEB 0.20	63.0	57.2	75.2
PRISM	86.3	72.7	90.3
ECOBWEB leaf prediction	71.8	67.4	68.2
ECOBWEB l.p. & information utility	82.7	71.3	68.0
Backpropagation	100	100	93.1
Backpropagation with weight decay	100	100	97.2
Cascade correlation	100	100	97.2
Monte Carlo HOBM	100	98.8	97.0
Decimated Boltzmann Machine	**100**	**100**	**98.2**

Table 3.9: Efficiency on solving the Monk's problem.

As in Ref. [Farguell et al., 2006], we compare both the execution time and the convergence speed of the high order Decimation method to our implementation of the standard BM based on MC dynamics. Furthermore, the last temperature used in the cooling schedule was set at $T \simeq 0.3$ and the number of samples used to evaluate correlations in the MC algorithm was 1000 times the sum of the number of hidden and output units. This implementation produced an average error $|\varepsilon_{err}| \leq 0.02$, which can be identified with the standard deviation of the simulation. Results for both algorithms are given in Table 3.10. The information reported is the time (in seconds) needed to evaluate a full set of correlations (which is referred to as an epoch) in a weight update iteration, and the mean epochs needed to reach convergence. The values reported in Table 3.10 were obtained using a personal computer equipped with a 2.6 GHz Pentium IV processor and 512 Mb of RAM, working at 533 MHz.

Task	HOD $\mathrm{T_e}$	HOD $\langle \mathrm{e} \rangle$	MC $\mathrm{T_e}$	MC $\langle \mathrm{e} \rangle$
M_1	0.025	67	0.49	108
M_2	0.010	12	0.53	31
M_3	0.025	20	0.48	23

Table 3.10: High order Decimation algorithm convergence times, in seconds. $\mathrm{T_e}$ and $\langle \mathrm{e} \rangle$ stand for time per epoch and mean number of epochs, respectively.

These results indicate that the Decimation algorithm is *faster* than the standard implementation of the Boltzmann Machine, both in the time needed to compute an epoch and in the number of iterations required for the algorithm to converge. Moreover, the HOD method allows for a HOBM implementation with no hidden units: the algorithm itself can be used to decimate any kind of connection and the high order weights supply enough degrees of freedom to make this process feasible. The high order Decimation equations state that the information stored by the hidden units can be introduced in the high order weights instead.

Chapter 4

BM learning through Hadamard matrices

4.1 Introduction

In this chapter, we show that the Boltzmann probability distribution reproduced by a HOBM model can be described in terms of Hadamard matrices and a set of high order weights. The multiple decimation process can then be used to show that a standard BM with hidden units is equivalent to a smaller (with less units) HOBM with no hidden units, at the price of having to deal with high order weights.

This chapter is distributed as follows: the effect that a set of weights connecting only the input units has on the behavior of the neural network is discussed in section 4.2. In section 4.3, we discuss a *forward* problem, which is simply to find the resulting probability distribution of a HOBM when the complete set of connections linking all the units in the network is known. In section 4.4, we present the *backwards* problem, which can be considered as a *learning process* for a HOBM: given a probability distribution that characterizes a known problem we want the neural network to learn it. This chapter concludes with the application of a particular solution to the backwards problem.

4.2 Reduction of connections between input units on a HOBM

In this section, we consider a Boltzmann Machine (a standard BM or a HOBM) with n_i input units, n_h hidden units and n_o output units for a total of $N = n_i + n_h + n_o$ neurons. We assume that input units are connected among themselves, and therefore we have $\begin{pmatrix} N \\ 1 \end{pmatrix}$ bias terms, $\begin{pmatrix} N \\ 2 \end{pmatrix}$ second order terms, $\begin{pmatrix} N \\ 3 \end{pmatrix}$ third order terms, and so on, until the last, single N-th order weight connecting all units; yielding a total of $2^N - 1$ connections. We now recall the Boltzmann probability distribution and the definition of conditional probability to write down the equation

$$\begin{aligned} p(\alpha, \beta|\gamma) &= \frac{p(\alpha, \beta, \gamma)}{p(\gamma)} \\ &= \frac{\mathrm{e}^{-\mathcal{E}_{\alpha,\beta,\gamma}(S_o,S_h,S_i)}}{\sum_{\mu,\nu} \mathrm{e}^{-\mathcal{E}_{\mu,\nu,\gamma}(S_o,S_h,S_i)}} \ , \end{aligned} \tag{4.1}$$

corresponding to the probability of finding the output units S_o in the state α, the hidden units S_h in a state β and the input units S_i clamped to a state γ. In these expressions, the temperature normalized energy functional reads

$$\begin{aligned} \mathcal{E}_{\alpha,\beta,\gamma}(S_o, S_h, S_i) &= -\sum_{i_1 \in n_i} J^{(1)}_{i_1} S_{i_1} - \sum_{i_1 < i_2 \in n_i} J^{(2)}_{i_1 i_2} S_{i_1} S_{i_2} - \ldots - J^{(n_i)}_{123\ldots n_i} S_1 S_2 S_3 \ldots S_{n_i} + \\ &\quad + \tilde{\mathcal{E}}_{\alpha,\beta,\gamma}(S_o, S_h, S_i) \\ &= \mathcal{E}_\gamma(S_i) + \tilde{\mathcal{E}}_{\alpha,\beta,\gamma}(S_o, S_h, S_i) \ , \end{aligned} \tag{4.2}$$

where the $\mathcal{E}_\gamma(S_i)$ term in the first line connects only input units, while $\tilde{\mathcal{E}}_{\alpha,\beta,\gamma}(S_o, S_h, S_i)$ stands for the sum of all the other terms contributing to the energy. An example of this separation can be seen in Fig. 4.1, where a network with input units (empty circles), no hidden units and a single output unit, is shown. The terms contributing to $\mathcal{E}_\gamma(S_i)$ are marked with an arrow in Fig. 4.1a and are not present in Fig. 4.1b.

Since for a clamped input $\mathcal{E}_\gamma(S_i)$ is a constant, one can simplify it in the numerator

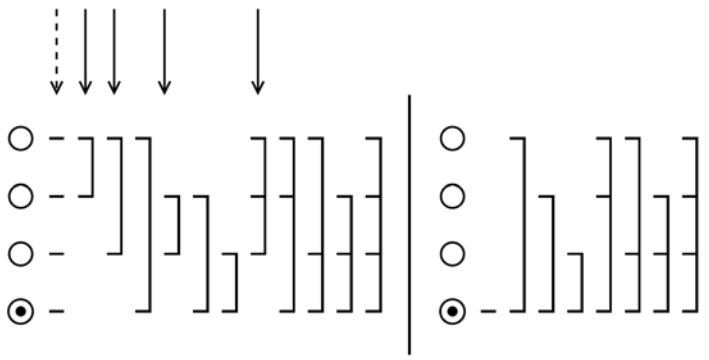

Figure 4.1: Scheme of a simple Boltzmann Machine, with the different terms contributing to $\mathcal{E}_{\alpha,\beta,\gamma}(S_o, S_h, S_i)$ (a) and $\tilde{\mathcal{E}}_{\alpha,\beta,\gamma}(S_o, S_h, S_i)$ (b). In (a) all terms are shown, with arrows pointing to those that contribute to $\mathcal{E}_\gamma(S_i)$. In (b), only the terms contributing to $\tilde{\mathcal{E}}_{\alpha,\beta,\gamma}(S_o, S_h, S_i)$ are depicted. Notice that the dashed arrow in (a) indicates that only the bias terms connecting input units belong to $\mathcal{E}_\gamma(S_i)$, and this is why a remaining bias term appears in the output unit in (b).

and denominator of Eq. 4.1 to find

$$p(\alpha, \beta \mid \gamma) = \frac{\mathrm{e}^{-\mathcal{E}_{\alpha,\beta,\gamma}(S_o,S_h,S_i)}}{\sum_{\mu,\nu} \mathrm{e}^{-\mathcal{E}_{\mu,\nu,\gamma}(S_o,S_h,S_i)}} \equiv \frac{\mathrm{e}^{-\tilde{\mathcal{E}}_{\alpha,\beta,\gamma}(S_o,S_h,S_i)}}{\sum_{o,h} \mathrm{e}^{-\tilde{\mathcal{E}}_{\mu,\nu,\gamma}(S_o,S_h,S_i)}} \quad . \tag{4.3}$$

Of course, the term $\mathcal{E}_\gamma(S_i)$ will change when the input pattern is changed, but it will anyway cancel in Eq. 4.3, irrespective of the value of $\{S_i\}$. This leads to the conclusion that for any given HOBM, weights linking only input units do not affect the output conditional probabilities corresponding to the situation where the input units are clamped to any input state γ.

4.3 The forward problem

Consider now an N-th order Boltzmann Machine with n_i inputs, n_o outputs and no hidden units, for a total of $N = n_i + n_o$ neurons. We assume here that this neural network has a known set of connections $\left\{J_\sigma^{(n)}\right\}$, with n the order of the weight and σ the set of labels denoting the units being connected. In the *forward* problem one seeks to find the

probability distribution associated to these weights assuming once again that input units can be connected among themselves. The forward problem is *trivial* since we know that the resulting probability distribution is of the Boltzmann type and a full set of weights determines it uniquely

$$p\left(\alpha,\gamma\right)=\frac{\mathrm{e}^{-\mathcal{E}_{\alpha,\gamma}}}{\mathcal{Z}}\ ,$$

$\mathcal{Z}$ being the partition function,

$$\mathcal{Z}=\sum_{\alpha,\gamma}\mathrm{e}^{-\mathcal{E}_{\alpha,\gamma}}\ .$$

4.4 The backwards problem

The backwards problem is exactly the opposite of the forward problem presented above. It is much closer to a real learning problem in neural network theory. The goal of the backwards problem is to find a set of weights that reproduce a given probability distribution.

The first problem that we analyze is how to find these weights (thus solving the learning problem) when we know the complete probability distribution of all the states of the neural network, we then discuss a numerical example to illustrate how the backwards problem is solved for a HOBM. We proceed with the extension of the backwards problem to the case where only input-output conditioned probabilities are known. A solution to this problem is presented in the next subsection, thus proving that this learning problem can be solved analytically. The section is concluded with the discussion of the situation where one does not know the complete probability distribution for all the states, which is the instance of a real learning problem where the neural network has to extrapolate the unknown probabilities.

4.4.1 The backwards problem for a known p.d.f.

In this section, we present an analytical solution to the learning problem of the high order Boltzmann Machine in the particular case where there are no hidden units and the complete probability distribution of all states is known. In this sense, the probability distribution for all possible states is directly shown to the neural network.

We start with the analysis of an N-th order Boltzmann Machine with n_i input units, n_o output units and no hidden units, for a total of $N = n_i + n_o$ neurons. The value of the weights is not known, though the neural network may have all possible connections up to order N. In the backwards problem it is assumed that the complete probability distribution associated to the BM, $p(\alpha, \gamma)$, is known for every input state γ and every output state α. We now look for a set of weights that reproduce this probability distribution, which shall be of the Boltzmann form

$$p(\alpha, \gamma) = \frac{\mathrm{e}^{-\mathcal{E}_{\alpha,\gamma}}}{\mathcal{Z}} \ .$$

Applying logarithms on both sides one finds

$$\ln p(\alpha, \gamma) = -\ln \mathcal{Z} + \tag{4.4}$$
$$+ \sum_{i_1} J^{(1)}_{i_1} S_{i_1} + \sum_{i_1<i_2} J^{(2)}_{i_1 i_2} S_{i_1} S_{i_2} + \sum_{i_1<i_2<i_3} J^{(3)}_{i_1 i_2 i_3} S_{i_1} S_{i_2} S_{i_3} + \ldots + J^{(N)}_{12\ldots N} S_1 S_2 \ldots S_N \ .$$

Of course, there are 2^N equations of this form corresponding to the 2^N different states the units in the network can take. The set of $\begin{pmatrix} N \\ 1 \end{pmatrix}$ bias terms, $\begin{pmatrix} N \\ 2 \end{pmatrix}$ two-body weights, $\begin{pmatrix} N \\ 3 \end{pmatrix}$ three-body weights, and up to $\begin{pmatrix} N \\ N \end{pmatrix}$=1 N-body weights form a total of $\sum_{i=1}^{N} \begin{pmatrix} N \\ i \end{pmatrix} = 2^N - 1$ unknown coefficients $\left\{ J^{(n)}_{\sigma} \right\}$. The relation given in Eq. 4.4 produces 2^N different equations when all possible values of the input and output units are used. In this way, Eq. 4.4 yields 2^N equations for the $2^N - 1$ weights $\left\{ J^{(n)}_{\sigma} \right\}$. One might think that the partition function $\mathcal{Z}$ can not be used as an independent variable, since it

is fixed by the $\left\{J_\sigma^{(n)}\right\}$ weights through its definition

$$\mathcal{Z} = \sum_{\nu,\mu} \mathrm{e}^{-\mathcal{E}_{\nu,\mu}} \ .$$

However, $\mathcal{Z}$ is just a real number and so an arbitrary energy shift $\mathcal{E}_0 = -J^{(0)} + \ln \mathcal{Z}$ can always be added to the energy functional without changing the probability distribution. This can be trivially seen by multiplying the numerator and denominator of the terms entering in the probabilities $p\left(\alpha,\gamma\right)$ by $\mathrm{e}^{J^{(0)}}$

$$p\left(\alpha,\gamma\right) = \frac{\mathrm{e}^{J^{(0)}}\mathrm{e}^{-\mathcal{E}_{\alpha,\gamma}}}{\mathrm{e}^{J^{(0)}}\sum_{\nu,\mu}\mathrm{e}^{-\mathcal{E}_{\nu,\mu}}} = \frac{\mathrm{e}^{J^{(0)}-\mathcal{E}_{\alpha,\gamma}}}{\sum_{\nu,\mu}\mathrm{e}^{J^{(0)}-\mathcal{E}_{\nu,\mu}}} = \frac{\mathrm{e}^{-\tilde{\mathcal{E}}_{\alpha,\gamma}}}{\sum_{\nu,\mu}\mathrm{e}^{-\tilde{\mathcal{E}}_{\nu,\mu}}} \ , \tag{4.5}$$

when $\tilde{\mathcal{E}}_{\alpha,\gamma}$ is the new shifted energy functional. We see then that one additional unknown weight $J^{(0)}$ can always be introduced without affecting the probability distributions, leading to a system of 2^N equations for 2^N unknown variables. In this way, one obtains a set of linear equations for the 2^N quantities $\left\{J^{(0)}, J_\sigma^{(n)}\right\}$, which will be in the following simply denoted by $\left\{J_\sigma^{(n)}\right\}$, thus implicitly understanding that a $J^{(0)}$ term has already been added. We then write the general equation for the backwards problem in the form

$$\begin{aligned} \ln\left[\vec{p}\left(\alpha,\gamma\right)\right] = J^{(0)} + \\ + \sum_{i_1} J_{i_1}^{(1)} S_{i_1} + \sum_{i_1<i_2} J_{i_1 i_2}^{(2)} S_{i_1} S_{i_2} + \sum_{i_1<i_2<i_3} J_{i_1 i_2 i_3}^{(3)} S_{i_1} S_{i_2} S_{i_3} + \ldots + J_{12\ldots N}^{(N)} S_1 S_2 \ldots S_N \ , \end{aligned} \tag{4.6}$$

where $\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]$ stands for a vector that contains the probability distribution that the

system is going to learn and that is completely known

$$\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=\left(\begin{array}{c}\ln p\left(\alpha_1,\gamma_1\right)\\ \ln p\left(\alpha_1,\gamma_2\right)\\ \vdots\\ \ln p\left(\alpha_1,\gamma_{2^{n_i}}\right)\\ \ln p\left(\alpha_2,\gamma_1\right)\\ \ln p\left(\alpha_2,\gamma_2\right)\\ \vdots\\ \ln p\left(\alpha_2,\gamma_{2^{n_i}}\right)\\ \vdots\\ \ln p\left(\alpha_{2^{n_o}},\gamma_1\right)\\ \ln p\left(\alpha_{2^{n_o}},\gamma_2\right)\\ \vdots\\ \ln p\left(\alpha_{2^{n_o}},\gamma_{2^{n_i}}\right)\end{array}\right). \tag{4.7}$$

This vector is ordered according to both the values of the input and output units. We now describe the order that the input units follow according to the input set γ as

$$\begin{aligned}\gamma_1&=\left\{S_{i_1}=-1,S_{i_2}=-1,\ldots,S_{i_{n_i}-1}=-1,S_{i_{n_i}}=-1\right\}\ ,\\ \gamma_2&=\left\{S_{i_1}=-1,S_{i_2}=-1,\ldots,S_{i_{n_i}-1}=-1,S_{i_{n_i}}=1\right\}\ ,\\ \gamma_3&=\left\{S_{i_1}=-1,S_{i_2}=-1,\ldots,S_{i_{n_i}-1}=1,S_{i_{n_i}}=-1\right\}\ ,\\ &\ldots\\ \gamma_{2^{n_i}}&=\left\{S_{i_1}=1,S_{i_2}=1,\ldots,S_{i_{n_i}-1}=1,S_{i_{n_i}}=1\right\}\ .\end{aligned} \tag{4.8}$$

Notice that this ordering corresponds to a binary counting sequence for all the input units;

the output set α is built by using the same concept with the output units

$$\begin{aligned}
\alpha_1 &= \left\{S_{o_1} = -1, S_{o_2} = -1, \ldots, S_{o_{n_o}-1} = -1, S_{o_{n_o}} = -1\right\} \ , \\
\alpha_2 &= \left\{S_{o_1} = -1, S_{o_2} = -1, \ldots, S_{o_{n_o}-1} = -1, S_{o_{n_o}} = 1\right\} \ , \\
\alpha_3 &= \left\{S_{o_1} = -1, S_{o_2} = -1, \ldots, S_{o_{n_i}-1} = 1, S_{o_{n_o}} = -1\right\} \ , \\
&\ldots \\
\alpha_{2^{n_o}} &= \left\{S_{o_1} = 1, S_{o_2} = 1, \ldots, S_{o_{n_o}-1} = 1, S_{o_{n_o}} = 1\right\} \ . \qquad (4.9)
\end{aligned}$$

The order in which these elements appear in $\ln\left[\vec{p}\,(\alpha,\gamma)\right]$ is shown in table 4.1.

Position	S_{i_1}	S_{i_2}	...	$S_{i_{n_i}}$	S_{o_1}	S_{o_2}	...	$S_{o_{n_o-1}}$	$S_{o_{n_o}}$	$\ln\left[\vec{p}\,(\alpha,\gamma)\right]$
1	-1	-1	...	-1	-1	-1	...	-1	-1	$\ln p\,(\alpha_1,\gamma_1)$
2	-1	-1	...	-1	-1	-1	...	-1	1	$\ln p\,(\alpha_2,\gamma_1)$
3	-1	-1	...	-1	-1	-1	...	1	-1	$\ln p\,(\alpha_3,\gamma_1)$
4	-1	-1	...	-1	-1	-1	...	1	1	$\ln p\,(\alpha_4,\gamma_1)$
...	...	...	...	...	...	...	...	...	...	...
2^N-1	1	1	...	1	1	1	...	1	-1	$\ln p\,(\alpha_{2^{n_o}-1},\gamma_{2^{n_i}})$
2^N	1	1	...	1	1	1	...	1	1	$\ln p\,(\alpha_{2^{n_o}},\gamma_{2^{n_i}})$

Table 4.1: Binary counting used used to order the probability distribution to learn.

This expression from Eq. 4.6 produces 2^N different equations corresponding to the 2^N available states of the network that can be built from the binary values $[+1,-1]$ that units S_1 to S_N can take. Consequently, this expression provides a set of 2^N linear equations for the 2^N unknown quantities $\left\{J_\sigma^{(n)}\right\}$. This set of equations has always a non-zero solution that is also unique, as the system of equations is structured on a Hadamard $2^N \times 2^N$ matrix $H_{2^N\times 2^N}$ ([Sylvester, 1867], see also the appendix)

$$\ln\left[\vec{p}\,(\alpha,\gamma)\right] = H_{2^N\times 2^N}\cdot\vec{J} \ , \qquad (4.10)$$

where $\vec{J}$ is a vector of 2^N components

$$\vec{J} = \left(J^{(0)}, J_1^{(1)}, J_2^{(1)}, \ldots, J_{12}^{(2)}, J_{13}^{(2)}, \ldots, J_{12\ldots N}^{(N)}\right) \ , \qquad (4.11)$$

formed with the complete set of weights $\left\{J_\sigma^{(n)}\right\}$. On the other hand, Hadamard matrices are made of orthogonal binary $[-1,+1]$ valued vectors, being their rows and columns orthogonal and fulfilling the relations

$$\det\left\{H_{2^N\times 2^N}\right\} \neq 0 \ , \tag{4.12}$$

$$H_{2^N\times 2^N}\cdot H^T_{2^N\times 2^N} = H^T_{2^N\times 2^N}\cdot H_{2^N\times 2^N} = 2^N I \ , \tag{4.13}$$

$$H^{-1}_{2^N\times 2^N} = \frac{1}{2^N} H^T_{2^N\times 2^N} \ . \tag{4.14}$$

Due to these properties, Eq. 4.10 can be multiplied by the transpose of the Hadamard matrix at the right to find a solution for the system

$$H^T_{2^N\times 2^N}\cdot \ln\left[\vec{p}(\alpha,\gamma)\right] = H^T_{2^N\times 2^N}\cdot H_{2^N\times 2^N}\cdot \vec{J} = 2^N I\cdot \vec{J} \ . \tag{4.15}$$

The general solution of the system of equations reads

$$\vec{J} = \frac{H^T_{2^N\times 2^N}\cdot \ln\left[\vec{p}(\alpha,\gamma)\right]}{2^N} \ . \tag{4.16}$$

In summary, a HOBM with no hidden units and a fully known probability distribution has a unique set of weights $\left\{J_\sigma^{(n)}\right\}$ given by this expression. **The arguments given above show also that the HOBM with no hidden units is always able to learn any probability distribution provided that no state has zero probability. Furthermore, this same argument implies that there is no need to use hidden units in a HOBM.** In this sense, the learning problem for that network is completely solved by the expression in Eq. 4.16. Notice now that, as it happened with the high order Decimation method in chapter 3, this is also the expression that is used to carry out a Walsh Hadamard transform [Shanks, 1969] over the logarithm of the probabilities that the system is expected to learn.

In order to write down an explicit solution for each weight, we now refer to the Hadamard matrix in terms of its columns

$$H_{2^N\times 2^N} = \tag{4.17}$$
$$= \Big(\ \{1\} \ \{S_1\} \ \ldots \ \{S_N\} \ \{S_1S_2\} \ \ldots \ \{S_NS_{N-1}\} \ \ldots \ \{S_1S_2\cdots S_{N-1}S_N\} \ \Big) \ .$$

This matrix is generated as follows: the first column is set at 1. The next N columns are then built by writing all the values that the N units from the neural network can take; this sequence is defined as the same binary counting that is used to order the probability distribution $\ln[\vec{p}(\alpha,\gamma)]$ that is shown to the system, and that is depicted in table 4.1. Notice also that these columns correspond to the units that are multiplying the bias terms entering in Eq. 4.6, though the sum is carried out with this ordering.

The next $\begin{pmatrix} N \\ 2 \end{pmatrix}$ columns correspond to the products of the units that connect the second order weights in Eq. 4.6. These columns are generated by multiplying the previous columns term by term, thus using all the possible combinations. The algorithm that is used to generate the next columns is the same: the $\begin{pmatrix} N \\ 3 \end{pmatrix}$ subsequent columns stand for the units associated to the third order terms entering in the energy functionals, these are built by multiplying term by term their correspondent columns from the binary sequence. The same rule applies for the $\begin{pmatrix} N \\ 4 \end{pmatrix}$, $\begin{pmatrix} N \\ 5 \end{pmatrix}$, ... up to $\begin{pmatrix} N \\ N \end{pmatrix}$ terms that connect the weights from the neural network. Since there is a relationship between the units connected to a given weight and the column of the Hadamard matrix, we will denote each vector column as $H_{\sigma}^{(n)}$, where (n) denotes the number of units the weight connects and σ is its label. Therefore, we have

$$
\begin{aligned}
&H_{2^N\times 2^N} = \qquad\qquad (4.18)\\
&= \begin{pmatrix} H^{(0)} & H_1^{(1)} & \dots & H_N^{(1)} & H_{12}^{(2)} & \dots & H_{N-1N}^{(2)} & \dots & H_{12\cdots N-1N}^{(N)} \end{pmatrix}\\
&= \begin{pmatrix} \{1\} & \{S_1\} & \dots & \{S_N\} & \{S_1S_2\} & \dots & \{S_NS_{N-1}\} & \dots & \{S_1S_2\cdots S_{N-1}S_N\} \end{pmatrix}.
\end{aligned}
$$

In this sense, each element from Eq. 4.16 can be written in the form

$$
J_{\sigma}^{(n)} = \frac{\left(H_{\sigma}^{(n)}\right)^T \cdot \ln[\vec{p}(\alpha,\gamma)]}{2^N} \ , \qquad (4.19)
$$

therefore the solution to a given weight of the neural network uses the column vector of the units it connects. Finally, one can apply an exponential operation to both sides of

Eq. 4.6 to find

$$\begin{aligned} p(\alpha,\gamma) &= \mathrm{e}^{J^{(0)}}\mathrm{e}^{\sum_{\sigma,n>0}J_\sigma^{(n)}\prod S_\rho} \ , \\ \sum_{\alpha,\gamma} p(\alpha,\gamma) &= \sum_{\alpha,\gamma}\mathrm{e}^{J^{(0)}}\mathrm{e}^{\sum_{\sigma,n>0}J_\sigma^{(n)}\prod S_\rho} \ , \end{aligned} \tag{4.20}$$

where all the equations of the set have been added up, thus arriving at

$$\begin{aligned} 1 &= \mathrm{e}^{J^{(0)}}\sum_{\alpha,\gamma}\mathrm{e}^{\sum_{\sigma,n>0}J_\sigma^{(n)}\prod S_\rho} \ , \\ \mathrm{e}^{-J^{(0)}} &= \sum_{\alpha,\gamma}\mathrm{e}^{\sum_{\sigma,n>0}J_\sigma^{(n)}\prod S_\rho} \ , \\ -J^{(0)} &= \ln\mathcal{Z} \ , \end{aligned} \tag{4.21}$$

hence providing a real solution for the $J^{(0)}$ term, which is now tied to the partition function of the system.

4.4.2 Backwards problem solution for a three units BM

We carry out now a backwards problem example with a small Hadamard matrix, corresponding to a two input units S_{i_1} and S_{i_2} and an output neuron S_o, as it can be seen on Fig. 4.2. We want this system to learn a known $p(\alpha,\gamma)$ probability distribution for all α, γ, which is shown in table 4.2. The values of the probability distribution have been given at random, and are also included in this table.

For the sake of simplicity, we write down the matrix in this order: input units will be written down first, as any possible combinations involving only input units. We will write then all connections between the inputs and the output, to conclude with the output cell

$$\vec{S} = \begin{pmatrix} 1 & S_{i_1} & S_{i_2} & S_o & S_{i_1}S_{i_2} & S_{i_1}S_o & S_{i_2}S_o & S_{i_1}S_{i_2}S_o \end{pmatrix} \ . \tag{4.22}$$

We now proceed to show how the matrix of the system is created for this example: the first four columns of the Hadamard matrix are the first to be written down. The first

$\vec{p}(\alpha,\gamma)$	$\ln[\vec{p}(\alpha,\gamma)]$	S_{i_1}	S_{i_2}	S_o
$p(\alpha_1,\gamma_1)=0.2158$	$\ln p(\alpha_1,\gamma_1)=-1.5333$	-1	-1	-1
$p(\alpha_2,\gamma_1)=0.0525$	$\ln p(\alpha_2,\gamma_1)=-2.9469$	-1	-1	1
$p(\alpha_1,\gamma_2)=0.1378$	$\ln p(\alpha_1,\gamma_2)=-1.9816$	-1	1	-1
$p(\alpha_2,\gamma_2)=0.1104$	$\ln p(\alpha_2,\gamma_2)=-2.2037$	-1	1	1
$p(\alpha_1,\gamma_3)=0.2025$	$\ln p(\alpha_1,\gamma_3)=-1.5972$	1	-1	-1
$p(\alpha_2,\gamma_3)=0.1731$	$\ln p(\alpha_2,\gamma_3)=-1.7538$	1	-1	1
$p(\alpha_1,\gamma_4)=0.1037$	$\ln p(\alpha_1,\gamma_4)=-2.2664$	1	1	-1
$p(\alpha_2,\gamma_4)=0.0042$	$\ln p(\alpha_2,\gamma_4)=-5.4720$	1	1	1

Table 4.2: Probability distribution for the three units example.

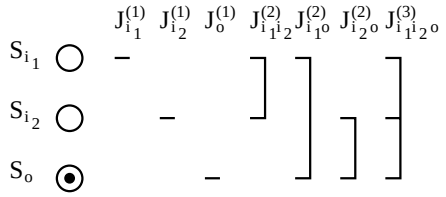

Figure 4.2: Three units neural network, with two inputs and an output unit.

column is a 1 value vector, the other ones refer to units S_{i_1}, S_{i_2} and S_o, respectively

$$H = \begin{pmatrix} 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \tag{4.23}$$

we then add the first set of products $S_{i_1}S_{i_2}$, $S_{i_1}S_o$, $S_{i_2}S_o$ as the next three columns of the

system

$$H = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} , \tag{4.24}$$

finally the $S_{i_1} S_{i_2} S_o$ product column is added

$$H = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} . \tag{4.25}$$

We now find the analytical values for all the weights of the system by using

$$J_\sigma^{(n)} = \frac{\left(H_\sigma^{(n)}\right)^T \cdot \ln\left[\vec{p}(\alpha, \gamma)\right]}{2^N} ,$$

assuming that

$$\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=\begin{pmatrix}-1.5333\\-2.9469\\-1.9816\\-2.2037\\-1.5972\\-1.7538\\-2.2664\\-5.4720\end{pmatrix}\ , \tag{4.26}$$

then

$$\begin{aligned}
J^{(0)}&=\frac{1}{2^3}\begin{pmatrix}1&1&1&1&1&1&1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-2.4694\ ,\\
J^{(1)}_{i_1}&=\frac{1}{2^3}\begin{pmatrix}-1&-1&-1&-1&1&1&1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.3030\ ,\\
J^{(1)}_{i_2}&=\frac{1}{2^3}\begin{pmatrix}-1&-1&1&1&-1&-1&1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.5116\ ,\\
J^{(1)}_{i_3}&=\frac{1}{2^3}\begin{pmatrix}-1&1&-1&1&-1&1&-1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.6247\ ,\\
J^{(2)}_{i_1i_2}&=\frac{1}{2^3}\begin{pmatrix}1&1&-1&-1&-1&-1&1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.5853\ ,\\
J^{(2)}_{i_1o}&=\frac{1}{2^3}\begin{pmatrix}1&-1&1&-1&-1&1&-1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.2158\ ,\\
J^{(2)}_{i_2o}&=\frac{1}{2^3}\begin{pmatrix}1&-1&-1&1&1&-1&-1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.2322\ ,\\
J^{(3)}_{i_1i_2o}&=\frac{1}{2^3}\begin{pmatrix}-1&1&1&-1&1&-1&-1&1\end{pmatrix}\cdot\ln\left[\vec{p}\left(\alpha,\gamma\right)\right]=-0.5301\ .
\end{aligned} \tag{4.27}$$

4.4.3 The backwards problem for a conditional p.d.f.

In this section we extend the previous discussion to the more realistic learning problem involving conditional probability distributions. A standard learning problem on a BM is usually given in terms of conditional probabilities: given a fixed input pattern γ, we know the probability $p\left(\alpha\mid\gamma\right)$ of finding a state α in the output units. Of course, we typically know $p\left(\alpha\mid\gamma\right)$ only for a restricted set of states γ and α and the network has to infer the

remaining probabilities. We now discuss what we call *the complete backwards problem* for a conditional probability distribution: given all the conditional probabilities $p(\alpha \mid \gamma)$ find the complete set of weights $\left\{J_{\sigma}^{(n)}\right\}$ of the HOBM that conforms to these probabilities.

We begin this discussion with an N-th order Boltzmann Machine with n_i input units S_i, n_o output units S_o and no hidden units, for a total of $N = n_i + n_o$ neurons. We do not know the value of its connections, though we know that the neural network may have all possible weights up to order N. In this instance of the backwards problem, we assume we know the complete conditional probability distribution associated to the HOBM, that is, we assume we know $p(\alpha \mid \gamma)$ for every output state α, conditioned to a clamped input state γ. We again look for a set of weights that reproduces this probability distribution. We know that, by definition

$$p(\alpha \mid \gamma) = \frac{p(\alpha, \gamma)}{p(\gamma)} \ , \tag{4.28}$$

so we can replace $p(\alpha, \gamma)$ by $p(\alpha \mid \gamma)\, p(\gamma)$ in Eq. 4.16 to find

$$\vec{J} = \frac{H^{T}_{2^N \times 2^N} \cdot (\ln[\vec{p}(\alpha \mid \gamma)] + \ln[\vec{p}(\gamma)])}{2^N} \ . \tag{4.29}$$

In this new formulation of the problem, we know all the $p(\alpha \mid \gamma)$. However, additional knowledge of $p(\gamma)$ is needed. One may wonder if $p(\gamma)$ can be recovered once we know $p(\alpha \mid \gamma)$ for every state α and γ. This is not the case, as there are infinite possible choices for $p(\gamma)$. In this sense, we realize that knowing $p(\alpha, \gamma)$ is enough to get $p(\alpha \mid \gamma)$ and $p(\gamma)$ but the inverse is not true. In order to see the conditional probabilities do not fix univoquely the input probabilities we discuss a simple example.

Let A and B be two binary (0 and 1) random variables that happen with probability $p(A, B)$. The conditional probabilities $p(A \mid B)$ that can be derived are

$$\begin{aligned}
p(A = 0 \mid B = 0) &= \frac{p(A = 0, B = 0)}{p(B = 0)} \ , \\
p(A = 1 \mid B = 0) &= \frac{p(A = 1, B = 0)}{p(B = 0)} \ , \\
p(A = 0 \mid B = 1) &= \frac{p(A = 0, B = 1)}{p(B = 1)} \ , \\
p(A = 1 \mid B = 1) &= \frac{p(A = 1, B = 1)}{p(B = 1)} \ ,
\end{aligned} \tag{4.30}$$

with the normalization condition

$$p\left(A=0,B=0\right)+p\left(A=1,B=0\right)+p\left(A=0,B=1\right)+p\left(A=1,B=1\right)=1\ . \tag{4.31}$$

Consider now a different probability distribution $q\left(A,B\right)$ such that

$$\begin{aligned} q\left(A=0,B=0\right) &= \lambda\ p\left(A=0,B=0\right)\ , \\ q\left(A=1,B=0\right) &= \lambda\ p\left(A=1,B=0\right)\ , \\ q\left(A=0,B=1\right) &= \mu\ p\left(A=0,B=1\right)\ , \\ q\left(A=1,B=1\right) &= \mu\ p\left(A=1,B=1\right)\ , \end{aligned} \tag{4.32}$$

for some real values of λ and μ. In the following, we show that there is an infinite set of solutions for λ and μ that produce the same conditional probabilities. First of all, the substitution of Eq. 4.32 into Eq. 4.30 tells us that

$$\begin{aligned} p\left(A=0\mid B=0\right) &= \frac{p\left(A=0,B=0\right)}{p\left(B=0\right)} = \frac{q\left(A=0,B=0\right)}{q\left(B=0\right)} = q\left(A=0\mid B=0\right)\ , \\ p\left(A=1\mid B=0\right) &= \frac{p\left(A=1,B=0\right)}{p\left(B=0\right)} = \frac{q\left(A=1,B=0\right)}{q\left(B=0\right)} = q\left(A=1\mid B=0\right)\ , \\ p\left(A=0\mid B=1\right) &= \frac{p\left(A=0,B=1\right)}{p\left(B=1\right)} = \frac{q\left(A=0,B=1\right)}{q\left(B=1\right)} = q\left(A=0\mid B=1\right)\ , \\ p\left(A=1\mid B=1\right) &= \frac{p\left(A=1,B=1\right)}{p\left(B=1\right)} = \frac{q\left(A=1,B=1\right)}{q\left(B=1\right)} = q\left(A=1\mid B=1\right)\ , \end{aligned} \tag{4.33}$$

so the conditional probabilities are the same. Now we show that these relations and the previous ones can be fulfilled for values other than the trivial $\lambda=\mu=1$. We impose now the additional normalization constraint

$$q\left(A=0,B=0\right)+q\left(A=1,B=0\right)+q\left(A=0,B=1\right)+q\left(A=1,B=1\right)=1\ , \tag{4.34}$$

which means

$$\lambda p\left(A=0,B=0\right)+\lambda p\left(A=1,B=0\right)+\mu p\left(A=0,B=1\right)+\mu p\left(A=1,B=1\right)=1\ . \tag{4.35}$$

Since both probability distributions are (properly) normalized to one, the sum over all $p(A,B)$ equals the sum over all $q(A,B)$ and then we can write

$$\begin{aligned}&\lambda\left[p(A=0,B=0)+p(A=1,B=0)\right]+\mu\left[p(A=0,B=1)+p(A=1,B=1)\right]=\\&=p(A=0,B=0)+p(A=1,B=0)+p(A=0,B=1)+p(A=1,B=1)\ .\end{aligned} \tag{4.36}$$

From here, we arrive to the condition

$$\frac{\lambda-1}{\mu-1}=-\frac{p(A=0,B=1)+p(A=1,B=1)}{p(A=0,B=0)+p(A=1,B=0)}\ , \tag{4.37}$$

where

$$\lambda=1+\frac{p(A=0,B=1)+p(A=1,B=1)}{p(A=0,B=0)+p(A=1,B=0)}(1-\mu)\ . \tag{4.38}$$

Now, we give some values to the previous numbers and carry out the following example

$$\begin{aligned}p(A=0,B=0)&=\frac{1}{10}\ ,\\p(A=1,B=0)&=\frac{2}{10}\ ,\\p(A=0,B=1)&=\frac{3}{10}\ ,\\p(A=1,B=1)&=\frac{4}{10}\ .\end{aligned} \tag{4.39}$$

We will find a $q(A,B)$ for $\lambda=\frac{3}{2}$ and $\lambda=2$. We fix $\lambda=\frac{3}{2}$ and find a suitable value for μ, thus

$$\frac{\frac{3}{2}-1}{\mu-1}=-\frac{\frac{3}{10}+\frac{4}{10}}{\frac{1}{10}+\frac{2}{10}}\ , \tag{4.40}$$

so

$$\mu=\frac{11}{14}\ , \tag{4.41}$$

and generate $q(A,B)$

$$\begin{aligned} q(A=0,B=0) &= \frac{3}{20} \ , \\ q(A=1,B=0) &= \frac{3}{10} \ , \\ q(A=0,B=1) &= \frac{33}{140} \ , \\ q(A=1,B=1) &= \frac{44}{140} \ . \end{aligned} \tag{4.42}$$

We sum these values to ensure that λ and μ are correct

$$\frac{3}{20}+\frac{3}{10}+\frac{33}{140}+\frac{44}{140} = \frac{21}{140}+\frac{42}{140}+\frac{33}{140}+\frac{44}{140} = 1 \ . \tag{4.43}$$

We now repeat this process for $\lambda = 2$, hence

$$\frac{2-1}{\mu-1} = -\frac{\frac{3}{10}+\frac{4}{10}}{\frac{1}{10}+\frac{2}{10}} \ , \tag{4.44}$$

and

$$\mu = \frac{4}{7} \ , \tag{4.45}$$

the new probability distribution $q(A,B)$ reads then as

$$\begin{aligned} q(A=0,B=0) &= \frac{1}{5} \ , \\ q(A=1,B=0) &= \frac{2}{5} \ , \\ q(A=0,B=1) &= \frac{12}{70} \ , \\ q(A=1,B=1) &= \frac{16}{70} \ . \end{aligned} \tag{4.46}$$

which does also satisfy that the sum of its terms is 1

$$\frac{1}{5}+\frac{2}{5}+\frac{12}{70}+\frac{16}{70} = \frac{14}{70}+\frac{28}{70}+\frac{12}{70}+\frac{16}{70} = 1 \ . \tag{4.47}$$

Notice then that we could find as many different values as desired for λ, and that it would lead to different values of μ that would yet provide new valid probability distributions. In this sense, there is an infinite set of probability distributions that lead to the same conditional distribution.

4.4.4 General solution for the backwards conditional problem

In this section we prove that there exists a general solution for the conditional probability backwards problem, which is formulated as

$$J_{\sigma}^{(n)} = \frac{H_{\sigma}^{(n)} \cdot \ (\ln [\vec{p}\,(\alpha \mid \gamma)] + \ln [\vec{p}\,(\gamma)])}{2^N} \ , \tag{4.48}$$

and that it can be found even if the input units probability distribution is not known. Though, this solution does only provide the values for the weights that are not limited to connecting only input units. The easiest case is considered to happen on a two inputs, one output topology; we then propose the generic case for N units, to prove that the system will yet be able to solve the backwards problem regardless the value $p\,(\gamma)$ might take.

We begin with a high order Boltzmann Machine as the one proposed in the previous section, with two input units S_{i_1} and S_{i_2} and an output neuron S_o. The vector of weights of the system reads as

$$\vec{J} = \begin{pmatrix} J^{(0)} \\ J_{i_1}^{(1)} \\ J_{i_2}^{(1)} \\ J_{o}^{(1)} \\ J_{i_1 i_2}^{(2)} \\ J_{i_1 o}^{(2)} \\ J_{i_2 o}^{(2)} \\ J_{i_1 i_2 o}^{(3)} \end{pmatrix} \ , \tag{4.49}$$

and the probability distribution is written down on Table 4.3.

We start from the probability distribution system of equations

$$\vec{J} = \frac{H_{2^3 \times 2^3}^{T} \cdot \ln [\vec{p}\,(S_o, S_{i_1}, S_{i_2})]}{2^3} \ , \tag{4.50}$$

S_{i_1}	S_{i_2}	S_o	$p\left(S_{i_1}, S_{i_2}, S_o\right) = p\left(S_o \vert S_{i_1}, S_{i_2}\right) p\left(S_{i_1}, S_{i_2}\right)$
-1	-1	-1	$p\left(-1,-1,-1\right) = p\left(-1 \mid -1,-1\right) p\left(-1,-1\right)$
-1	-1	1	$p\left(-1,-1,1\right) = p\left(1 \mid -1,-1\right) p\left(-1,-1\right)$
-1	1	-1	$p\left(-1,1,-1\right) = p\left(-1 \mid -1,1\right) p\left(-1,1\right)$
-1	1	1	$p\left(-1,1,1\right) = p\left(1 \mid -1,1\right) p\left(-1,1\right)$
1	-1	-1	$p\left(1,-1,-1\right) = p\left(-1 \mid 1,-1\right) p\left(1,-1\right)$
1	-1	1	$p\left(1,-1,1\right) = p\left(1 \mid 1,-1\right) p\left(1,-1\right)$
1	1	-1	$p\left(1,1,-1\right) = p\left(-1 \mid 1,1\right) p\left(1,1\right)$
1	1	1	$p\left(1,1,1\right) = p\left(1 \mid 1,1\right) p\left(1,1\right)$

Table 4.3: Standard and conditional probability distributions.

the solution for this system is

$$
\begin{aligned}
J_{i_1}^{(1)} &= \ln\left(\frac{p\left(-1,1,-1\right)p\left(1,1,-1\right)p\left(-1,1,1\right)p\left(1,1,1\right)}{p\left(-1,-1,-1\right)p\left(1,-1,-1\right)p\left(-1,-1,1\right)p\left(1,-1,1\right)}\right)\\
J_{i_2}^{(1)} &= \ln\left(\frac{p\left(-1,-1,1\right)p\left(-1,1,1\right)p\left(1,-1,1\right)p\left(1,1,1\right)}{p\left(-1,-1,-1\right)p\left(-1,1,-1\right)p\left(1,-1,-1\right)p\left(1,1,-1\right)}\right)\\
J_{o}^{(1)} &= \ln\left(\frac{p\left(1,-1,-1\right)p\left(1,1,-1\right)p\left(1,-1,1\right)p\left(1,1,1\right)}{p\left(-1,-1,-1\right)p\left(-1,1,-1\right)p\left(-1,-1,1\right)p\left(-1,1,1\right)}\right)\\
J_{i_1 i_2}^{(2)} &= \ln\left(\frac{p\left(-1,-1,-1\right)p\left(1,-1,-1\right)p\left(-1,1,1\right)p\left(1,1,1\right)}{p\left(-1,-1,1\right)p\left(1,-1,1\right)p\left(-1,1,-1\right)p\left(1,1,-1\right)}\right)\\
J_{i_1 o}^{(2)} &= \ln\left(\frac{p\left(-1,-1,-1\right)p\left(-1,-1,1\right)p\left(1,1,-1\right)p\left(1,1,1\right)}{p\left(1,-1,-1\right)p\left(1,-1,1\right)p\left(-1,1,-1\right)p\left(-1,1,1\right)}\right)\\
J_{i_2 o}^{(2)} &= \ln\left(\frac{p\left(-1,-1,-1\right)p\left(1,-1,1\right)p\left(-1,1,-1\right)p\left(1,1,1\right)}{p\left(1,-1,-1\right)p\left(-1,-1,1\right)p\left(1,1,-1\right)p\left(-1,1,1\right)}\right)\\
J_{i_1 i_2 o}^{(3)} &= \ln\left(\frac{p\left(1,-1,-1\right)p\left(-1,-1,1\right)p\left(-1,1,-1\right)p\left(1,1,1\right)}{p\left(-1,-1,-1\right)p\left(1,-1,1\right)p\left(1,1,-1\right)p\left(-1,1,1\right)}\right) \quad .
\end{aligned}
\tag{4.51}
$$

The conditional probability solution shows that the input probability references are

cleared for any solution that is not limited to connecting only input units

$$
\begin{aligned}
J_{i_1}^{(1)} &= \ln\left(\frac{p\left(-1\mid 1,-1\right)p\left(1\mid 1,-1\right)p\left(-1\mid 1,1\right)p\left(1\mid 1,1\right)}{p\left(-1\mid -1,-1\right)p\left(1\mid -1,-1\right)p\left(-1\mid -1,1\right)p\left(1\mid -1,1\right)}\right) \\
&\quad +2\ln\left(\frac{p\left(1,-1\right)p\left(1,1\right)}{p\left(-1,1\right)p\left(-1,-1\right)}\right) \ , \\
J_{i_2}^{(1)} &= \ln\left(\frac{p\left(-1\mid -1,1\right)p\left(-1\mid 1,1\right)p\left(1\mid -1,1\right)p\left(1\mid 1,1\right)}{p\left(-1\mid -1,-1\right)p\left(-1\mid 1,-1\right)p\left(1\mid -1,-1\right)p\left(1\mid 1,-1\right)}\right) \\
&\quad +2\ln\left(\frac{p\left(-1,1\right)p\left(1,1\right)}{p\left(1,-1\right)p\left(-1,-1\right)}\right) \ , \\
J_{o}^{(1)} &= \ln\left(\frac{p\left(1\mid -1,-1\right)p\left(1\mid 1,-1\right)p\left(1\mid -1,1\right)p\left(1\mid 1,1\right)}{p\left(-1\mid -1,-1\right)p\left(-1\mid 1,-1\right)p\left(-1\mid -1,1\right)p\left(-1\mid 1,1\right)}\right) \ , \\
J_{i_1i_2}^{(2)} &= \ln\left(\frac{p\left(-1\mid -1,-1\right)p\left(1\mid -1,-1\right)p\left(-1\mid 1,1\right)p\left(1\mid 1,1\right)}{p\left(-1\mid -1,1\right)p\left(1\mid -1,1\right)p\left(-1\mid 1,-1\right)p\left(1\mid 1,-1\right)}\right) \\
&\quad +2\ln\left(\frac{p\left(-1,-1\right)p\left(1,1\right)}{p\left(-1,1\right)p\left(1,-1\right)}\right) \ , \\
J_{i_1o}^{(2)} &= \ln\left(\frac{p\left(-1\mid -1,-1\right)p\left(-1\mid -1,1\right)p\left(1\mid 1,-1\right)p\left(1\mid 1,1\right)}{p\left(1\mid -1,-1\right)p\left(1\mid -1,1\right)p\left(-1\mid 1,-1\right)p\left(-1\mid 1,1\right)}\right) \ , \\
J_{i_2o}^{(2)} &= \ln\left(\frac{p\left(-1\mid -1,-1\right)p\left(1\mid -1,1\right)p\left(-1\mid 1,-1\right)p\left(1\mid 1,1\right)}{p\left(1\mid -1,-1\right)p\left(-1\mid -1,1\right)p\left(1\mid 1,-1\right)p\left(-1\mid 1,1\right)}\right) \ , \\
J_{i_1i_2o}^{(3)} &= \ln\left(\frac{p\left(1\mid -1,-1\right)p\left(-1\mid -1,1\right)p\left(-1\mid 1,-1\right)p\left(1\mid 1,1\right)}{p\left(-1\mid -1,-1\right)p\left(1\mid -1,1\right)p\left(1\mid 1,-1\right)p\left(-1\mid 1,1\right)}\right) \ . \qquad (4.52)
\end{aligned}
$$

The resulting equations for the weights connecting output and hidden units are the same as before, but using $p\left(S_o\mid S_{i_1},S_{i_2}\right)$ instead of $p\left(S_{i_1},S_{i_2},S_o\right)$. These probabilities are numerically different and one may think that the equations are, in consequence, different. However, this happens not to be true, since $p\left(S_{i_1},S_{i_2},S_o\right)=p\left(S_o\mid S_{i_1},S_{i_2}\right)p\left(S_{i_1},S_{i_2}\right)$ and those $p\left(S_{i_1},S_{i_2}\right)$ are further cleared. We can see then that the weights $J_o^{(1)}$, $J_{i_1o}^{(2)}$, $J_{i_2o}^{(2)}$ and $J_{i_1i_2o}^{(3)}$ are the same for both systems of equations; we also realize that this happens regardless of the values $p\left(S_{i_1},S_{i_2}\right)$ may take.

Now, we analyze what happens for a number N of units in the neural network, with n_i input units S_i and n_o output units S_o; we consider again that the problem is solved when the values of the weights connecting only output or either input and output units are found. We have a given weight connecting any number m_i of input units S_{i_1}, S_{i_2}, up to $S_{i_{m_i}}$ with any number of m_o output units S_{o_1}, S_{o_2}, up to $S_{o_{m_o}}$, being that $m_o\neq 0$.

From Eq 4.48, we can see that the solution for such weight involves a vector $H_{\sigma}^{(n)}$ which is composed of the combinations of products of the very same units that it connects

$$H_{\sigma}^{(n)} = \left\{ S_{i_1} S_{i_2} \cdots S_{i_{m_i}} S_{o_1} S_{o_2} \cdots S_{o_{m_o}} \right\} \ . \tag{4.53}$$

When the backwards problem is proposed, the Hadamard matrix that is related to such vector is then given values on an orderly manner, as function of the α and γ vectors

$$\begin{aligned}
H_{\sigma,1,1}^{(n)} &= \left\{ S_{i_1}^{\gamma_1} S_{i_2}^{\gamma_1} \cdots S_{i_{m_i}}^{\gamma_1} S_{o_1}^{\alpha_1} S_{o_2}^{\alpha_1} \cdots S_{o_{m_o}}^{\alpha_1} \right\} \ , \\
H_{\sigma,1,2}^{(n)} &= \left\{ S_{i_1}^{\gamma_1} S_{i_2}^{\gamma_1} \cdots S_{i_{m_i}}^{\gamma_1} S_{o_1}^{\alpha_2} S_{o_2}^{\alpha_2} \cdots S_{o_{m_o}}^{\alpha_2} \right\} \ , \\
H_{\sigma,1,3}^{(n)} &= \left\{ S_{i_1}^{\gamma_1} S_{i_2}^{\gamma_1} \cdots S_{i_{m_i}}^{\gamma_1} S_{o_1}^{\alpha_3} S_{o_2}^{\alpha_3} \cdots S_{o_{m_o}}^{\alpha_3} \right\} \ , \\
&\vdots \\
H_{\sigma,2^{m_i},2^{m_o}}^{(n)} &= \left\{ S_{i_1}^{\gamma_{2^{m_i}}} S_{i_2}^{\gamma_{2^{m_i}}} \cdots S_{i_{m_i}}^{\gamma_{2^{m_i}}} S_{o_1}^{\alpha_{2^{m_o}}} S_{o_2}^{\alpha_{2^{m_o}}} \cdots S_{o_{m_o}}^{\alpha_{2^{m_o}}} \right\} \ ,
\end{aligned} \tag{4.54}$$

hence for each input γ vector, we will also have 2^{m_o} different output possible α combinations. We now define a subset of vectors where the input units are clamped to the same γ state and the output units are those 2^{m_o} different α output states associated to such input values

$$\begin{aligned}
H_{\sigma,\gamma,1}^{(n)} &= \left\{ S_{i_1}^{\gamma} S_{i_2}^{\gamma} \cdots S_{i_{m_i}}^{\gamma} S_{o_1}^{\alpha_1} S_{o_2}^{\alpha_1} \cdots S_{o_{m_o}}^{\alpha_1} \right\} \ , \\
H_{\sigma,\gamma,2}^{(n)} &= \left\{ S_{i_1}^{\gamma} S_{i_2}^{\gamma} \cdots S_{i_{m_i}}^{\gamma} S_{o_1}^{\alpha_2} S_{o_2}^{\alpha_2} \cdots S_{o_{m_o}}^{\alpha_2} \right\} \ , \\
&\vdots \\
H_{\sigma,\gamma,2^{m_o}}^{(n)} &= \left\{ S_{i_1}^{\gamma} S_{i_2}^{\gamma} \cdots S_{i_{m_i}}^{\gamma} S_{o_1}^{\alpha_{2^{m_o}}} S_{o_2}^{\alpha_{2^{m_o}}} \cdots S_{o_{m_o}}^{\alpha_{2^{m_o}}} \right\} \ .
\end{aligned} \tag{4.55}$$

Notice that any $H_{\sigma,\gamma,\alpha}^{(n)}$ is a vector that considers all possible product terms from the units that the weight is connecting, and that this is true for all α related to the same γ input vector. This is then a vector with the same number of $+1$ and -1 terms; thus $H_{\sigma,\gamma,\alpha}^{(n)} \cdot \vec{1} = 0$, hence for any weight connecting only output or, either, input and output

units

$$\begin{aligned} J_{\sigma}^{(n)} &= \frac{H_{\sigma}^{(n)} \cdot \left(\ln\left[\vec{p}\left(\alpha \mid \gamma\right)\right]\right)}{2^N} + \frac{H_{\sigma}^{(n)} \cdot \left(\ln\left[\vec{p}\left(\gamma\right)\right]\right)}{2^N} \\ &= \frac{H_{\sigma}^{(n)} \cdot \left(\ln\left[\vec{p}\left(\alpha \mid \gamma\right)\right]\right)}{2^N} + \sum_{\gamma} \frac{\ln\left[\vec{p}\left(\gamma\right)\right]\left(H_{\sigma,\alpha,\gamma}^{(n)} \cdot \vec{1}\right)}{2^N} \ , \end{aligned} \tag{4.56}$$

being $\ln\left[\vec{p}\left(\gamma\right)\right]$ the probabilities of reaching a given γ input state, expressed as a vector. Finally

$$J_{\sigma}^{(n)} = \frac{H_{\sigma}^{(n)} \cdot \left(\ln\left[\vec{p}\left(\alpha \mid \gamma\right)\right]\right)}{2^N} \ , \tag{4.57}$$

as far as $J_{\sigma}^{(n)}$ only connects input and output, or only output units.

4.4.5 The backwards incomplete problem

We have already analyzed what happens when an N units, N-th order HOBM with no hidden units has to learn a probability distribution $p\left(\alpha, \gamma\right)$ known for all α, γ: there is only one possible set of weights that is able to yield the corresponding Boltzmann probability distribution, as far as there are no $p = 0$ nor $p = 1$ values. We will now discuss what happens when this is not the case, that is, when we only know some of the probabilities $p\left(\alpha, \gamma\right)$; this is also the realistic case when the neural network is learning a set of vectors that is not fully defined.

We begin from a standard, high order BM with n_i input units S_i and n_o output units S_o for a total of $n_i + n_o = N$ neurons, and a partially known absolute probability distribution $p\left(\alpha, \gamma\right)$ such as

$$p\left(\alpha, \gamma\right) = \left\{\varphi_1 \ldots \varphi_{m_1} \lambda_1 \ldots \lambda_{m_2}\right\} \ , \tag{4.58}$$

where the subset φ_i of $m_1 < 2^N$ values is known and we consider that there are $m_2 = 2^N - m_1$ unknown λ_j m_2 vectors. We make use of Eq. 4.10 to relate the probability distribution from Eq. 4.58 to the set of weights for this neural network

$$\ln\left[\vec{p}\left(\alpha, \gamma\right)\right] = H_{2^N \times 2^N} \cdot \vec{J} \ ,$$

this system of equations can be written down on its matricial notation

$$\ln \begin{pmatrix} \varphi_1 \\ \cdots \\ \varphi_{m_1} \\ \lambda_1 \\ \cdots \\ \lambda_{m_2} \end{pmatrix} = H_{2^N \times 2^N} \cdot \vec{J} \ , \tag{4.59}$$

where there are a total of m_2 unknown values as the natural logarithm of the terms from the probability distribution. Let then any λ_j take an arbitrary value -constrained to $\sum_{\alpha,\gamma} p(\alpha,\gamma) = 1$ and $p(\alpha,\gamma) \neq 0$, $p(\alpha,\gamma) \neq 1$-; then Eq. 4.59 is solved regardless of these values, because the matrix of the system is Hadamard. In this sense, there is an infinite range of solutions that will yet be compatible with φ_i terms.

Furthermore, let $r(\alpha,\gamma)$ be a non-normalized probability distribution such as

$$\sum_{\alpha,\gamma} r(\alpha,\gamma) = k \ , \tag{4.60}$$

for $k \in \Re^+$ and $r(\alpha,\gamma) > 0$, $\forall \alpha,\gamma$; thus assuming that all λ_j values are again arbitrary. Notice that Eq. 4.59 is solved regardless of these values, because the matrix of the system is Hadamard. Let $p(\alpha,\gamma)$ be a normalized probability distribution such as

$$p(\alpha,\gamma) = \frac{r(\alpha,\gamma)}{k} \ , \tag{4.61}$$

thus $\sum_{\alpha,\gamma} p(\alpha,\gamma) = 1$. Notice that

$$\ln \vec{p}\,(\alpha,\gamma) = \ln \left[\frac{\vec{r}\,(\alpha,\gamma)}{k} \right] = H_{2^N \times 2^N} \cdot \vec{J} \ , \tag{4.62}$$

the constant can be moved to the rhs of the equation

$$\ln \vec{r}\,(\alpha,\gamma) = \ln k + H_{2^N \times 2^N} \cdot \vec{J} \ , \tag{4.63}$$

effectively being added to the $J^{(0)}$ constant term: notice from Eq. 4.6 that this zero order terms always has 1 as coefficient and therefore arriving at a new $J^{(0)'} = J^{(0)} + \ln k$. In this sense, the set $\vec{J}_\sigma^{(n>0)}$ is the same for both $p(\alpha,\gamma)$ and $r(\alpha,\gamma)$.

Hence, the system for a partially known normalized probability distribution $p(\alpha, \gamma)$ where the unknown terms can be arbitrarily fixed -such that $\sum_{\alpha,\gamma} p(\alpha, \gamma) = 1$- reads as

$$\ln \begin{pmatrix} \varphi_1 \\ \dots \\ \varphi_{m_1} \end{pmatrix} = H_{2^N \times m_1} \cdot \vec{J} \ , \tag{4.64}$$

where the rows of the Hadamard matrix considered are those from the known probability distribution. We will arrive to a system where as much as m_1 weights will need to be fixed to obtain the numerical values that match the probability distribution. Therefore, the system is not analytically solved but rather it should be solved by numerical means.

4.5 Backwards incomplete problem LU solution

The backwards complete problem is a unique problem approach as far as the learning vectors are completely known, which is something that does not happen when dealing against real-life problems. It has already been shown that this problem is always accurately solved for a high order Boltzmann Machine with no hidden units, since the Hadamard matrix that is generated by the very same vectors leads to a solvable system of equations

$$\ln \vec{p}(\alpha \mid \gamma) = H_{2^N \times 2^N} \cdot \vec{J} \ ,$$

where N stands for the total number of both input and output units, $\vec{p}(\alpha \mid \gamma)$ is a 2^N component vector of probabilities for the system and $\vec{J}$ is vector that represents the weights of the BM.

In this section we will discuss the solution for a learning set of vectors that is not complete, which has been named as the backwards incomplete problem. As it has already been shown, the system of equations that is generated in this case is incomplete

$$\ln \vec{p}(\alpha, \gamma) = H_p \cdot \vec{J} \ ,$$

where H_p is a non-square matrix that has some selected rows from a Hadamard matrix, thus obtaining a certain number of degrees of freedom that should be fixed. In this

section we propose a simple solution to this problem, which is a regression through the LU decomposition of H_p. In the first part of this section we show that this solution will indeed reach a global minimum of the Kullback-Leibler distance; the section is then concluded with the application of the LU solution to a typical toy problem, which is known as the *priority encoder*.

4.5.1 Kullback-Leibler distance optimization and the LU solution

We now inquiry about the correct solution to the backwards incomplete problem that one can obtain through LU factorization. Since this method leads to a mathematical regression, the incomplete backwards problem will be solved as a regression where the points are approximated by a function which is now the high order energy functional of the BM. The matricial equation for the incomplete backwards problem reads as

$$\ln \vec{p}(\alpha, \gamma) = H_p \cdot \vec{J} \ , \tag{4.65}$$

where $\vec{p}(\alpha, \gamma)$ is the set of m probability values that are known at this learning stage, from a total of 2^N for a total of N units. On the other hand, $\vec{J}$ are the weights that the neural network will set up once it learns, while H_p is a matrix whose rows are the combinations of input and output units that are known. Its columns are the same as a Hadamard matrix hence

$$H_p \cdot H_p' = mI \ , \tag{4.66}$$

being m the number of rows that the matrix has and I the identity matrix. Since this is *not* a Hadamard matrix, it also happens that

$$H_p' \cdot H_p \neq mI \ , \tag{4.67}$$

notice that the dimension of this matrix is actually $2^N \times m$, and that we have $m^{'} = 2^N - m$ degrees of freedom. Notice also that, since this is an incomplete learning set, the real value

for these weights is not known and may not be set to zero. Said this, we show a possible solution for this problem, which consists on a standard regression calculus and show it to be a global minimum of the Kullback-Leibler distance.

Let an error function Θ which results from the LU factorization of the previous system, as seen in Ref. [The Mathworks^{TM}, 2008a], be defined as

$$\Theta = \frac{1}{2} \sum_{\alpha,\gamma} \left(\sum_{\sigma} J_{\sigma}^{(n)} \prod S_{\rho \in \alpha,\gamma} - \ln p\left(\alpha, \gamma\right) \right)^2 \quad , \tag{4.68}$$

where $\prod S_{\rho \in \alpha,\gamma}$ stands for all the units which are linked by a given connection $J_{\sigma}^{(n)}$ when an input pattern γ and an output pattern α are set. This system is solved by using Least Squares, hence we need to carry out the partial derivative of this expression for a given weight $J_{\tau}^{(n)}$ as

$$\frac{\partial \Theta}{\partial J_{\tau}^{(n)}} = \sum_{\alpha,\gamma} \left(\sum_{\sigma} J_{\sigma}^{(n)} \prod S_{\rho \in \alpha,\gamma} - \ln p\left(\alpha, \gamma\right) \right) \prod S_{\varrho \in \alpha,\gamma} = 0 \quad , \tag{4.69}$$

$\prod S_{\varrho \in \alpha,\gamma}$ being the units linked by the weight $J_{\tau}^{(n)}$. This expression can be worked out to reach

$$\sum_{\alpha,\gamma} \left(\sum_{\sigma} J_{\sigma}^{(n)} \prod S_{\rho \in \alpha,\gamma} \right) \prod S_{\varrho \in \alpha,\gamma} = \sum_{\alpha,\gamma} \ln p\left(\alpha, \gamma\right) \prod S_{\varrho \in \alpha,\gamma} \quad , \tag{4.70}$$

now we apply the conditioned probability definition where

$$p\left(\alpha, \gamma\right) = p\left(\alpha \mid \gamma\right) p\left(\gamma\right) \quad , \tag{4.71}$$

thus arriving at

$$\sum_{\alpha,\gamma} \left(\sum_{\sigma} J_{\sigma}^{(n)} \prod S_{\rho \in \alpha,\gamma} \right) \prod S_{\varrho \in \alpha,\gamma} = \sum_{\alpha,\gamma} \left(\ln p\left(\alpha \mid \gamma\right) + \ln p\left(\gamma\right) \right) \prod S_{\varrho \in \alpha,\gamma} \quad . \tag{4.72}$$

We now recall the discussion about the relevance of $p\left(\gamma\right)$ over the values of $p\left(\alpha \mid \gamma\right)$ from section 4.4.4: the value of the weights is the same for a given $p\left(\alpha \mid \gamma\right)$ regardless $p\left(\gamma\right)$ as far as these do not connect only input units. In this sense, Eq. 4.72 becomes

$$\sum_{\alpha,\gamma} \left(\sum_{\sigma} J_{\sigma}^{(n)} \prod S_{\rho \in \alpha,\gamma} \right) \prod S_{\varrho \in \alpha,\gamma} = \sum_{\alpha,\gamma} \ln p\left(\alpha \mid \gamma\right) \prod S_{\varrho \in \alpha,\gamma} \quad , \tag{4.73}$$

for any weight that is not limited to connecting only input units. Thus, the probability distribution $p(\gamma)$ does not need to be known.

Finally, we discuss the relationship between Eq. 4.73 and the Kullback-Leibler distance minimization used in the Boltzmann Machine learning process. This is

$$G = \sum_{\alpha,\gamma} r(\alpha \mid \gamma) \ln\left(\frac{r(\alpha \mid \gamma)}{p(\alpha \mid \gamma)}\right) \ ,$$

where $r(\alpha \mid \gamma)$ is the probability distribution that we want the neural network to learn and $p(\alpha \mid \gamma)$ is the current response, due to its weights and units. The derivative of this expression is the standard learning equation for the BM and reads

$$\frac{\partial G}{\partial J_{\sigma}^{(n)}} = \frac{1}{T}\left(\left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle^{*} - \left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle\right) \ .$$

We assume now that the system has learned a given probability distribution, and there is no further variation on its weights. At this point, $\frac{\partial G}{\partial J_{\sigma}^{(n)}} = 0$ and

$$\left(\left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle^{*} - \left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle\right) = 0 \ ,$$

we now get the following equality

$$\begin{aligned} \left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle^{*} &= \left\langle \prod_{\rho\in\sigma} S_{\rho} \right\rangle \ , \\ \sum_{\alpha,\gamma} r(\alpha,\gamma) \prod S_{\varrho\in\alpha,\gamma} &= \sum_{\alpha,\gamma} p(\alpha,\gamma) \prod S_{\varrho\in\alpha,\gamma} \ , \end{aligned} \tag{4.74}$$

where we define the Boltzmann probability $p(\alpha,\gamma)$ as

$$p(\alpha,\gamma) = \mathrm{e}^{J^{(0)} + \sum J_{i_1}^{(1)} S_{i_1} + \sum J_{i_1 i_2}^{(2)} S_{i_1} S_{i_2} + \sum J_{i_1 i_2 i_3}^{(3)} S_{i_1} S_{i_2} S_{i_3} + \ldots} \ , \tag{4.75}$$

considering that the partition function $\mathcal{Z}$ stands for

$$\ln \mathcal{Z} = -J^{0} \ , \tag{4.76}$$

thus, it is already included into the equation. Since $r\left(\alpha,\gamma\right)$ are the target values to learn, when this system is written down in matricial notation one can proceed as follows

$$\begin{aligned}
\vec{r}\left(\alpha,\gamma\right)\cdot H_p &= \vec{p}\left(\alpha,\gamma\right)\cdot H_p \ , \\
\vec{r}\left(\alpha,\gamma\right)\cdot H_p\cdot H_p &= \vec{p}\left(\alpha,\gamma\right)\cdot H_p\cdot H_p \ , \\
\vec{r}\left(\alpha,\gamma\right) &= \vec{p}\left(\alpha,\gamma\right) \ , \\
\ln\vec{r}\left(\alpha,\gamma\right) &= \ln\vec{p}\left(\alpha,\gamma\right) \ , \\
\ln\vec{r}\left(\alpha,\gamma\right) &= J^{(0)}+\sum J_{i_1}^{(1)}S_{i_1}+\sum J_{i_1}^{(2)}i_2S_{i_1}S_{i_2}+\ldots \ ,
\end{aligned} \tag{4.77}$$

which matches *exactly* the backwards problem, whether it is the incomplete or the complete case

$$\ln\vec{r}\left(\alpha,\gamma\right)=H_p\cdot\vec{J} \ , \tag{4.78}$$

thus, proving that this is one correct solution from the multiple set of solutions that may be taken. Notice then that the difference between applying LU factorization and the Kullback-Leibler standard learning algorithm is the way in which they explore the space of solutions in the space: the LU method finds a solution that is not moving through such distance.

4.5.2 The priority encoder problem

We now solve a toy problem that is often used to test the ability of a given learning algorithm: the priority encoder is generated through an arithmetic boolean function that can be defined for any given set of bits. We define a sequence $\vec{X}$ of n_i bits, ordered as

$$\vec{X}=\{x_{n_i},x_{n_1-1},\ldots,x_2,x_1\} \ , \tag{4.79}$$

where the Most Significant Bit (MSB) is in the n_i position and the Least Significant Bit stands for x_1. The output units are activated representing a decimal value that matches the most significant bit activated in the input units, minus 1; hence there are as many outputs y as $log_2 n_i$. We can see standard priority encoder for 4 input units in Table 4.4; notice that input units with ? value stand for *don't care* values.

x_4	x_3	x_2	x_1	y_2	y_1
0	0	0	0	?	?
0	0	0	1	0	0
0	0	1	?	0	1
0	1	?	?	1	0
1	?	?	?	1	1

Table 4.4: Priority encoder truth table.

Since the output units will take their given value regardless ? input values, the boolean function for this encoder becomes

$$\begin{aligned} y_1 &= x_4 + \overline{x}_4\overline{x}_3x_2 \ , \\ y_2 &= x_4 + x_3 \ , \end{aligned} \tag{4.80}$$

which can be generalized for any number of input units. Notice that this is not an exhaustive problem, since there is an input combination whose output result is not cared for. In order not to get an exhaustive problem that we already know that the BM can learn by using Hadamard matrices, we add some noise bits that do not contribute to the output function; in our case we have added up to 5 noisy bits, as seen in Table 4.5, and disregarded the input pattern that produces an unknown combination of output units.

x_9	x_8	x_7	y_6	y_5	x_4	x_3	x_2	x_1	y_2	y_1
?	?	?	?	?	0	0	0	1	0	0
?	?	?	?	?	0	0	1	?	0	1
?	?	?	?	?	0	1	?	?	1	0
?	?	?	?	?	1	?	?	?	1	1

Table 4.5: Noisy priority encoder truth table.

Notice however that this input values produce up to $2^9 - 2^5 = 480$ combinations, though the real information is contained in $2^4 = 16$ input vectors. However, and since there are 9 different inputs, the neural network would not be able to generalize under a

given number of combinations. The benchmark that we have carried out to the neural network considers only removing random instances of the learning set, hence the neural network will have to figure out the function from Eq. 4.80 by removing the unnecessary additional information. Table 4.6 reports the efficiency on recognizing the full dataset; this values are computed as a mean over 1000 learning and test repetitions where the excluded learning vectors are randomly selected.

Instances removed	Efficiency
48 (10%)	100.0%
96 (20%)	100.0%
144 (30%)	100.0%
150 (32%)	99.8%

Table 4.6: Noisy priority encoder truth table.

Since the test is carried out by using LU factorization, the BM that is being used at this point is a high order Boltzmann Machine with no hidden units that may have any available connection between its units. Notice that the performance of the neural network decreases when 150 instances are removed. Starting at this point, the efficiency of the BM decreases almost lineally with every new vector that is removed from the training set.

The LU method can be used to effectively carry out a learning process on a HOBM. However, this algorithm is included in the same version from Ref. [The Mathworks^{TM}, 2008b], and it is stated to always select the same degrees of freedom from the whole set. In this sense, the system is giving values to a set of weights that do actually meet the desired learning distribution. However, these weights are not able to generalize the function that the BM is expected to learn, this does then cause the performance to drop. A possible solution to this issue would be to let the system change the weights that it selects as degrees of freedom, though the criteria that should be followed in this sense is yet unknown.

Chapter 5

Analytical learning process for a BM

5.1 Introduction

In this chapter we present a method that, through the decimation equations and the backwards problem, is used to build a second order Boltzmann Machine that can learn any given pattern. This structure is used to fix the number of hidden units and connections between the neurons of the BM that effectively solves the problem at hand. However, this process is not intended for building a BM that solves a so called *real-life problem*. Instead, it has been conceived as a way to provide the size and topology of a BM that can solve it.

This chapter is divided in two sections: in the first one we analyze this method, from the simplest BM that emulates basic logical, Boole based operations to a more complex neural network with any number of input and output neurons, and that is able to learn any given probability distribution. In this section we discuss the numerical error that the process introduces and show that it can become negligible. The chapter is then concluded with some practical examples on how to build these systems, where we calculate all the weights required to build these systems.

5.2 Boole arithmetic representation on a BM

In this section, we propose a numerical method that can be used to create a second order BM that is able to learn any probability distribution. The method can be used to set the size of the neural network that is able to solve a problem at hand, but it is not intended to be used as a solution to *real-life problems* nor a commercial application.

In the first part of this section we propose a simple method to build a second order BM that is able to reproduce simple logical gates, which are defined as AND, OR, NAND and NOR [Ercegovac et al., 1998], and the modifications that one can apply to make such systems learn a probability output distribution other than closer to 0 and 1 for a given input vector. The second part of this section discusses how this procedure is used to implement more complex boolean systems. We then proceed with the extension of this method to the case of an exhaustive probability distribution, where all the combinations of any number of output and input units are defined in the learning pattern, and associated to a certain probability distribution. We conclude this section with a discussion of the numerical error that is introduced when one uses this method to build a BM, and prove it to become negligible.

5.2.1 Basic logic operations

The BM is a stochastic neural network that can not learn exact zero nor one probabilities. Thus, one has to decide how the values that define a Boole based or a classifying problem are represented in terms of a BM. In this sense, consider the two inputs AND operation shown in table 5.1, which has been expressed in terms of this neural network (hence 0 and 1 values are depicted as -1 and 1) for a set of valid probability values. However, for the sake of simplicity, we will consider that any probability greater than 0.99 is treated as being effectively 1.0, and that any probability smaller than 0.01 is treated as 0.0: we are not willing the neural network to learn and extrapolate a probability distribution, we want it to represent a boolean function. We could use values greater than 0.99 and

smaller than 0.01 but, as we will later see, this would just mean higher values associated to the weights. In this sense, we would regard $p\left(S_o|S_{i_1},S_{i_2}\right)_2$ as a better approximation to an AND gate than $p\left(S_o|S_{i_1},S_{i_2}\right)_1$ is, and we would consider $p\left(S_o|S_{i_1},S_{i_2}\right)_3$ as the best (of the subset $p\left(S_o|S_{i_1},S_{i_2}\right)_1$, $p\left(S_o|S_{i_1},S_{i_2}\right)_2$ and $p\left(S_o|S_{i_1},S_{i_2}\right)_3$) representation that a BM can achieve.

S_{i_1}	S_{i_2}	S_o	$p\left(S_o\|S_{i_1},S_{i_2}\right)_1$	$p\left(S_o\|S_{i_1},S_{i_2}\right)_2$	$p\left(S_o\|S_{i_1},S_{i_2}\right)_3$
-1	-1	-1	0.7	0.9	$p > 0.99$
-1	-1	1	0.3	0.1	$p < 0.01$
-1	1	-1	0.7	0.9	$p > 0.99$
-1	1	1	0.3	0.1	$p < 0.01$
1	-1	-1	0.7	0.9	$p > 0.99$
1	-1	1	0.3	0.1	$p < 0.01$
1	1	-1	0.3	0.1	$p < 0.01$
1	1	1	0.7	0.9	$p > 0.99$

Table 5.1: Example of three different probability distributions that one can use to represent an AND operation with a BM.

We now show in table 5.2 how one can represent any of the basic logical operations AND, OR, NAND and NOR with a small BM. Notice however that both XOR and NXOR are not considered as basic here because

$$\begin{aligned} a \oplus b &= a\bar{b} + \bar{a}b \ , \\ \overline{a \oplus b} &= ab + \bar{a}\bar{b} \ . \end{aligned} \tag{5.1}$$

The structure that we consider in this section is a simple one with two input units S_{i_1} and S_{i_2} and an output neuron S_o, as depicted in Fig. 5.1. It has only two weights and a bias term.

As an example, we now design an AND gate with the BM shown in Fig. 5.1. We will consider that the weights of the system are set as

$$\left|w_o^{(1)}\right| = \left|w_{i_1 o}^{(2)}\right| = \left|w_{i_2 o}^{(2)}\right| = w \ , \tag{5.2}$$

S_{i_1}	S_{i_2}	S_o	p_{AND}	p_{OR}	p_{NAND}	p_{NOR}
-1	-1	-1	$p > 0.99$	$p > 0.99$	$p < 0.01$	$p < 0.01$
-1	-1	1	$p < 0.01$	$p < 0.01$	$p > 0.99$	$p > 0.99$
-1	1	-1	$p > 0.99$	$p < 0.01$	$p > 0.99$	$p < 0.01$
-1	1	1	$p < 0.01$	$p > 0.99$	$p < 0.01$	$p > 0.99$
1	-1	-1	$p > 0.99$	$p < 0.01$	$p > 0.99$	$p < 0.01$
1	-1	1	$p < 0.01$	$p > 0.99$	$p < 0.01$	$p > 0.99$
1	1	-1	$p < 0.01$	$p < 0.01$	$p > 0.99$	$p > 0.99$
1	1	1	$p > 0.99$	$p > 0.99$	$p < 0.01$	$p < 0.01$

Table 5.2: Basic boolean operations represented with a BM.

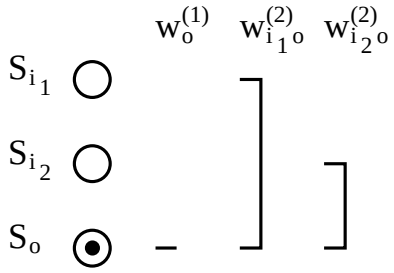

Figure 5.1: Two input units neural network with an output unit.

being $w \in \Re^+$ an arbitrary value that we will show how to calculate by using standard decimation process. We consider the weights to be temperature normalized

$$J_\sigma^{(n)} = \frac{w_\sigma^{(n)}}{T} \ . \tag{5.3}$$

Thus

$$\left|J_o^{(1)}\right| = \left|J_{i_1 o}^{(2)}\right| = \left|J_{i_2 o}^{(2)}\right| = J \ , \tag{5.4}$$

and using the definition of mean value for a single units from Eq. 3.52 and standard parallel association (the clamped input units are associated with the bias term of the

neural network) to obtain

$$\begin{aligned}\langle S_o\rangle &= \tanh\left(\frac{w_o^{(1)}}{T}+\frac{w_{i_1o}^{(2)}}{T}S_{i_1}+\frac{w_{i_2o}^{(2)}}{T}S_{i_2}\right)\\ &= \tanh\left(J_o^{(1)}+J_{i_1o}^{(2)}S_{i_1}+J_{i_2o}^{(2)}S_{i_2}\right)\ ,\end{aligned}\tag{5.5}$$

where we can establish a relationship between $\langle S_o\rangle$ and the probability of the output unit $p\left(S_o=1\right)$ by using this expression

$$\begin{aligned}\langle S_o\rangle &= p\left(S_o=1\right)-p\left(S_o=-1\right)\\ &= p\left(S_o=1\right)-\left(1-p\left(S_o=1\right)\right)\\ \langle S_o\rangle &= 2p\left(S_o=1\right)-1\ ,\end{aligned}\tag{5.6}$$

$$p\left(S_o=1\right)=\frac{\langle S_o\rangle+1}{2}\ .\tag{5.7}$$

S_{i_1}	S_{i_2}	$\langle S_o\rangle_j$	$p\left(S_o=1\right)$
-1	-1	$\langle S_o\rangle_0 < -0.98$	0.01
-1	1	$\langle S_o\rangle_1 < -0.98$	0.01
1	-1	$\langle S_o\rangle_2 < -0.98$	0.01
1	1	$\langle S_o\rangle_3 > 0.98$	0.99

Table 5.3: AND gate expected output value.

We now analyze the values of the weights for the AND case, assuming that the values that we want to obtain are those shown in table 5.3, and are found by using Eq. 5.5. Notice that the label j of the term $\langle S_o\rangle_j$ in this table is given by the numerical value of the binary sequence.

This process is done by checking the mean value of the output unit for each possible different set of input values, and assigning the proper value to the sign of the weight. We choose

$$\begin{aligned}J_{i_1o}^{(2)} &= J_{i_2o}^{(2)} = J\ ,\\ J_o^{(1)} &= -J\ ,\end{aligned}\tag{5.8}$$

S_{i_1}	S_{i_2}	$\langle S_o \rangle_{OR}$	p	$\langle S_o \rangle_{NAND}$	p	$\langle S_o \rangle_{NOR}$	p
-1	-1	$\langle S_o \rangle < -0.98$	0.01	$\langle S_o \rangle > 0.98$	0.99	$\langle S_o \rangle > 0.98$	0.99
-1	1	$\langle S_o \rangle > 0.98$	0.99	$\langle S_o \rangle > 0.98$	0.99	$\langle S_o \rangle < -0.98$	0.01
1	-1	$\langle S_o \rangle > 0.98$	0.99	$\langle S_o \rangle > 0.98$	0.99	$\langle S_o \rangle < -0.98$	0.01
1	1	$\langle S_o \rangle > 0.98$	0.99	$\langle S_o \rangle < -0.98$	0.01	$\langle S_o \rangle < -0.98$	0.01

Table 5.4: Basic boolean operations represented with the mean value and the probability $p\left(S_o = 1\right)$ of the output unit on a BM.

thus

$$\begin{aligned}
S_{i_1} = -1 \;\; S_{i_2} = -1 \quad & \langle S_o \rangle_0 = \tanh\left(J_o^{(1)} - J_{i_1 o}^{(2)} - J_{i_2 o}^{(2)}\right) \\
& = \tanh\left(-J - J - J\right) = \tanh\left(-3J\right) \;\; , \\
S_{i_1} = -1 \;\; S_{i_2} = 1 \quad & \langle S_o \rangle_1 = \tanh\left(-J - J + J\right) = \tanh\left(-J\right) \;\; , \\
S_{i_1} = 1 \;\; S_{i_2} = -1 \quad & \langle S_o \rangle_2 = \tanh\left(-J + J - J\right) = \tanh\left(-J\right) \;\; , \\
S_{i_1} = 1 \;\; S_{i_2} = 1 \quad & \langle S_o \rangle_3 = \tanh\left(-J + J + J\right) = \tanh\left(J\right) \;\; , \qquad (5.9)
\end{aligned}$$

where we have the constraint $\tanh\left(J\right) > 0.98$, which is solved as

$$J > \operatorname{atanh}\left(0.98\right) = 2.2976 \;\; . \qquad (5.10)$$

Weight	OR	NAND	NOR
$J_o^{(1)}$	J	J	$-J$
$J_{i_1 o}^{(2)}$	J	$-J$	$-J$
$J_{1_2 o}^{(2)}$	J	$-J$	$-J$

Table 5.5: Weights needed to build the Basic boolean operations.

This operation can be repeated for the boolean operators OR, NAND and NOR, whose expected values and probability for the output unit are represented in table 5.4; notice that p stands for $p\left(S_o = 1\right)$. Starting from Eq. 5.4, if this process is carried out for the other logical operations, a proper choice for the weights is shown in table 5.5, considering again that $J > \operatorname{atanh}\left(0.98\right)$.

5.2.2 Extensions of the basic logic operations

We have already discussed how one can build a BM that reproduces a given logical gate with two inputs. This concept, however, can be extended to some different situations that we discuss in this section. We begin by building some basic logical gates where the number of inputs is increased, thus building an n-inputs logical operation. We then proceed by discussing the case where any input is negated, and conclude this section by analyzing what happens when one of the regular outputs of the gate is changed for any other value than 0 or 1 (asymptotic) probability. We will need this three tools to conclude the chapter with a structure that uses a combination of all them, and that will be able to learn any Boole based set of rules.

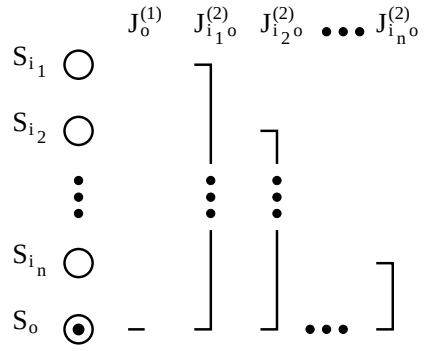

Figure 5.2: BM used to represent the n-input AND problem.

We now consider an n-input AND operation, which is represented by the neural network of Fig. 5.2, and described in table 5.6. Notice that the neural network is expected to carry out a simple operation, that is reduced to

$$\text{if } S_{i_1} = S_{i_2} = \ldots = S_{i_n} = 1 \text{ then } \langle S_o \rangle > 0.98 \ , \tag{5.11}$$

otherwise $\langle S_o \rangle < -0.98$. In this sense, we consider

$$\begin{aligned} J_{i_1 o}^{(2)} &= J_{i_2 o}^{(2)} = \ldots = J_{i_n o}^{(2)} = J \ , \\ J_o^{(1)} &= kJ \ , \ k \in \Re \ , \end{aligned} \tag{5.12}$$

where the unknown terms are both k and J. To solve this problem, one has to make sure that when all the inputs are active this condition is satisfied

$$\langle S_o \rangle = \tanh\left(kJ + nJ\right) > 0.98 \ . \tag{5.13}$$

S_{i_1}	S_{i_2}	$\ldots$	$S_{i_{n-1}}$	S_{i_n}	$\langle S_o \rangle$
-1	-1	$\ldots$	-1	-1	$\langle S_o \rangle_0 < -0.98$
-1	-1	$\ldots$	-1	1	$\langle S_o \rangle_1 < -0.98$
-1	-1	$\ldots$	1	-1	$\langle S_o \rangle_2 < -0.98$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
1	1	$\ldots$	-1	1	$\langle S_o \rangle_{2^n-2} < -0.98$
1	1	$\ldots$	1	-1	$\langle S_o \rangle_{2^n-1} < -0.98$
1	1	$\ldots$	1	1	$\langle S_o \rangle_{2^n} > 0.98$

Table 5.6: BM representation of the n-input AND operation.

On the other hand, any other input units value has to fulfill $\langle S_o \rangle < -0.98$. This happens when

$$\langle S_o \rangle = \tanh\left(kJ + (n-2)\,J\right) < -0.98 \ , \tag{5.14}$$

and the total contribution from the other neurons is at most $(n-2)\,J$. Any other combination of values from the input units will lead to a value smaller than this. We have to solve then the following system of inequations

$$\tanh\left((k+n)\,J\right) > 0.98 \ , \tag{5.15}$$

$$\tanh\left((k+n-2)\,J\right) < -0.98 \ . \tag{5.16}$$

A simple solution is to set $k = 1 - n$ and then

$$\tanh\left((1-n+n)\,J\right) = \tanh\left(J\right) > 0.98 \ , \tag{5.17}$$

$$\tanh\left(-J\right) < -0.98 \ , \tag{5.18}$$

and thus the final solution is achieved by solving $J > \text{atanh}\,(0.98)$, again. Consider now that any other n-input logic gate can be implemented by following the same concept; the solution to build n-input OR, NAND and NOR operations is given in table 5.7. We are representing $J^{(2)}_{ijo}$ as the weight connecting the j-nth input unit S_{i_j} with the output neuron S_o from this model.

Weight	OR	NAND	NOR
$J^{(1)}_o$	$(n-1)\,J$	$(n-1)\,J$	$(1-n)\,J$
$J^{(2)}_{ijo}$	J	$-J$	$-J$

Table 5.7: Weights $J^{(2)}_{ijo}$ for the input unit j in the n-input BM implementing OR, NAND and NOR operations.

The other possible variation to a given boolean operator is to negate an input. We show an example of a simple boolean operation where a given input is negated in table 5.8, notice however that this is still a variation of the basic AND operation that would be represented as

$$S_o = S_{i_1} \cdot \overline{S}_{i_2} \ . \tag{5.19}$$

S_{i_1}	S_{i_2}	$\overline{S}_{i_2}$	$S_{i_1} \cdot \overline{S}_{i_2}$	$\langle S_o \rangle$
-1	-1	1	-1	$\langle S_o \rangle_0 < -0.98$
-1	1	-1	-1	$\langle S_o \rangle_1 < -0.98$
1	-1	1	1	$\langle S_o \rangle_2 > 0.98$
1	1	-1	-1	$\langle S_o \rangle_3 < -0.98$

Table 5.8: $S_{i_1} \cdot \overline{S}_{i_2}$ operation with inputs S_{i_1}, S_{i_2} and output S_o.

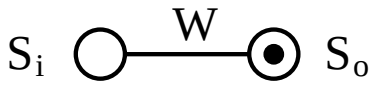

Figure 5.3: NOT gate structure.

We now show a way to build the NOT gate that is needed for this operation. We propose the structure depicted in Fig. 5.3 for a standard NOT gate, where the weight W is chosen such as $W \gg 1$. We discuss the equation for the expected value of S_o, which reads

$$\langle S_o \rangle = \tanh\left(-W S_i\right) \quad , \tag{5.20}$$

now we give values to S_i and assume $W \to \infty$, thus

$$\begin{aligned} S_i = 1 \qquad & \langle S_o \rangle = \tanh\left(-W\right) = -1 = -S_i \quad , \\ S_i = -1 \qquad & \langle S_o \rangle = \tanh\left(W\right) = 1 = -S_i \quad , \end{aligned} \tag{5.21}$$

so we use the structure from Fig. 5.4 to solve the negated AND, where $J^{(2)}_{i_1 o} = J$, $J^{(2)}_{i_2 h} = -W$, $J^{(2)}_{ho} = J$, $J^{(1)}_{o} = -J$ and $W \gg J$, for $J \in \Re^{+}$.

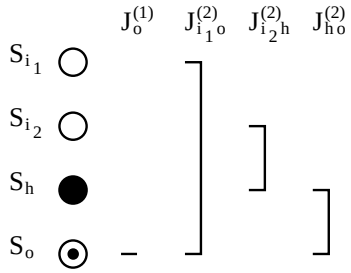

Figure 5.4: NOT gate applied to a BM with two input units.

Notice however that one can carry out serial association between W and J for an effective $J^{(2)}_{i_2 o}$ weight

$$J^{(2)}_{i_2 o} = \frac{1}{2} \ln \left(\frac{\cosh\left(J^{(2)}_{i_2 o} + J^{(2)}_{ho}\right)}{\cosh\left(J^{(2)}_{i_2 o} - J^{(2)}_{ho}\right)} \right) = \frac{1}{2} \ln \left(\frac{\cosh\left(J - W\right)}{\cosh\left(J + W\right)} \right) \quad . \tag{5.22}$$

Since $W \gg J$ we write this expression as

$$J^{(2)}_{i_2 o} = \frac{1}{2} \ln \left(\frac{\cosh\left(W - J\right)}{\cosh\left(W + J\right)} \right) \quad , \tag{5.23}$$

where it is possible to apply the following approximation

$$\begin{aligned} \lim_{W \to \infty} \ln\left(\cosh\left(W + J\right)\right) &= \lim_{W \to \infty} \ln\left(\mathrm{e}^{W-J} + \mathrm{e}^{-W+J}\right) - \ln 2 \\ &\simeq W - J - \ln 2 \quad , \end{aligned} \tag{5.24}$$

thus

$$\begin{aligned}\lim_{W\to\infty}\frac{1}{2}\ln\left(\frac{\cosh(W-J)}{\cosh(W+J)}\right) &= \lim_{W\to\infty}\frac{1}{2}\ln\left(\frac{\mathrm{e}^{W-J}+\mathrm{e}^{-W+J}}{\mathrm{e}^{W+J}+\mathrm{e}^{-W-J}}\right)\\ &\simeq \frac{1}{2}\ln\left(\frac{\mathrm{e}^{W-J}}{\mathrm{e}^{W+J}}\right)\\ &\simeq \frac{1}{2}\ln\left(\mathrm{e}^{-2J}\right)=-J\ ,\end{aligned} \tag{5.25}$$

so

$$J^{(2)}_{i_2o}=-J\ , \tag{5.26}$$

effectively resulting in a change of sign. In essence, the operation from Eq. 5.19 is simply carried out on the BM by changing the sign of the weight associated to the input unit

$$\begin{aligned}J^{(2)}_{i_1o}&=-J^{(2)}_{i_2o}=J\ ,\\ J^{(1)}_{o}&=-J\ .\end{aligned} \tag{5.27}$$

It is however easy to check that these weights implement such an operation

$$\begin{aligned}S_{i_1}=-1\quad S_{i_2}=-1\quad \langle S_o\rangle &= \tanh\left(J^{(1)}_o-J^{(2)}_{i_1o}+J^{(2)}_{i_2o}\right)\\ &= \tanh(-J-J+J)=\tanh(-J)\ ,\\ S_{i_1}=-1\quad S_{i_2}=1\quad \langle S_o\rangle &= \tanh(-J-J-J)=\tanh(-3J)\ ,\\ S_{i_1}=1\quad S_{i_2}=-1\quad \langle S_o\rangle &= \tanh(-J+J+J)=\tanh(J)\ ,\\ S_{i_1}=1\quad S_{i_2}=1\quad \langle S_o\rangle &= \tanh(-J+J-J)=\tanh(-J)\ ,\end{aligned} \tag{5.28}$$

where the condition to be satisfied is $\tanh(J)>0.98$ again. Therefore, we have shown that one can always use a BM to build any (probabilistic) AND, OR, NAND and NOR logical gate of an arbitrary number of inputs. It has also been shown then that, in general, one can generate a BM that reproduces the behavior of a certain logical operation.

We now analyze the situation where we want to introduce a mean value for the output unit other than $|\langle S_o\rangle|>0.98$. An example of this is shown in table 5.9 as a modified, two input AND operator.

S_{i_1}	S_{i_2}	$\langle S_o \rangle$
-1	-1	$\langle S_o \rangle_0 < -0.98$
-1	1	$\langle S_o \rangle_1 < -0.98$
1	-1	$\langle S_o \rangle_2 < -0.98$
1	1	$\langle S_o \rangle_3 \in (-1, +1)$

Table 5.9: AND gate with $\langle S_o \rangle_3$ taking any possible value within $(-1, +1)$.

The concept that describes best the process that one uses to build this AND gate is using a bias term H such that its magnitude is closer to J, thus $|H| \sim |J|$, but with some little added value d that ensures $|H| - |J| = d$. According to this idea, we give values to the weights of the system

$$\begin{aligned} J^{(2)}_{i_1 o} &= J^{(2)}_{i_2 o} = J \ , \\ J^{(1)}_{o} &= -2J + d \ , \end{aligned} \tag{5.29}$$

where $d \in \Re$ is an arbitrary constant whose value is found by calculating the mean value of the output unit for each different input set

$$\begin{aligned} S_{i_1} = -1 \quad S_{i_2} = -1 \quad & \langle S_o \rangle_0 = \tanh\left(J^{(1)}_o - J^{(2)}_{i_1 o} - J^{(2)}_{i_2 o} \right) \\ & = \tanh\left(-2J + d - J - J\right) = \tanh\left(-4J + d\right) \ , \\ S_{i_1} = -1 \quad S_{i_2} = 1 \quad & \langle S_o \rangle_1 = \tanh\left(-2J + d - J + J\right) = \tanh\left(-2J + d\right) \ , \\ S_{i_1} = 1 \quad S_{i_2} = -1 \quad & \langle S_o \rangle_2 = \tanh\left(-2J + d + J - J\right) = \tanh\left(-2J + d\right) \ , \\ S_{i_1} = 1 \quad S_{i_2} = 1 \quad & \langle S_o \rangle_3 = \tanh\left(-2J + d + J + J\right) = \tanh\left(d\right) \ , \end{aligned} \tag{5.30}$$

the actual value of d becomes then

$$d = \operatorname{atanh}\left(\langle S_o \rangle_3\right) \ . \tag{5.31}$$

Notice that this process can be reproduced with the other basic boolean operators, as reported in table 5.10. The values of the weights associated to the basic OR, NAND and NOR systems are shown in table 5.11, where $d \in \Re$ is introduced again as a parameter that has to be tuned.

S_{i_1}	S_{i_2}	$\langle S_o \rangle_{OR}$	$\langle S_o \rangle_{NAND}$	$\langle S_o \rangle_{NOR}$
-1	-1	$\langle S_o \rangle_0 \in (-1,+1)$	$\langle S_o \rangle_0 > 0.98$	$\langle S_o \rangle_0 \in (-1,+1)$
-1	1	$\langle S_o \rangle_1 > 0.98$	$\langle S_o \rangle_1 > 0.98$	$\langle S_o \rangle_1 < -0.98$
1	-1	$\langle S_o \rangle_2 > 0.98$	$\langle S_o \rangle_2 > 0.98$	$\langle S_o \rangle_2 < -0.98$
1	1	$\langle S_o \rangle_3 > 0.98$	$\langle S_o \rangle_3 \in (-1,+1)$	$\langle S_o \rangle_3 < -0.98$

Table 5.10: Variation to the expected values of the basic boolean operations.

Weight	OR	NAND	NOR
$J_o^{(1)}$	$2J+d$	$2J+d$	$-2J+d$
$J_{i_1 o}^{(2)}$	J	$-J$	$-J$
$J_{1_2 o}^{(2)}$	J	$-J$	$-J$

Table 5.11: Weights for the non-deterministic OR, NAND and NOR operations.

Notice also that, if one uses these values

$$\begin{aligned} J_{i_1 o}^{(2)} &= J_{i_2 o}^{(2)} = J \ , \\ J_o^{(1)} &= d \ , \end{aligned} \tag{5.32}$$

the behavior of the neural network is changed into

$$\begin{aligned} S_{i_1}=-1 \quad S_{i_2}=-1 \quad \langle S_o \rangle_0 &= \tanh\left(J_o^{(1)} - J_{i_1 o}^{(2)} - J_{i_2 o}^{(2)}\right) \\ &= \tanh\left(d-J-J\right) = \tanh\left(-2J+d\right) \ , \\ S_{i_1}=-1 \quad S_{i_2}=1 \quad \langle S_o \rangle_1 &= \tanh\left(d-J+J\right) = \tanh\left(d\right) \ , \\ S_{i_1}=1 \quad S_{i_2}=-1 \quad \langle S_o \rangle_1 &= \tanh\left(d+J-J\right) = \tanh\left(d\right) \ , \\ S_{i_1}=1 \quad S_{i_2}=1 \quad \langle S_o \rangle_2 &= \tanh\left(d+J+J\right) = \tanh\left(2J+d\right) \ , \end{aligned} \tag{5.33}$$

thus effectively having a probability distribution as described in table 5.12. However, this behavior is not useful to the discussion that is carried out in the next section, hence it will not be considered again: we will want the neural network to reproduce a given probability distribution due to only one input combination.

Notice however that the multiple input case and the negation of a given input can

S_{i_1}	S_{i_2}	$\langle S_o \rangle$
-1	-1	$\langle S_o \rangle_0 < -0.98$
-1	1	$\langle S_o \rangle_1 \in (-1, +1)$
1	-1	$\langle S_o \rangle_1 \in (-1, +1)$
1	1	$\langle S_o \rangle_2 > 0.98$

Table 5.12: Modification to the AND gate with $\langle S_o \rangle_1$ taking any possible value within $(-1, +1)$.

be used in the same model. In essence, this process can be applied to create a n-input operator AND, OR, NAND, NOR operator with $n \geq 2$. The values of the corresponding weights $J^{(2)}_{ij_o}$ connecting any input and output units are shown in table 5.13. Bear in mind, though, that by using this method, only one input pattern can be given a probability significantly different from 0 or 1.

Weight	AND	OR	NAND	NOR
$J^{(1)}_o$	$-nJ + d$	$nJ + d$	$nJ + d$	$-nJ + d$
$J^{(2)}_{ij_o}$	J	J	$-J$	$-J$

Table 5.13: Weights for the non-deterministic, n-inputs OR, NAND and NOR operations.

We finally propose an example to illustrate how a BM with this structure is built. We will use the topology depicted in Fig. 5.5 to build a different logical gate implementing $f = S_{i_1} + \overline{S}_{i_2} + S_{i_3}$, that is represented in table 5.14; we will also change the probability of the output due to input vector $S_{i_1} = -1$, $S_{i_2} = 1$ and $S_{i_3} = -1$.

It has been shown that the values of the weights of the system for an n-input OR operation with an output other than $|\langle S_o \rangle| > 0.98$ are

$$\begin{aligned} J^{(2)}_{i_1 o} &= J^{(2)}_{i_2 o} = J^{(2)}_{i_3 o} = J \ , \\ J^{(1)}_o &= nJ + d \ , \end{aligned} \tag{5.34}$$

S_{i_1}	S_{i_2}	$\overline{S}_{i_2}$	S_{i_3}	$\langle S_o \rangle$
-1	-1	1	-1	$\langle S_o \rangle_0 > 0.98$
-1	-1	1	1	$\langle S_o \rangle_1 > 0.98$
-1	1	-1	-1	$\langle S_o \rangle_2 = 0.3$
-1	1	-1	1	$\langle S_o \rangle_3 > 0.98$
1	-1	1	-1	$\langle S_o \rangle_4 > 0.98$
1	-1	1	1	$\langle S_o \rangle_5 > 0.98$
1	1	-1	-1	$\langle S_o \rangle_6 > 0.98$
1	1	-1	1	$\langle S_o \rangle_7 > 0.98$

Table 5.14: Table for the 3 input example.

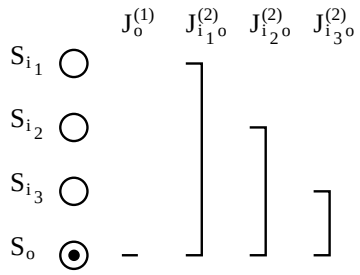

Figure 5.5: BM used to build the 3-input example.

we apply these values to our example, considering that $n = 3$ and S_{i_2} is negated

$$\begin{aligned} J^{(2)}_{i_1 o} &= -J^{(2)}_{i_2 o} = J^{(2)}_{i_3 o} = J \ , \\ J^{(1)}_{o} &= 3J + d \ , \end{aligned} \tag{5.35}$$

and we calculate the value for d. We proceed by analyzing the case where $\langle S_o \rangle_2 = 0.3$, because this is when all the values save to d are canceled

$$\langle S_o \rangle_2 = \tanh\left(J^{(1)}_o - J^{(2)}_{i_1 o} - J^{(2)}_{i_2 o} - J^{(2)}_{i_3 o} \right) = \tanh\left(3J + d - J - J - J\right) = \tanh\left(d\right) \ . \tag{5.36}$$

In consequence, d is fixed as $d = \operatorname{atanh}(0.3) = 0.3095$. We now find the values of J such that $\langle S_o \rangle_b < 0.98$ in any other case, this is, $b \neq 2$. If we consider all the possible

S_{i_1}	S_{i_2}	$\overline{S}_{i_2}$	S_{i_3}	$\langle S_o \rangle$
-1	-1	1	-1	$\langle S_o \rangle_0 = 0.9805 > 0.98$
-1	-1	1	1	$\langle S_o \rangle_1 = 0.9996 > 0.98$
-1	1	-1	-1	$\langle S_o \rangle_2 = 0.3000$
-1	1	-1	1	$\langle S_o \rangle_3 = 0.9805 > 0.98$
1	-1	1	-1	$\langle S_o \rangle_4 = 0.9996 > 0.98$
1	-1	1	1	$\langle S_o \rangle_5 = 1.0000 > 0.98$
1	1	-1	-1	$\langle S_o \rangle_6 = 0.9805 > 0.98$
1	1	-1	1	$\langle S_o \rangle_7 = 0.9996 > 0.98$

Table 5.15: Results for the three input OR problem.

values for the input units and the previous equation, we have the following possibilities

$$\langle S_o \rangle = \tanh(2J + d) > 0.98 \ , \tag{5.37}$$

$$\langle S_o \rangle = \tanh(4J + d) > 0.98 \ , \tag{5.38}$$

$$\langle S_o \rangle = \tanh(6J + d) > 0.98 \ , \tag{5.39}$$

being $\tanh(2J + d) > 0.98$ the most restrictive. Then

$$2J + 0.3095 > \operatorname{atanh}(0.98) = 2.2976 \ ,$$

$$J > 0.9940 \ . \tag{5.40}$$

We fix $J = 1.0$ and then we check these values to be correct in table 5.15.

5.2.3 Two stage logic operations

In the previous section, we discussed a method to build a BM that is able to reproduce a given n-inputs OR, AND, NAND, NOR operation. We have also discussed the possibility of adding noise to these gates, making one (and only one) input pattern to behave stochastically, with output probabilities appreciably different from 0 or 1. In this section we analyze the generic case for more complex Boole based operations, and propose a

method that one can use to build a second order BM where the complete input/output probability distribution is passed to the neural network.

S_{i_1}	S_{i_2}	S_{i_3}	$\langle S_o \rangle$
-1	-1	-1	$\langle S_o \rangle_0$
-1	-1	1	$\langle S_o \rangle_1$
-1	1	-1	$\langle S_o \rangle_2$
-1	1	1	$\langle S_o \rangle_3$
1	-1	-1	$\langle S_o \rangle_4$
1	-1	1	$\langle S_o \rangle_5$
1	1	-1	$\langle S_o \rangle_6$
1	1	1	$\langle S_o \rangle_7$

Table 5.16: Complete probability distribution for a three inputs BM, represented by using the expected values of the output neuron.

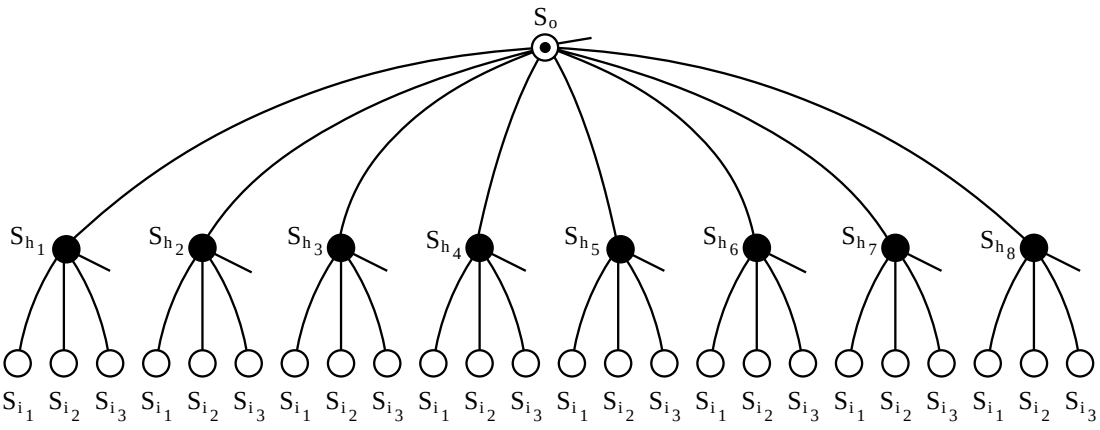

Figure 5.6: Three input, second order BM with inputs S_{i_1}, S_{i_2} and S_{i_3} and output S_o.

Consider now the general case of 3 input and 1 output units shown in table 5.16. The second order BM that we use to implement this operation is depicted in Fig. 5.6. The hidden units are expected to behave as separate AND gates, and one activates when a given input vector is passed to the input units. This is represented in table 5.17, where the function $f\left(S_o\right)_j$ stands for an unknown function that represents the behavior of the

hidden unit S_{h_j}, which becomes active due to a certain input vector. The other units remain inactive at a value close to -1 (here depicted as -1 for the sake of simplicity). On the other hand, the output unit behaves as an OR operation: if the expected values could certainly be -1 or $+1$, this neuron would behave as a real OR operator, becoming active due to a certain hidden unit activation. Since the active hidden unit S_{h_j} is having some expected value different from -1, this value is translated to the output unit; we will then adjust the weights of the system to reach the values shown in table 5.16. Notice however that this concept is exactly the same that is carried out to build large boolean operations [Ercegovac et al., 1998], and that we represent in Fig. 5.7.

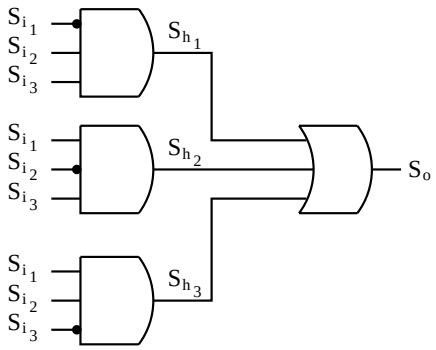

Figure 5.7: Standard digital implementation. Notice how the hidden units imitate the behavior of the intermediate AND gates.

Now we use decimation to analyze what happens on any AND gate, represented by a given hidden unit, when a certain input is used. As a matter of fact, we will consider that any $J_{ih}^{(n)} \gg J_{ho}^{(n)}$, for the same order $n = 1, 2$ in this, our BM. We will also consider that the AND operations that are made by the hidden units are stochastic with expected values assimptotically close to ± 1 as we did in the previous section, and that the weights have the values described there (see Eq. 5.8). Since we want each hidden unit to become active for each separate input vector, we need to set the weights at the values shown in table 5.18. In this table, we consider only one two-body weight $W \in \Re^+$, and eight different terms $d_{h_j} \ll W$, $d_{h_j} \in \Re$ that are added to the biases, just as we did in the previous section.

S_{i_1}	S_{i_2}	S_{i_3}	$\langle S_{h_1}\rangle$	$\langle S_{h_2}\rangle$	$\langle S_{h_3}\rangle$	$\langle S_{h_4}\rangle$	$\langle S_{h_5}\rangle$	$\langle S_{h_6}\rangle$	$\langle S_{h_7}\rangle$	$\langle S_{h_8}\rangle$
-1	-1	-1	$f(S_o)_1$	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	-1	$f(S_o)_2$	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	$f(S_o)_3$	-1	-1	-1	-1	-1
-1	1	1	-1	-1	-1	$f(S_o)_4$	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	$f(S_o)_5$	-1	-1	-1
1	-1	1	-1	-1	-1	-1	-1	$f(S_o)_6$	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	$f(S_o)_7$	-1
1	1	1	-1	-1	-1	-1	-1	-1	-1	$f(S_o)_8$

Table 5.17: Expected values for the hidden units of the system, when the unit is active it depends on the value that we want at the output neuron.

We begin the discussion of the behavior of the network by analyzing the first possible input value, which is $S_{i_1} = S_{i_2} = S_{i_3} = -1$. We associate all the weights connecting input units with their bias terms by using parallel association from decimation, and find a new set of bias terms $H_i^{(1)}$ that leads to the representation shown in Fig. 5.8.

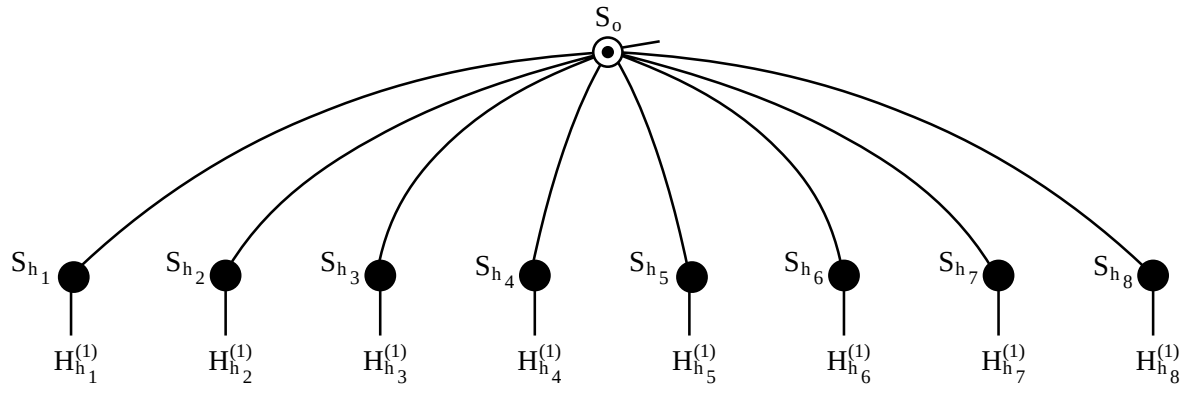

Figure 5.8: Parallel association with the input units and the bias terms.

	$J^{(2)}_{i_1h}$	$J^{(2)}_{i_2h}$	$J^{(2)}_{i_3h}$	$J^{(1)}_{h}$
S_{h_1}	$-W$	$-W$	$-W$	$-3W+d_{h_1}$
S_{h_2}	$-W$	$-W$	W	$-3W+d_{h_2}$
S_{h_3}	$-W$	W	$-W$	$-3W+d_{h_3}$
S_{h_4}	$-W$	W	W	$-3W+d_{h_4}$
S_{h_5}	W	$-W$	$-W$	$-3W+d_{h_5}$
S_{h_6}	W	$-W$	W	$-3W+d_{h_6}$
S_{h_7}	W	W	$-W$	$-3W+d_{h_7}$
S_{h_8}	W	W	W	$-3W+d_{h_8}$

Table 5.18: Weights connecting input and hidden units.

We reach then the following values

$$
\begin{aligned}
H^{(1)}_1 &= J^{(2)}_{i_1h_1}S_{i_1}+J^{(2)}_{i_2h_1}S_{i_2}+J^{(2)}_{i_3h_1}S_{i_3}+J^{(1)}_{h_1} = W+W+W-3W+d_{h_1}=d_{h_1}\ ,\\
H^{(1)}_2 &= W+W-W-3W+d_2=-2W+d_{h_2}\ ,\\
H^{(1)}_3 &= W-W+W-3W+d_3=-2W+d_{h_3}\ ,\\
H^{(1)}_4 &= W-W-W-3W+d_4=-4W+d_{h_4}\ ,\\
H^{(1)}_5 &= -W+W+W-3W+d_5=-2W+d_{h_5}\ ,\\
H^{(1)}_6 &= -W+W-W-3W+d_6=-4W+d_{h_6}\ ,\\
H^{(1)}_7 &= -W-W+W-3W+d_7=-4W+d_{h_7}\ ,\\
H^{(1)}_8 &= -W-W-W-3W+d_8=-6W+d_{h_8}\ .
\end{aligned}
\tag{5.41}
$$

Notice that if $|W| \gg d_{h_\alpha}$, $\forall\alpha$, one can consider that $H^{(1)}_j \to \infty$ except for $H^{(1)}_1$, being this one the only relevant contribution.

We now associate these weights with those connecting the hidden units with the output neuron by serial association. As we are performing an OR operation, the values of the weights $J^{(2)}_{ho}$ and $J^{(1)}_o$ are set to

$$
\begin{aligned}
&J^{(2)}_{h_1o}=J^{(2)}_{h_2o}=J^{(2)}_{h_3o}=J^{(2)}_{h_4o}=J^{(2)}_{h_5o}=J^{(2)}_{h_6o}=J^{(2)}_{h_7o}=J^{(2)}_{h_8o}=J\ ,\\
&J^{(1)}_o=7J\ ,
\end{aligned}
\tag{5.42}
$$

where $J \in \Re$ and $|J| \ll |W|$. We now calculate the serial association of $H_j^{(1)}$ and $J_{h_j o}^{(2)}$, thus producing new bias terms for S_o of the form $h_j^{(1)}$ as

$$h_j^{(1)} = \frac{1}{2} \ln \left(\frac{\cosh\left(J_{h_j o}^{(2)} + H_j^{(1)}\right)}{\cosh\left(J_{h_j o}^{(2)} - H_j^{(1)}\right)} \right) \ , \tag{5.43}$$

this process is shown in Fig. 5.9.

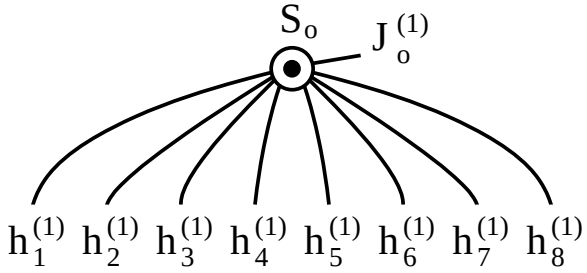

Figure 5.9: Serial association of the bias terms from the hidden units $H_j^{(1)}$ and the weights $J_{h_j o}^{(2)}$ connecting them with the output unit, resulting in the new $h_j^{(1)}$ connections.

For any $j \neq 1$, we are willing to achieve the following

$$h_j^{(1)} \simeq -J_{h_j o}^{(2)} = -J \ , \tag{5.44}$$

as with $J \gg 1$, the corresponding unit S_{h_j} will remain inactive; thus meaning that $\left\langle S_{h_j} \right\rangle \simeq -1$ and $p\left(S_{h_j} = 1\right) \simeq 0$. This can be done if W is big enough; in this case we assume $W \to \infty$ in

$$\begin{aligned} \lim_{W \to \infty} \ln\left(\cosh\left(J - nW + d_{h_j}\right)\right) &= \lim_{W \to \infty} \ln\left(\mathrm{e}^{-J+nW-d_{h_j}} + \mathrm{e}^{J-nW+d_{h_j}}\right) - \ln 2 \\ &\simeq \ln\left(\mathrm{e}^{nW-J-d_{h_j}}\right) - \ln 2 \\ &\simeq nW - J - d_{h_j} - \ln 2 \ , \end{aligned} \tag{5.45}$$

and then

$$\begin{aligned} h_j^{(1)} &= \frac{1}{2} \ln \left(\frac{\cosh\left(J_{h_j o}^{(2)} + H_j^{(1)}\right)}{\cosh\left(J_{h_j o}^{(2)} - H_j^{(1)}\right)} \right) = \frac{1}{2} \ln \left(\frac{\cosh\left(J - nW + d_{h_j}\right)}{\cosh\left(J + nW - d_{h_j}\right)} \right) \\ &= \ln \left(\frac{\mathrm{e}^{J-nW+d_{h_j}} + \mathrm{e}^{-J+nW-d_{h_j}}}{\mathrm{e}^{J+nW-d_{h_j}} + \mathrm{e}^{-J-nW+d_{h_j}}} \right) \simeq \frac{1}{2} \ln \left(\frac{\mathrm{e}^{-J+nW-d_{h_j}}}{\mathrm{e}^{J+nW-d_{h_j}}} \right) \\ &\simeq \frac{1}{2} \ln\left(\mathrm{e}^{-2J}\right) = -J \ , \end{aligned} \tag{5.46}$$

for $n = 2$, $n = 4$ and $n = 6$, which are the possible values that n can take. Then $h_j^{(1)} \simeq -J, \forall j \neq 1$. Since $J_o^{(1)} = 7J$, the parallel association of all the $h_j^{(1)}$ terms results in

$$\begin{aligned} h_o^{(1)} &= h_1^{(1)} + h_2^{(1)} + h_3^{(1)} + h_4^{(1)} + h_5^{(1)} + h_6^{(1)} + h_7^{(1)} + h_8^{(1)} + J_o^{(1)} \\ &= \frac{1}{2} \ln \left(\frac{\cosh\left(J + H_1^{(1)}\right)}{\cosh\left(J - H_1^{(1)}\right)} \right) - 7J + 7J \\ &= \frac{1}{2} \ln \left(\frac{\cosh\left(J + d_{h_1}\right)}{\cosh\left(J - d_{h_1}\right)} \right) \quad . \end{aligned} \tag{5.47}$$

We now assume $J \gg d_{h_j}$ and repeat the approximation

$$h_o^{(1)} = \lim_{J \to \infty} \frac{1}{2} \ln \left(\frac{\cosh\left(J + d_{h_1}\right)}{\cosh\left(J - d_{h_1}\right)} \right) = d_{h_1} \quad , \tag{5.48}$$

and therefore

$$\langle S_o \rangle_1 = \tanh\left(h_o^{(1)}\right) = \tanh d_{h_1} \quad , \tag{5.49}$$

hence the value of the output unit in this case depends only on d_{h_1}, provided that $W \gg J \gg d_{h_j}$.

In summary, we have seen that the expected value of the output unit depends on the value of the bias from the active hidden unit; but we have also shown that this topology can learn any problem involving three input and one output units. This same process can be carried out for a neural network with n_i input units and 1 output neuron, thus using up to $n_h = 2^{n_i}$ hidden units (it will cover all the possible combinations that the input units can take).

S_{i_1}	S_{i_2}	S_{i_3}	$\langle S_o \rangle$
-1	-1	-1	$\langle S_o \rangle_1$
-1	1	1	$\langle S_o \rangle_2$
1	1	-1	$\langle S_o \rangle_3$
1	1	1	$\langle S_o \rangle_4$

Table 5.19: Non-exhaustive probability distribution for a three inputs BM.

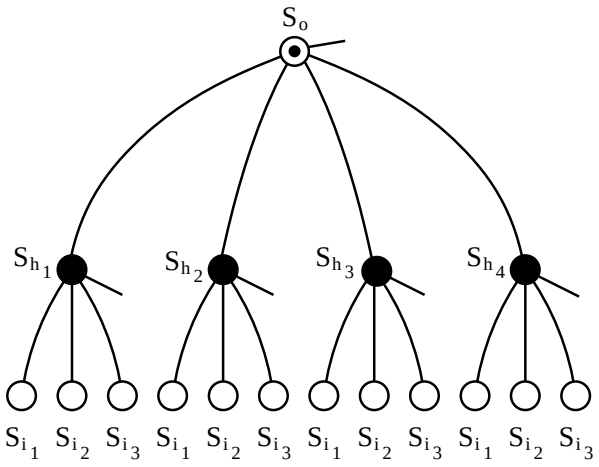

Figure 5.10: BM topology for the non-exhaustive probability distribution.

Notice however that it is not mandatory to provide an exhaustive probability distribution to the system: we could build the equivalent AND systems in the first stage of the BM to become active due to certain inputs, according to the probability distribution shown in table 5.19.

S_{i_1}	S_{i_2}	S_{i_3}	$\langle S_o \rangle$
-1	-1	-1	$\langle S_o \rangle_1$
-1	-1	1	-1
-1	1	-1	-1
-1	1	1	$\langle S_o \rangle_2$
1	-1	-1	-1
1	-1	1	-1
1	1	-1	$\langle S_o \rangle_3$
1	1	1	$\langle S_o \rangle_4$

Table 5.20: Real behavior of the smaller BM built with the Boolean equivalence. Even though it is not possible for the neural network to reach -1 values, the real result would be closer.

The topology for this neural network is shown in Fig. 5.10, the values of its connections

calculated by using the same method that has been described so far. In this sense, the weights resulting from the process would be the ones shown in table 5.21; though the bias term $J_o^{(1)}$ is changed into $J_o^{(1)} = 3J$, because we now consider the output unit to behave as a four input OR type gate. However, this BM would show a probability distribution as the one described in table 5.20, because the hidden units would stay only active as a response to some input values and remain inactive for the other ones, thus forcing the output unit to remain inactive.

	$J_{i_1h}^{(2)}$	$J_{i_2h}^{(2)}$	$J_{i_3h}^{(2)}$	$J_h^{(1)}$
S_{h_1}	$-W$	$-W$	$-W$	$-3W + d_{h_1}$
S_{h_2}	$-W$	W	W	$-3W + d_{h_2}$
S_{h_3}	W	W	$-W$	$-3W + d_{h_3}$
S_{h_4}	W	W	W	$-3W + d_{h_4}$

Table 5.21: Weights connecting input and hidden units for the non-exhaustive model.

5.2.4 System with two output units and several inputs

Now we consider the situation where the probability distribution of the output units is described in terms of both units, thus being a joint probability distribution. We propose a simple example of this problem in table 5.22, where a three input neural network with two output units is expected to learn a given probability distribution. Since it is hard to fit all the possible values that the units can take in the table, we have not written them all. Instead, they are represented in terms of the decimal values that correspond to the binary counting (where 0 value is replaced by -1) that the units can take as

$$p\left(S_{o_1}, S_{o_2} | S_{i_1}, S_{i_2}, S_{i_3}\right) = p\left(m_o | m_i\right) \quad . \tag{5.50}$$

Then, we would use

$$p\left(S_{o_1} = -1, S_{o_2} = -1 | S_{i_1} = -1, S_{i_2} = -1, S_{i_3} = -1\right) = p\left(0|0\right) \quad . \tag{5.51}$$

S_{i_1}	S_{i_2}	S_{i_3}	S_{o_1}	S_{o_2}	$p\left(S_{o_1}, S_{o_2} \vert S_{i_1}, S_{i_2}, S_{i_3}\right)$
-1	-1	-1	-1	-1	$p\left(0\vert 0\right)$
-1	-1	-1	-1	-1	$p\left(1\vert 0\right)$
-1	-1	-1	-1	-1	$p\left(2\vert 0\right)$
-1	-1	-1	-1	-1	$p\left(3\vert 0\right)$
-1	-1	-1	-1	-1	$p\left(0\vert 1\right)$
-1	-1	-1	-1	-1	$p\left(1\vert 1\right)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
-1	-1	-1	-1	-1	$p\left(1\vert 7\right)$
-1	-1	-1	-1	-1	$p\left(2\vert 7\right)$
-1	-1	-1	-1	-1	$p\left(3\vert 7\right)$

Table 5.22: Probability distribution for a three input units BM with two output neurons.

The second order structure that we will use to learn this probability distribution is shown in Fig. 5.11. This structure has 8 hidden units linked with the three inputs and separately connected to the output units, for a total of 16 hidden units. There are also 8 hidden units connecting both output units that are linked with the input ones, hence there are $16 + 8 = 24$ hidden neurons. All these hidden neurons will remain inactive by following the same principles that we described in the previous section: they work as simple logical gates that are activated only when the input vector is the desired one.

The weights that we use in the neural network are described in terms of the units they connect. The weights connecting all input units S_{i_1}, S_{i_2} and S_{i_3} to the hidden ones S_{h_j} are set as

$$\left|J^{(2)}_{i_1 h_j}\right| = \left|J^{(2)}_{i_2 h_j}\right| = \left|J^{(2)}_{i_3 h_j}\right| = W \ , \tag{5.52}$$

while the bias terms of the hidden units are

$$J^{(1)}_{h_j} = -3W + d_{h_j} \ , \ d_{h_j} \in \Re \ , \tag{5.53}$$

On the other hand, the hidden units are connected to the outputs S_{o_1} and S_{o_2} by the

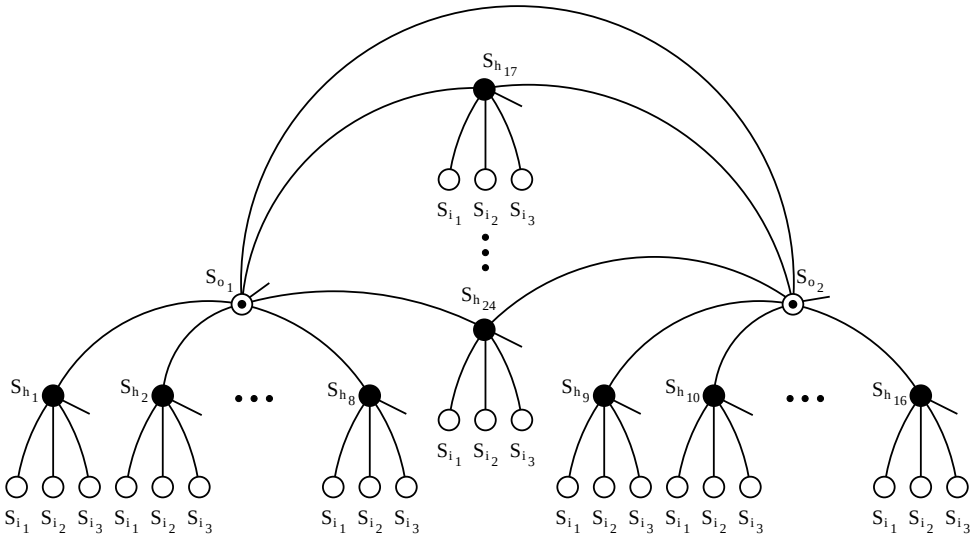

Figure 5.11: Sparsely connected BM with three input units and two outputs.

S_{i_1}	S_{i_2}	S_{i_3}	Active units
-1	-1	-1	S_{h_1}, S_{h_9}, $S_{h_{17}}$
-1	-1	1	S_{h_2}, $S_{h_{10}}$, $S_{h_{18}}$
-1	1	-1	S_{h_3}, $S_{h_{11}}$, $S_{h_{19}}$
-1	1	1	S_{h_4}, $S_{h_{12}}$, $S_{h_{20}}$
1	-1	-1	S_{h_5}, $S_{h_{13}}$, $S_{h_{21}}$
1	-1	1	S_{h_6}, $S_{h_{14}}$, $S_{h_{22}}$
1	1	-1	S_{h_7}, $S_{h_{15}}$, $S_{h_{23}}$
1	1	1	S_{h_8}, $S_{h_{16}}$, $S_{h_{24}}$

Table 5.23: Activation pattern for the hidden units.

weights

$$J^{(2)}_{h_j o_1} = J^{(2)}_{h_j o_2} = J \ , \tag{5.54}$$

consider however that the discussion from the previous sections does also apply here, and that the signs of the weights are placed depending on whether we want the hidden unit to become active due to a certain input value; it does also happen that $W \gg J \gg 1$ and that $W \gg J \gg \left|d_{h_j}\right|$. The hidden units that become active and the input that activates

them is shown in table 5.23, and the weights connecting these sets of units with the input ones are represented in table 5.24.

Set of hidden units connected to the inputs	S_{i_1}	S_{i_2}	S_{i_3}
S_{h_1}, S_{h_9}, $S_{h_{17}}$	$-J$	$-J$	$-J$
S_{h_2}, $S_{h_{10}}$, $S_{h_{18}}$	$-J$	$-J$	J
S_{h_3}, $S_{h_{11}}$, $S_{h_{19}}$	$-J$	J	$-J$
S_{h_4}, $S_{h_{12}}$, $S_{h_{20}}$	$-J$	J	J
S_{h_5}, $S_{h_{13}}$, $S_{h_{21}}$	J	$-J$	$-J$
S_{h_6}, $S_{h_{14}}$, $S_{h_{22}}$	J	$-J$	J
S_{h_7}, $S_{h_{15}}$, $S_{h_{23}}$	J	J	$-J$
S_{h_8}, $S_{h_{16}}$, $S_{h_{24}}$	J	J	J

Table 5.24: Weights connecting the hidden units with the input ones.

We begin the example by analyzing the first case, which is given by $S_{i_1} = S_{i_2} = S_{i_3} = -1$. Now we show that the parallel association of the bias terms and weights connecting the input and the hidden units will lead to the following values for the bias terms of units S_{h_1}, S_{h_9} and $S_{h_{17}}$

$$\begin{aligned}
J_{h_1}^{(1)} &= -J_{i_1h_1}^{(2)} - J_{i_2h_1}^{(2)} - J_{i_3h_1}^{(2)} + J_{h_1}^{(1)} = W + W + W - 3W + d_{h_1} = d_{h_1} \ , \\
J_{h_9}^{(1)} &= -J_{i_1h_9}^{(2)} - J_{i_2h_9}^{(2)} - J_{i_3h_9}^{(2)} + J_{h_9}^{(1)} = W + W + W - 3W + d_{h_9} = d_{h_9} \ , \\
J_{h_{17}}^{(1)} &= -J_{i_1h_{17}}^{(2)} - J_{i_2h_{17}}^{(2)} - J_{i_3h_{17}}^{(2)} + J_{h_{17}}^{(1)} = W + W + W - 3W + d_{h_{17}} = d_{h_{17}} \ .
\end{aligned} \tag{5.55}$$

The parallel association for the other, inactive, hidden units leads to

$$J_{h_j}^{(1)} = -J_{i_1h_j}^{(2)} - J_{i_2h_j}^{(2)} - J_{i_3h_j}^{(2)} + J_{h_j}^{(1)} = W - W - W - 3W + d_{h_j} = -4W + d_{h_j} \ , \tag{5.56}$$

for $j = 2, 3, 5, 10, 11, 13, 18, 19, 21$, while

$$J_{h_j}^{(1)} = -J_{i_1h_j}^{(2)} - J_{i_2h_j}^{(2)} - J_{i_3h_j}^{(2)} + J_{h_j}^{(1)} = W + W - W - 3W + d_{h_j} = -2W + d_{h_j} \ , \tag{5.57}$$

for $j = 4, 6, 7, 12, 14, 15, 20, 22, 23$, and

$$J_{h_j}^{(1)} = -J_{i_1 h_j}^{(2)} - J_{i_2 h_j}^{(2)} - J_{i_3 h_j}^{(2)} + J_{h_j}^{(1)} = -W - W - W - 3W + d_{h_j} = -6W + d_{h_j} \ , \tag{5.58}$$

for $j = 8, 16, 24$, thus arriving to the structure shown in Fig. 5.12.

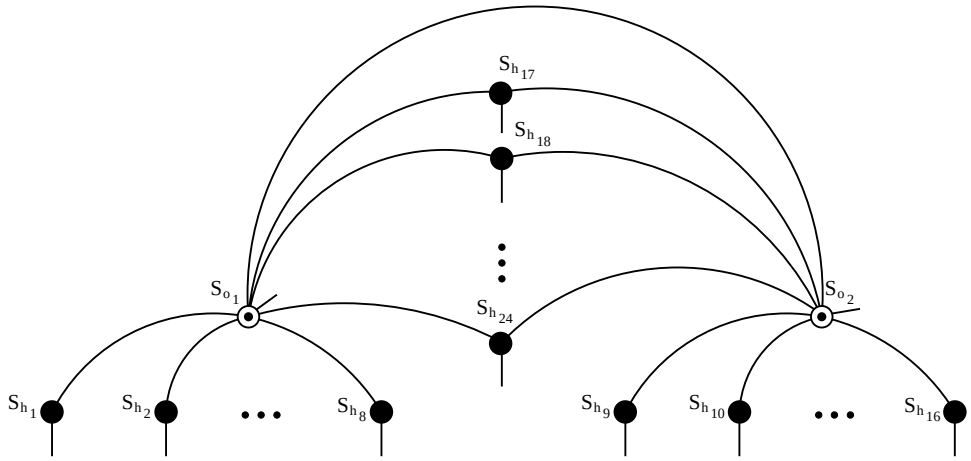

Figure 5.12: Parallel association of the weights connecting the input units and the bias terms from the hidden units.

Now we carry out serial association to suppress hidden units from S_{h_1} to $S_{h_{16}}$, thus associating weights $J_{h_j}^{(1)}$ with $J_{h_j o_k}^{(2)}$ for j from 1 to 16; the resulting model is shown in Fig. 5.13. We have split this process in two steps: in the the first one we decimate units S_{h_1} and S_{h_9}, which are the active ones, thus obtaining new set of biases $T_{o_1}^{(1)}$ and $T_{o_2}^{(1)}$ for the output neurons; we will then proceed with the remaining hidden units. This second step will produce some different bias terms, which will be represented as $H_{o_1}^{(1)}$ and $H_{o_2}^{(1)}$. Notice however that this process is the same that one would follow in order to build the two stage structure that was shown in the previous section. We find $T_{o_1}^{(1)}$ and $T_{o_2}^{(1)}$ as

$$T_{o_1}^{(1)} = \frac{1}{2} \ln \left(\frac{\cosh\left(J + d_{h_1}\right)}{\cosh\left(J - d_{h_1}\right)} \right) = d_{h_1} \ , \tag{5.59}$$

$$T_{o_2}^{(1)} = \frac{1}{2} \ln \left(\frac{\cosh\left(J + d_{h_9}\right)}{\cosh\left(J - d_{h_9}\right)} \right) = d_{h_9} \ . \tag{5.60}$$

We now associate the bias terms $J_{h_j}^{(1)}$ and the weights connecting the inactive hidden units $J_{h_j o_k}^{(2)}$ with the output neurons, to find $H_{o_1}^{(1)}$ and $H_{o_2}^{(1)}$. This process is done for j spanning

from 2 to 8 and from 10 to 16, and considers again that $W \gg J$ in the following equations

$$H_{o_1}^{(1)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J+nW-d_{h_j}\right)}{\cosh\left(J-nW+d_{h_j}\right)}\right) \ , \tag{5.61}$$

$$H_{o_2}^{(1)} = \frac{1}{4}\ln\left(\frac{\cosh\left(J+nW-d_{h_j}\right)}{\cosh\left(J-nW-d_{h_j}\right)}\right) \ , \tag{5.62}$$

where n stands for $n = -2, -4, -6$ depending on the value of j. We apply then the approximation from Eq 5.24, and thus

$$\begin{aligned} 4H_{o_1}^{(1)} &= |n|W + d_{h_j} - 2J - |n|W - d_{h_j} - 2J = -4J \ , \\ 4H_{o_2}^{(1)} &= |n|W + d_{h_j} - 2J - |n|W - d_{h_j} - 2J = -4J \ , \end{aligned} \tag{5.63}$$

so $H_{o_1}^{(1)} = H_{o_2}^{(1)} = -J$.

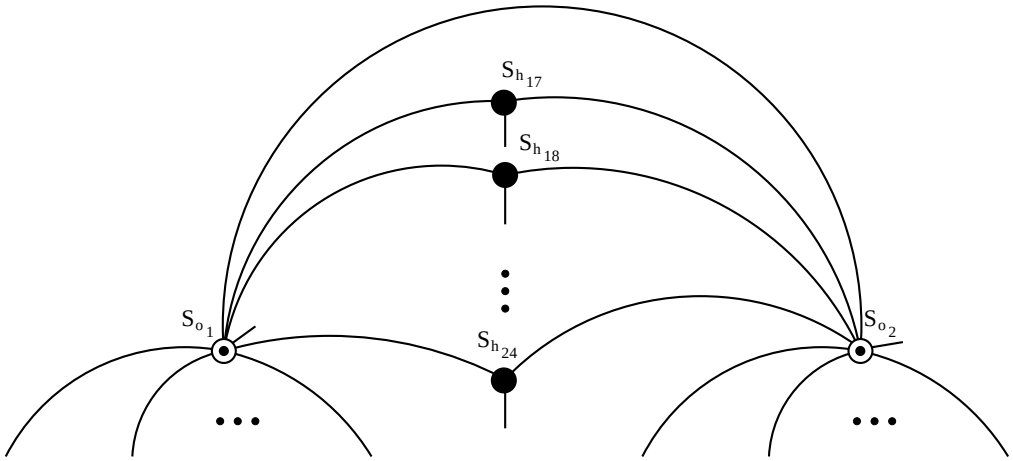

Figure 5.13: Parallel and serial association with the input units, the bias terms and the hidden units from units S_{h_1} to $S_{h_{16}}$.

Now we use star-triangle decimation from the basic decimation procedures to suppress unit $S_{h_{17}}$, which is connected to units S_{o_1} and S_{o_2} by weights $J_{h_{17}o_1}^{(2)}$ and $J_{h_{17}o_2}^{(2)}$ and has a bias term $J_{h_{17}}^{(1)}$. This operation will generate a second order weight $G_{o_1o_2}^{(2)}$ linking S_{o_1} with S_{o_2} and two new bias terms $G_{o_1}^{(1)}$, $G_{o_2}^{(1)}$ connected to these units. The equations that we

use are the following

$$G^{(2)}_{o_1o_2} = \frac{1}{4}\ln\left(\frac{\cosh\left(J^{(2)}_{h_{17}o_1} + J^{(2)}_{h_{17}o_2} - J^{(1)}_{h_{17}}\right)\cosh\left(J^{(2)}_{h_{17}o_1} + J^{(2)}_{h_{17}o_2} + J^{(1)}_{h_{17}}\right)}{\cosh\left(J^{(2)}_{h_{17}o_1} - J^{(2)}_{h_{17}o_2} + J^{(1)}_{h_{17}}\right)\cosh\left(J^{(2)}_{h_{17}o_1} - J^{(2)}_{h_{17}o_2} - J^{(1)}_{h_{17}}\right)}\right) \quad , \quad (5.64)$$

$$G^{(1)}_{o_1} = \frac{1}{4}\ln\left(\frac{\cosh\left(J^{(2)}_{h_{17}o_1} - J^{(2)}_{h_{17}o_2} + J^{(1)}_{h_{17}}\right)\cosh\left(J^{(2)}_{h_{17}o_1} + J^{(2)}_{h_{17}o_2} + J^{(1)}_{h_{17}}\right)}{\cosh\left(J^{(2)}_{h_{17}o_1} + J^{(2)}_{h_{17}o_2} - J^{(1)}_{h_{17}}\right)\cosh\left(J^{(2)}_{h_{17}o_1} - J^{(2)}_{h_{17}o_2} - J^{(1)}_{h_{17}}\right)}\right) \quad , \quad (5.65)$$

$$G^{(1)}_{o_2} = \frac{1}{4}\ln\left(\frac{\cosh\left(J^{(2)}_{h_{17}o_1} - J^{(2)}_{h_{17}o_2} - J^{(1)}_{h_{17}}\right)\cosh\left(J^{(2)}_{h_{17}o_1} + J^{(2)}_{h_{17}o_2} + J^{(1)}_{h_{17}}\right)}{\cosh\left(J^{(2)}_{h_{17}o_1} - J^{(2)}_{h_{17}o_2} + J^{(1)}_{h_{17}}\right)\cosh\left(J^{(2)}_{h_{17}o_1} + J^{(2)}_{h_{17}o_2} - J^{(1)}_{h_{17}}\right)}\right) \quad . \quad (5.66)$$

If we put values to these expressions we obtain

$$\begin{aligned} G^{(2)}_{o_1o_2} &= \frac{1}{4}\ln\left(\frac{\cosh\left(J + J - d_{h_{17}}\right)\cosh\left(J + J + d_{h_{17}}\right)}{\cosh\left(J - J + d_{h_{17}}\right)\cosh\left(J - J - d_{h_{17}}\right)}\right) \\ &= \frac{1}{4}\ln\left(\frac{\cosh\left(2J + d_{h_{17}}\right)\cosh\left(2J - d_{h_{17}}\right)}{\cosh^2\left(d_{h_{17}}\right)}\right) \quad , \quad (5.67) \\ G^{(1)}_{o_1} &= \frac{1}{4}\ln\left(\frac{\cosh\left(J + J + d_{h_{17}}\right)}{\cosh\left(J + J - d_{h_{17}}\right)}\right) \quad , \quad (5.68) \\ G^{(1)}_{o_2} &= \frac{1}{4}\ln\left(\frac{\cosh\left(J + J + d_{h_{17}}\right)}{\cosh\left(J + J - d_{h_{17}}\right)}\right) \quad , \quad (5.69) \end{aligned}$$

where we now analyze the values of the weights for $J \gg d_{h_{17}}$ and J is big enough to carry out the approximation from Eq. 5.24 again

$$\begin{aligned} &4G^{(2)}_{o_1o_2} \simeq 4J - 2\ln 2 - \ln\cosh^2\left(d_{h_{17}}\right) \quad , \\ &4G^{(1)}_{o_1} = 4G^{(1)}_{o_2} \simeq 2J + d_{h_{17}} - \ln 2 - 2J + d_{h_{17}} + \ln 2 = d_{h_{17}} \quad . \quad (5.70) \end{aligned}$$

We finally use star-triangle decimation to suppress the units that are connected to the output units, and that remain inactive; these are units $S_{h_{18}}$ to $S_{h_{24}}$. The resulting weights will be named as $H^{(2)}_{o_1o_2}$, $H^{(1)}_{o_1}$ and $H^{(1)}_{o_2}$, and stand for the second order weight that connect units S_{o_1} and S_{o_2} and their respective bias terms. Notice that these are the same names that we previously used when decimating the other inactive units: we will see that the

result is the same. We use the following equations

$$H^{(2)}_{o_1 o_2} = \frac{1}{4}\ln\left(\frac{\cosh\left(2J - nW + d_{h_j}\right)\cosh\left(2J + nW - d_{h_j}\right)}{\cosh\left(nW\right)\cosh\left(-nW + d_{h_j}\right)}\right) \tag{5.71}$$

$$H^{(1)}_{o_1} = \frac{1}{4}\ln\left(\frac{\cosh\left(2J + nW - d_{h_j}\right)}{\cosh\left(2J - nW + d_{h_j}\right)}\right) \ , \tag{5.72}$$

$$H^{(1)}_{o_2} = \frac{1}{4}\ln\left(\frac{\cosh\left(2J + nW - d_{h_j}\right)}{\cosh\left(2J - nW + d_{h_j}\right)}\right) \ , \tag{5.73}$$

where n stands again for $n = -2, -4, -6$ depending on the value of j. We apply then the approximation from Eq 5.24, and thus

$$\begin{aligned} 4H^{(2)}_{o_1 o_2} &= |n|W - 2J + d_{h_j} + |n|W + 2J + d_{h_j} - 2|n|W - d_{h_j} = 0 \ , \\ 4H^{(1)}_{o_1} &= |n|W - 2J + d_{h_j} - |n|W - 2J - d_{h_j} = -4J \ , \\ 4H^{(1)}_{o_2} &= |n|W - 2J + d_{h_j} - |n|W - 2J - d_{h_j} = -4J \ , \end{aligned} \tag{5.74}$$

so $H^{(2)}_{o_1 o_2} = 0$, $H^{(1)}_{o_1} = H^{(1)}_{o_2} = -J$.

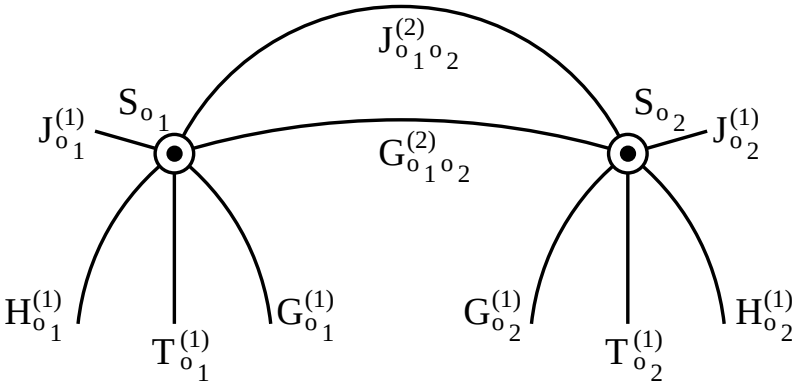

Figure 5.14: Equivalent decimated neural network.

Consider now that all the hidden units have been decimated, and thus we have an equivalent neural network that is represented in Fig. 5.14. In this picture, we have represented $J^{(2)}_{o_1 o_2}$, $J^{(1)}_{o_1}$ and $J^{(1)}_{o_2}$ as the connections that the original network had, $G^{(2)}_{o_1 o_2}$, $G^{(1)}_{o_1}$ and $G^{(1)}_{o_2}$ as the weights obtained through star-triangle association of unit $S_{h_{17}}$, $H^{(2)}_{o_1 o_2}$, $H^{(1)}_{o_1}$ and $H^{(1)}_{o_2}$ as the terms obtained decimating all the hidden, inactive units of the neural

network (this is, from unit S_{h_2} to S_{h_8}, from $S_{h_{10}}$ to $S_{h_{16}}$ and from $S_{h_{18}}$ to $S_{h_{24}}$) and, finally, $T_{o_1}^{(1)}$ and $T_{o_2}^{(1)}$ are the bias terms that are found by decimating active units S_{h_1} and S_{h_9}.

We now write down the values of these variables

$$\begin{aligned}
G_{o_1}^{(1)} &= \frac{d_{h_{17}}}{2} \ , \\
G_{o_2}^{(1)} &= \frac{d_{h_{17}}}{2} \ , \\
G_{o_1 o_2}^{(2)} &= H - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{17}}\right) \ , \\
H_{o_1}^{(1)} &= -J \ , \\
H_{o_2}^{(1)} &= -J \ , \\
T_{o_1}^{(1)} &= d_{h_1} \ , \\
T_{o_2}^{(1)} &= d_{h_9} \ ,
\end{aligned} \tag{5.75}$$

and add their value to the original $J_\sigma^{(n)}$ set of weights that connects the output units, thus obtaining a new set of weights $J_\sigma^{(n)'}$

$$\begin{aligned}
J_{o_1 o_2}^{(2)'} &= J_{o_1 o_2}^{(2)} + G_{o_1 o_2}^{(2)} \ , \\
J_{o_1}^{(1)'} &= J_{o_1}^{(1)} + G_{o_1}^{(1)} + 14 H_{o_1}^{(1)} + T_{o_1}^{(1)} \ , \\
J_{o_2}^{(1)'} &= J_{o_2}^{(1)} + G_{o_2}^{(1)} + 14 H_{o_2}^{(1)} + T_{o_2}^{(1)} \ ,
\end{aligned} \tag{5.76}$$

which leads to

$$\begin{aligned}
J_{o_1 o_2}^{(2)'} &= J_{o_1 o_2}^{(2)} + J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{17}}\right) \ , \\
J_{o_1}^{(1)'} &= J_{o_1}^{(1)} + \frac{d_{h_{17}}}{2} - 14J + d_{h_1} \ , \\
J_{o_2}^{(1)'} &= J_{o_2}^{(1)} + \frac{d_{h_{17}}}{2} - 14J + d_{h_9} \ .
\end{aligned} \tag{5.77}$$

Notice that $J_{o_1 o_2}^{(2)}$, $J_{o_1}^{(1)}$ and $J_{o_2}^{(1)}$ are still free and untouched. We can give values to them freely, and make them take the right values that cancel with quantities we want to remove. On the other hand, we compute $J_{o_1 o_2}^{(2)'}$, $J_{o_1}^{(1)'}$ and $J_{o_2}^{(1)'}$ by solving the backwards problem for the model shown in Fig. 5.15. In this sense, we would use the four probabilities $p\left(0|0\right)$,

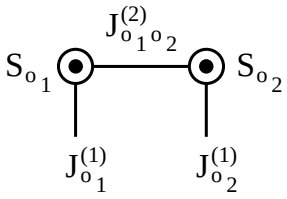

Figure 5.15: Backwards problem structure solved for two output units.

$p\,(1|0)$, $p\,(2|0)$, $p\,(3|0)$ as follows

$$\begin{aligned}
\ln\ln p\,(0|0) &= J^{(0)'} - J^{(1)'}_{o_1} - J^{(1)'}_{o_2} + J^{(2)'}_{o_1 o_2} \ , \\
\ln\ln p\,(1|0) &= J^{(0)'} - J^{(1)'}_{o_1} + J^{(1)'}_{o_2} - J^{(2)'}_{o_1 o_2} \ , \\
\ln\ln p\,(2|0) &= J^{(0)'} + J^{(1)'}_{o_1} - J^{(1)'}_{o_2} - J^{(2)'}_{o_1 o_2} \ , \\
\ln\ln p\,(3|0) &= J^{(0)'} + J^{(1)'}_{o_1} + J^{(1)'}_{o_2} + J^{(2)'}_{o_1 o_2} \ ,
\end{aligned} \tag{5.78}$$

so now we can find the values for the $J^{(n)}_{\sigma}$ set of weights. To this purpose, we analyze Eq. 5.77 and decide that d_{h_1}, d_{h_9} and $d_{h_{17}}$ depend on the value of the input units; an easy solution for the bias terms is then

$$\begin{aligned}
J^{(1)}_{o_1} &= 14J \ , \\
J^{(1)}_{o_2} &= 14J \ ,
\end{aligned} \tag{5.79}$$

however the second order weight is not so direct. Notice that

$$J^{(2)}_{o_1 o_2} = -J + \frac{1}{2}\ln 2 \ , \tag{5.80}$$

may lead to a non-existent solution for Eq. 5.77, because the inverse of the hyperbolic cosine might not exist. To prevent from this, we take

$$-J^{(2)'}_{o_1 o_2} - J^{(2)}_{o_1 o_2} - J + \frac{1}{2}\ln 2 = \frac{1}{4}\ln\cosh^2\left(d_{h_{17}}\right) \ , \tag{5.81}$$

and then carry out some basic operations

$$\sqrt{\mathrm{e}^{4\left(-J^{(2)'}_{o_1 o_2} - J^{(2)}_{o_1 o_2} - J + \frac{1}{2}\ln 2\right)}} = \cosh\left(d_{h_{17}}\right) \ . \tag{5.82}$$

In order to provide an always existing solution to this expression, one has to force the following condition

$$\sqrt{\mathrm{e}^{4\left(-J^{(2)\prime}_{o_1o_2}-J^{(2)}_{o_1o_2}-J+\frac{1}{2}\ln 2\right)}} \geq 1 \ , \tag{5.83}$$

and therefore

$$J^{(2)}_{o_1o_2} \leq \frac{1}{2}\ln 2 - J^{(2)\prime}_{o_1o_2} - J \ . \tag{5.84}$$

The process that one should follow then in order to find the correct weight is to solve the system in Eq. 5.78 for all possible output values, for all the possible values that the input units can take. In this example, and since there are 3 input units, this would lead to $2^3 = 8$ possible sets of equations; we would then find $J^{(2)}_{o_1o_2}$ for $d_{h_{17}}$ to $d_{h_{24}}$ and select the value that allows solving

$$\sqrt{\mathrm{e}^{4\left(-J^{(2)\prime}_{o_1o_2}-J^{(2)}_{o_1o_2}-J+\frac{1}{2}\ln 2\right)}} = \cosh\left(d_{h_j}\right) \ \text{, for any } j \in [17, 24] \ . \tag{5.85}$$

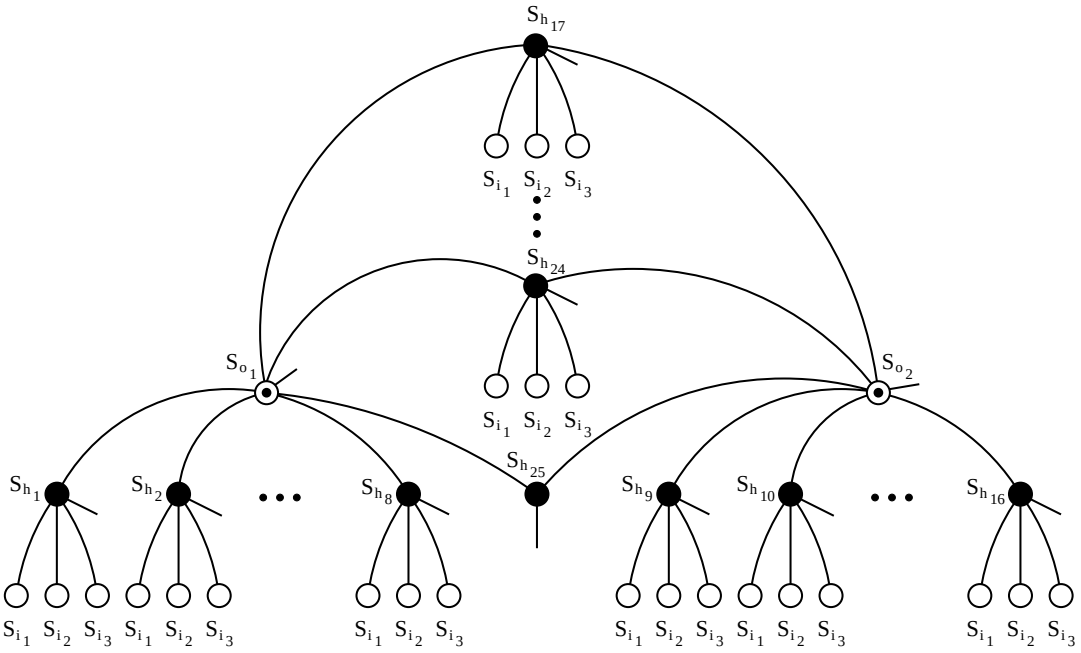

Figure 5.16: Structure with no second order weight connecting the output units, this connection is replaced by unit $S_{h_{25}}$.

However, we are more interested in showing that this system can be solved: it is always possible to build a second order neural network with two output units that solves a given problem that requires two output units. In this case, we would have to use $3 \cdot 2^3$ hidden neurons: 2^3 for each output, and 2^3 linked to both outputs; this is actually the maximum number of weights that one would use to connect these units. Notice now that, for an n-input BM, one would require up to $3 \cdot 2^n$ hidden neurons. In the following section, we now discuss the case for n_o output units and n_i input neurons.

Finally, it is interesting to point out that $J^{(2)}_{o_1 o_2}$ is included in our analysis but this may not be required. In this sense, we could have used the topology shown in Fig. 5.16 instead, where this connection is replaced by a hidden unit connected to both output units, due to the star-triangle decimation equivalence.

5.2.5 General case for the output joint probability distribution

We now show that the topology proposed in the previous sections can be extended to analyze the more general case of having n_i inputs and n_o outputs. We begin this discussion with a three outputs, second order BM with n_i input units. For the sake of simplicity, we consider that all input units have already been parallel associated, thus leading to the neural network that we represent in Fig. 5.17. In this figure, we use the concepts detailed above: there are 2^{n_i} hidden units for each output unit, 2^{n_i} hidden units for each second order connection between the three units, thus yielding $3 \cdot 2^{n_i}$ neurons; and finally 2^{n_i} hidden units connecting all three output units altogether. Finally, the values of the weights are $J^{(2)}_{ho} = J$ for connections between hidden and output units, $\left|J^{(2)}_{ih}\right| = W$ for the connections between input and hidden units; the sign for these weights is again given depending on the input combination that activates them. The bias terms $J^{(1)}_h$ are set as $J^{(1)}_h = -n_i W + d_h$, where d_h is a value that depends on each hidden unit and whose value will be discussed in this section.

Notice however that the hidden units connecting the output units in pairs have already been discussed in the previous section, as the hidden units connected only to a given

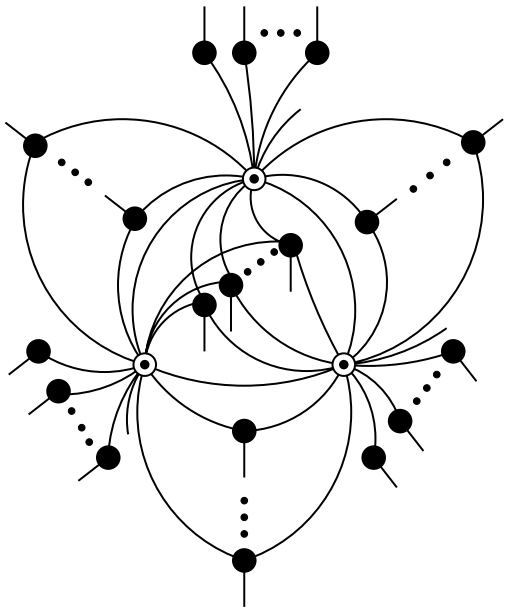

Figure 5.17: Structure with three output units and n_i input neurons. Notice that, in order to create a simpler figure, the input units connecting the hidden neurons are already associated with the bias terms.

output. Since each one of these neurons is isolated from the other hidden units, the analysis carried out before is still valid. In this sense, we do only need to analyze what happens with the hidden units connecting all the outputs altogether, which are the units shown in Fig. 5.18, and discuss how decimating these hidden units affects the system.

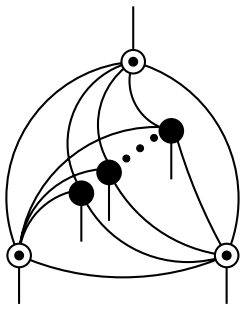

Figure 5.18: Structure with three output units with a decimated structure, there are only left these connections that will generate third order weights.

We will also consider that the hidden units from Fig. 5.18 are activated by following

the same principles that we describe above: there is a single hidden unit that is active for a certain input value, while the other ones are inactive and thus $S_h = -1$. We will name the active unit as S_{h_a}, the inactive ones will be therefore referred to as S_{h_i}. Since they are not connected between themselves, this should not be an issue. We begin our discussion by decimating unit S_{h_a} and so we recall the high order Decimation equations for this case

$$\begin{aligned}
G^{(3)}_{o_1o_2o_3} &= \frac{1}{8}\ln\left(\frac{A_1A_2A_4A_7}{A_0A_3A_5A_6}\right) \ , \\
G^{(2)}_{o_1o_2} &= \frac{1}{8}\ln\left(\frac{A_0A_1A_6A_7}{A_2A_3A_4A_5}\right) \ , \\
G^{(2)}_{o_1o_3} &= \frac{1}{8}\ln\left(\frac{A_0A_2A_5A_7}{A_1A_3A_4A_6}\right) \ , \\
G^{(2)}_{o_2o_3} &= \frac{1}{8}\ln\left(\frac{A_0A_3A_4A_7}{A_1A_2A_5A_6}\right) \ , \\
G^{(1)}_{o_1} &= \frac{1}{8}\ln\left(\frac{A_4A_5A_6A_7}{A_0A_1A_2A_3}\right) \ , \\
G^{(1)}_{o_2} &= \frac{1}{8}\ln\left(\frac{A_2A_3A_6A_7}{A_0A_1A_4A_5}\right) \ , \\
G^{(1)}_{o_3} &= \frac{1}{8}\ln\left(\frac{A_1A_3A_5A_7}{A_0A_2A_4A_6}\right) \ ,
\end{aligned} \tag{5.86}$$

where A_i is given by the following relations

$$\begin{aligned}
\ln\cosh\left(J^{(1)}_{h_a} - J^{(2)}_{h_ao_1} - J^{(2)}_{h_ao_2} - J^{(2)}_{h_ao_3}\right) &= \ln A_0 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} - J^{(2)}_{h_ao_1} - J^{(2)}_{h_ao_2} + J^{(2)}_{h_ao_3}\right) &= \ln A_1 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} - J^{(2)}_{h_ao_1} + J^{(2)}_{h_ao_2} - J^{(2)}_{h_ao_3}\right) &= \ln A_2 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} - J^{(2)}_{h_ao_1} + J^{(2)}_{h_ao_2} + J^{(2)}_{h_ao_3}\right) &= \ln A_3 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} + J^{(2)}_{h_ao_1} - J^{(2)}_{h_ao_2} - J^{(2)}_{h_ao_3}\right) &= \ln A_4 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} + J^{(2)}_{h_ao_1} - J^{(2)}_{h_ao_2} + J^{(2)}_{h_ao_3}\right) &= \ln A_5 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} + J^{(2)}_{h_ao_1} + J^{(2)}_{h_ao_2} - J^{(2)}_{h_ao_3}\right) &= \ln A_6 \ , \\
\ln\cosh\left(J^{(1)}_{h_a} + J^{(2)}_{h_ao_1} + J^{(2)}_{h_ao_2} + J^{(2)}_{h_ao_3}\right) &= \ln A_7 \ ,
\end{aligned} \tag{5.87}$$

where $J^{(2)}_{h_ao_k} = J \ \forall k$ and, for the same reasons discussed in the previous section, we take

$J_{h_a}^{(1)} = d_{h_a}$, considering again that $J \gg d_{h_a}$. Now

$$\begin{aligned}
\ln A_0 &= \ln\cosh\left(3J - d_{h_a}\right) \ , \\
\ln A_1 &= \ln\cosh\left(J - d_{h_a}\right) \ , \\
\ln A_2 &= \ln\cosh\left(J - d_{h_a}\right) \ , \\
\ln A_3 &= \ln\cosh\left(J + d_{h_a}\right) \ , \\
\ln A_4 &= \ln\cosh\left(J - d_{h_a}\right) \ , \\
\ln A_5 &= \ln\cosh\left(J + d_{h_a}\right) \ , \\
\ln A_6 &= \ln\cosh\left(J + d_{h_a}\right) \ , \\
\ln A_7 &= \ln\cosh\left(3J + d_{h_a}\right) \ ,
\end{aligned} \tag{5.88}$$

and using the approximation from Eq. 5.24

$$\ln\cosh\left(nJ\right) \simeq nJ - \ln 2 \ , \tag{5.89}$$

we use Eq. 5.86 to get

$$\begin{aligned}
\ln A_0 &\simeq 3J - d_{h_a} \ , \\
\ln A_1 &\simeq J - d_{h_a} \ , \\
\ln A_2 &\simeq J - d_{h_a} \ , \\
\ln A_3 &\simeq J + d_{h_a} \ , \\
\ln A_4 &\simeq J - d_{h_a} \ , \\
\ln A_5 &\simeq J + d_{h_a} \ , \\
\ln A_6 &\simeq J + d_{h_a} \ , \\
\ln A_7 &\simeq 3J + d_{h_a} \ ,
\end{aligned} \tag{5.90}$$

so we now find the values of the weights

$$
\begin{aligned}
G^{(3)}_{o_1o_2o_3} &= -\frac{d_{h_a}}{2} \ , \\
G^{(2)}_{o_1o_2} &= \frac{J}{2} \ , \\
G^{(2)}_{o_1o_3} &= \frac{J}{2} \ , \\
G^{(2)}_{o_2o_3} &= \frac{J}{2} \ , \\
G^{(1)}_{o_1} &= \frac{d_{h_a}}{2} \ , \\
G^{(1)}_{o_2} &= \frac{d_{h_a}}{2} \ , \\
G^{(1)}_{o_3} &= \frac{d_{h_a}}{2} \ .
\end{aligned} \tag{5.91}
$$

Notice how the new bias and the high order term depend only on the original parameter from the hidden active unit d_{h_a}.

We also have to analyze the case where the hidden units are inactive. In order to do so, we proceed as we did in the previous section: the equivalent bias term for units S_{h_i} is now $J^{(1)}_{h_i} = -nW + d_{h_i}$, being n any value that results from the parallel association of the weights from the input units. We use the same approximation as above, thus considering that $W \gg J$ and hence

$$
\begin{aligned}
\ln A_0 &\simeq nW + 3J - d_{h_i} \ , \\
\ln A_1 &\simeq nW + J - d_{h_i} \ , \\
\ln A_2 &\simeq nW + J - d_{h_i} \ , \\
\ln A_3 &\simeq nW - J - d_{h_i} \ , \\
\ln A_4 &\simeq nW + J - d_{h_i} \ , \\
\ln A_5 &\simeq nW - J - d_{h_i} \ , \\
\ln A_6 &\simeq nW - J - d_{h_i} \ , \\
\ln A_7 &\simeq nW - 3J - d_{h_i} \ ,
\end{aligned} \tag{5.92}
$$

thus yielding

$$
\begin{aligned}
G^{(3)}_{o_1 o_2 o_3} &= 0 \ , \\
G^{(2)}_{o_1 o_2} &= 0 \ , \\
G^{(2)}_{o_1 o_3} &= 0 \ , \\
G^{(2)}_{o_2 o_3} &= 0 \ , \\
G^{(1)}_{o_1} &= -J \ , \\
G^{(1)}_{o_2} &= -J \ , \\
G^{(1)}_{o_3} &= -J \ . \qquad (5.93)
\end{aligned}
$$

This process can be repeated for all the possible values that the input units can take, thus effectively finding all the connections of the neural network. It is then possible to find all the weights needed to build a second order BM with three output units and an arbitrary number of input neurons.

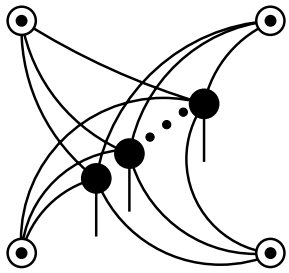

Figure 5.19: Structure that is added for a four output units and n_i input neurons.

Now we inquiry about the case for a neural network with four output units: we create a structure that uses the same topology as shown above, that should be added to the one shown in Fig. 5.19. We will add to this model 2^{n_i} hidden units connected to each separate output unit and to all the inputs, for a total of $4 \cdot 2^{n_i}$ units. We will also add 2^{n_i} hidden units connecting all the possible pairs of outputs, for a total of $6 \cdot 2^{n_i}$ hidden units; and finally 2^{n_i} hidden units connecting the output units in groups of three, this will make $4 \cdot 2^{n_i}$ more hidden units; and a total number of $14 \cdot 2^{n_i}$ which adds to the 2^{n_i} units

from Fig. 5.19 for a final number of $15 \cdot 2^{n_i}$ hidden units. Notice then that this stands for $(2^4 - 1)2^{n_i}$ hidden units.

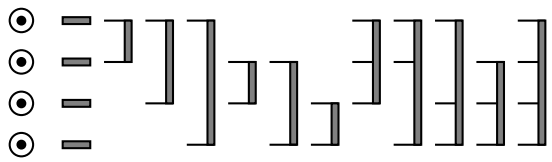

Figure 5.20: Dressed weights structure to build a BM with 4 output units.

The resulting structure is shown in a simplified version in Fig. 5.20. This representation uses *dressed* weights in order to show the hidden units connected to the output units: the equivalence for this weights is depicted in Fig. 5.21, where one can generalize for a high order dressed weight.

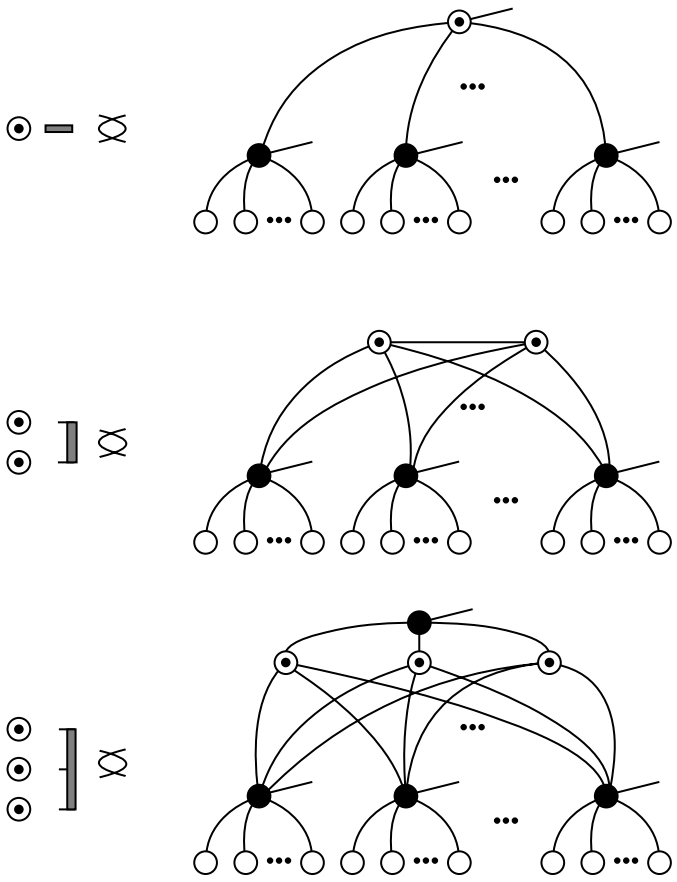

Figure 5.21: Dressed weights equivalence up to a third order dressed connection.

Notice now that a BM with five output units will use as many as $2^{n_o} - 1$ dressed weights, which represent all the required connections with the input and hidden units. This structure is shown in Fig. 5.22.

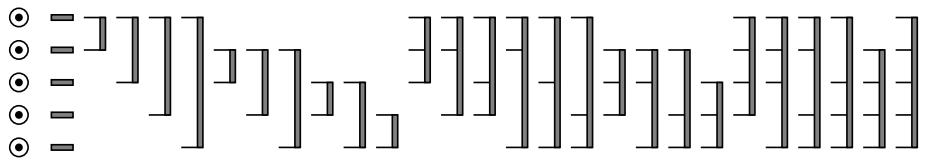

Figure 5.22: Structure used to build a 5 outputs BM.

This process can be repeated to build any BM, regardless of the number of input and output units that it needs. In essence, the structure that one needs is shown in Fig. 5.23, where there are $2^{n_o}-1$ dressed weights that one uses to reproduce any probability distribution. Since each of these weights stands for 2^{n_i} input units, the resulting neural network yields a total of $(2^{n_o} - 1)2^{n_i}$ hidden units.

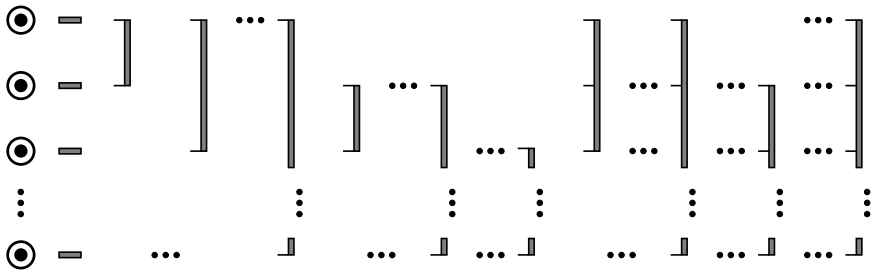

Figure 5.23: Structure used to build an n_o outputs BM.

5.2.6 Error term due to the hyperbolic cosine approximation

In the previous sections, we have made an approximation for the hyperbolic cosine terms of the decimation expressions; so now we inquiry about the error introduced there. We recall the approximation from Eq. 5.24, which in a general form is expressed as

$$\lim_{W\to\infty} \ln\cosh\left(nW - J - d_h\right) = nW - J - d_h - \ln 2 \ , \tag{5.94}$$

for finite J and d_h, and $n \in \mathbb{N}^*$; or either

$$\lim_{J\to\infty} \ln\cosh\left(nJ - d_h\right) = nJ - d_h - \ln 2 \ , \tag{5.95}$$

for finite d_h and $n \in \mathbb{N}^*$, again; and depending on the section of the neural network that we are decimating. Notice however that the first expression assumes $|W| \gg |J| \pm |d_h|$, while the second one goes for $|J| \gg |d_h|$. Since the approximation that we are using here considers the difference of the biggest terms in the operation where the error is introduced, we will later show how both Eq. 5.94 and Eq. 5.95 are unified.

We will consider that, for a multiple variable function

$$z = f\left(x_1, x_2, \ldots, x_N\right) \ , \tag{5.96}$$

the error ε_z is, to first order

$$\varepsilon_z = \sqrt{\left(\frac{\partial f}{\partial x_1}\varepsilon_{x_1}\right)^2 + \left(\frac{\partial f}{\partial x_2}\varepsilon_{x_2}\right)^2 + \ldots + \left(\frac{\partial f}{\partial x_M}\varepsilon_{x_M}\right)^2} \ , \tag{5.97}$$

where ε_{x_j} is the error introduced by the j-th variable. In our case, we define

$$z = f\left(x_1, x_2, \ldots, x_M\right) = \sum_{j=1}^{M} (-1)^{\alpha} \ln\cosh\left(x_j\right) \ \ , \ \ \alpha_j = 0, 1 \ , \tag{5.98}$$

and consider that M is the product of three values

$$M = m_1 \cdot m_2 \cdot m_3 \ , \tag{5.99}$$

where m_1 is the number of times that one has to apply decimation to the neural network. Since we decimate all the hidden units, this stands for $2^{n_i+n_o} - 2^{n_i}$. On the other hand, m_2 is the number of times that we associate the resulting weights in parallel, this is, $m_2 = 2^{n_i+n_o} - 2^{n_i}$, which is the number of dressed weights that we have in the system and the sum of the resulting decimated weights that they represent. Finally, m_3 is the number of operations that one has to carry out when decimating a given unit. This value depends on the number of neurons this unit is connected to, being 2 for two units and 2^{n_o} for n_o output units. The error reaches its worst value for $m_3 = 2^{n_o}$, hence

$$M \leq \left(2^{n_i+n_o} - 2^{n_i}\right)^2 2^{n_o} \ . \tag{5.100}$$

Now we need to define the error for a given x_j, we begin from

$$\begin{aligned} \ln\cosh\left(x_j\right) &= \ln\left(\mathrm{e}^{x_j}+\mathrm{e}^{-x_j}\right)-\ln 2 = \ln\left[\mathrm{e}^{x_j}\left(1+\mathrm{e}^{-2x_j}\right)\right]-\ln 2 \\ &= x_j+\ln\left(1+\mathrm{e}^{-2x_j}\right)-\ln 2 \ , \end{aligned} \tag{5.101}$$

hence

$$\varepsilon_{x_j} = \ln\left(1+\mathrm{e}^{-2x_j}\right) \ , \tag{5.102}$$

because $x_j \to \infty$. Now we calculate the partial derivatives of z as

$$\frac{\partial f}{\partial x_j} = \frac{1}{\cosh\left(x_j\right)}\sinh\left(x_j\right) = \tanh\left(x_j\right) \ , \tag{5.103}$$

and finally arrive at

$$\varepsilon_z = \sqrt{\sum_{j=1}^{M}\tanh^2\left(x_j\right)\ln^2\left(1+\mathrm{e}^{-2x_j}\right)} \ . \tag{5.104}$$

The worst case for this expression is that $\tanh^2\left(x_j\right)=1$, and thus

$$\varepsilon_z = \sqrt{\sum_{j=1}^{M}\tanh^2\left(x_j\right)\ln^2\left(1+\mathrm{e}^{-2x_j}\right)} \leq \sqrt{\sum_{j=1}^{M}\ln^2\left(1+\mathrm{e}^{-2x_j}\right)} \ . \tag{5.105}$$

If we consider the biggest x_j to be y, we can obtain a standard expression form these worst cases

$$x_j = y = |W| - J - |d| = A_j - B_j \ , \ \forall \ j \ , \tag{5.106}$$

for $B_j = J + |d|$, or either

$$x_j = y = J - |d| = A_j - B_j \ , \ \forall \ j \ , \tag{5.107}$$

and so we will use a certain $A-B$ such as the minimum value of any given, possible worst case

$$A - B = \max\left\{|W| - J - |d|\,, J - |d|\right\} \ , \tag{5.108}$$

thus we can write a general equation for the error as

$$\varepsilon_z = \left(2^{n_i+n_o}-2^{n_o}\right)2^{\frac{n_o}{2}}\ln\left(1+\mathrm{e}^{-2A+2B}\right) \ , \tag{5.109}$$

notice then that the error can be expressed in terms of a Taylor expansion for the logarithmic term as

$$\varepsilon_z \simeq \left(2^{n_i+n_o} - 2^{n_o}\right) 2^{\frac{n_o}{2}} \frac{1}{\mathrm{e}^{-2A+2B}} \ , \tag{5.110}$$

However, since we are building the BM, n_i and n_o are imposed and W, J and d are fixed. In this sense, we can guarantee that the error can be made arbitrarily small, because we will select the values that satisfy $W \gg J \gg d$ and $J \gg n_i + n_o$. It has been shown then that it is possible to build a second order BM that is able to learn any given probability distribution with an error as low as desired.

5.3 Practical implementation of a BM

In this section we build two BMs by using the method presented above. We first describe the process that should be followed in order to use the equations shown above to build a given BM model. We then proceed by building a simple neural network with two input units and two output neurons, in this example all the weights of the system are calculated and explicitly written down. The section is concluded highlighting the more relevant aspects of the process, regarding a second order BM with three input and three output units.

5.3.1 Description of the implementation

We now describe the process that one should follow in order to build a given BM model according to a certain p.d.f. that describes its behavior. This process is carried out by using the equations that have been discussed above, and hence we will divide it in the following steps:

1. Count the number of required hidden units.

2. Carry out a backwards learning problem to obtain the values of the high order weights for each given input value.

3. Set the values of the d_{h_j} terms from the bias terms of the hidden units that become active due to each one of the input patterns.

4. Give values to the weights connecting any pair of connected units, according to the previous equations and the desired error term value.

The number oh hidden units n_h depends on the number of input n_i and output n_o neurons: we will need $2^{n_i}+1$ hidden units for each connection that is generated through the backwards problem solution, with the exception of the first and second order terms. This value is directly found as $2^{n_o}-1$, hence one arrives to

$$\begin{aligned} n_h &= 2^{n_i}\left(2^{n_o}-1\right) \text{ if } n_o \in [1,3] \ , \\ n_h &= \left(2^{n_i}+1\right)\left(2^{n_o}-1\right) - n_o - \frac{n_o\left(n_o-1\right)}{2} \text{ if } n_o > 3 \ . \end{aligned} \tag{5.111}$$

5.3.2 Two inputs, two outputs BM

In this section we show how the process described so far is used to build a second order BM with two inputs and two output units. We will create a system that is able to learn the probability distribution that is represented in table 5.25.

To build this neural network, we use the structure represented in Fig. 5.24, and that has been discussed in the previous section. We will carry out this process by assuming that $J^{(2)}_{ij h_k} = |W|$, $J^{(2)}_{h_j o_k} = J$ and that $J^{(1)}_{h_j} = -2W + d_{h_j}$, the objective is therefore to find W (where the sign is fixed according to the unit that becomes active due to a certain unit), J and d_{h_j} for $j = 1$ to $j = 12$. We begin this process by finding $J^{(1)}_{o_1}$, $J^{(1)}_{o_2}$ and $J^{(2)}_{o_1 o_2}$. In order to calculate these values it is necessary to set d_{h_j}, d_{h_k} and d_{h_l}, corresponding to the hidden units that connect S_{o_1}, S_{o_2} and both of them, respectively. We first decimate the system in Fig. 5.24 to reach the one shown in Fig. 5.25, which contains the weights $J^{(1)}_{o_1}$, $J^{(1)}_{o_2}$ and $J^{(2)}_{o_1 o_2}$.

We then proceed by decimating this structure to obtain the final one shown in Fig. 5.26, which has the set of weights $G^{(1)}_{o_1}$, $G^{(1)}_{o_2}$ and $G^{(2)}_{o_1 o_2}$. These weights are related to $J^{(1)}_{o_1}$, $J^{(1)}_{o_2}$

S_{i_1}	S_{i_2}	S_{o_1}	S_{o_2}	$p\left(S_{o_1}, S_{o_2} \vert S_{i_1}, S_{i_2}\right)$
-1	-1	-1	-1	0.1
-1	-1	-1	1	0.4
-1	-1	1	-1	0.4
-1	-1	1	1	0.1
-1	1	-1	-1	0.2
-1	1	-1	1	0.3
-1	1	1	-1	0.4
-1	1	1	1	0.1
1	-1	-1	-1	0.4
1	-1	-1	1	0.1
1	-1	1	-1	0.1
1	-1	1	1	0.4
1	1	-1	-1	0.7
1	1	-1	1	0.1
1	1	1	-1	0.1
1	1	1	1	0.1

Table 5.25: Output probability distribution for a two input two output BM.

and $J^{(2)}_{o_1 o_2}$ by the following expressions

$$G^{(1)}_{o_1} = J^{(1)}_{o_1} + \frac{d_{h_l}}{2} + d_{h_j} - 6J \ , \tag{5.112}$$

$$G^{(1)}_{o_2} = J^{(1)}_{o_2} + \frac{d_{h_l}}{2} + d_{h_k} - 6J \ , \tag{5.113}$$

$$G^{(2)}_{o_1 o_2} = J^{(2)}_{o_1 o_2} + J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_l}\right) \ . \tag{5.114}$$

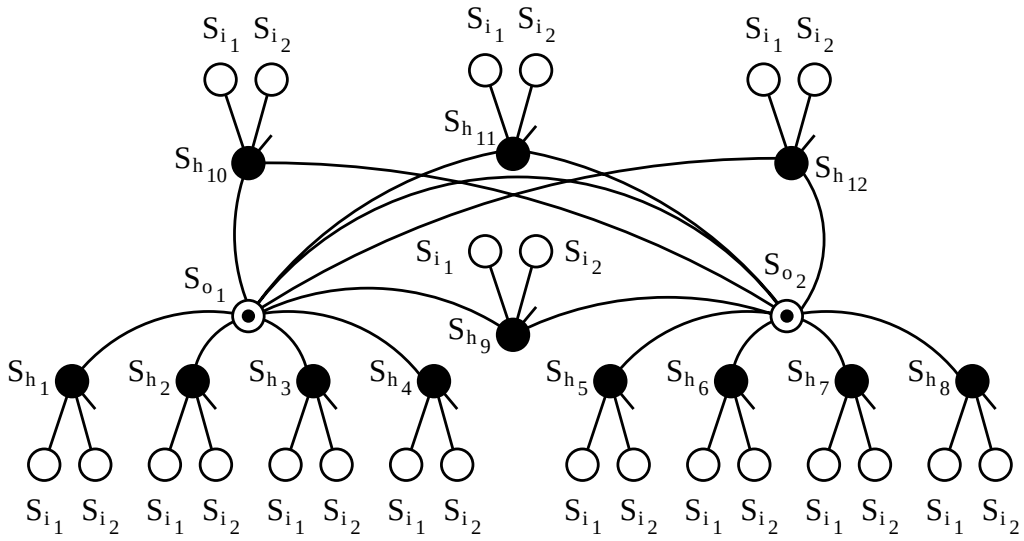

Figure 5.24: Structure with two output units and two input neurons.

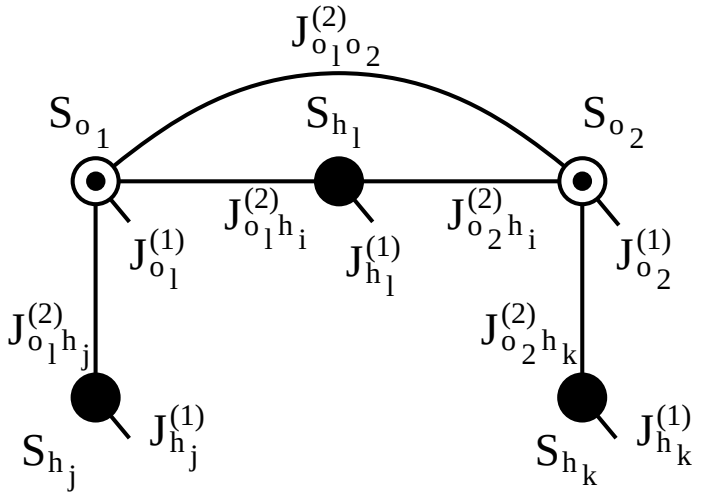

Figure 5.25: Decimated structure with the hidden units.

The set of weights $G^{(1)}_{o_1}$, $G^{(1)}_{o_2}$ and $G^{(2)}_{o_1 o_2}$ is found by solving the backwards problem as

$$\begin{aligned}
\ln p\left(S_{o_1}=-1, S_{o_2}=-1\right) &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(2)}_{o_1 o_2} \ , \\
\ln p\left(S_{o_1}=-1, S_{o_2}=1\right) &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(2)}_{o_1 o_2} \ , \\
\ln p\left(S_{o_1}=1, S_{o_2}=-1\right) &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(2)}_{o_1 o_2} \ , \\
\ln p\left(S_{o_1}=1, S_{o_2}=1\right) &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(2)}_{o_1 o_2} \ , \qquad (5.115)
\end{aligned}$$

notice however that these expressions are solved for each one of the combinations that

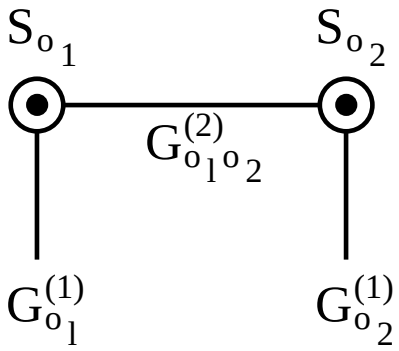

Figure 5.26: Structure with the active set of hidden units.

the input units can take. Hence, one will have to solve this equations for $S_{i_1} = S_{i_2} = -1$, as well as $S_{i_1} = 1$, $S_{i_2} = -1$; $S_{i_1} = -1$, $S_{i_2} = 1$ and finally $S_{i_1} = S_{i_2} = 1$. The results for each input vector are represented in table 5.26.

S_{i_1}	S_{i_2}	$G^{(1)}_{o_1}$	$G^{(1)}_{o_2}$	$G^{(2)}_{o_1 o_2}$
-1	-1	0.0	0.0	-0.15
-1	1	0.0	-0.05	-0.10
1	-1	0.0	0.0	0.15
1	1	-0.15	-0.15	0.15

Table 5.26: Backwards problem solution for each input vector combination.

Now, using Eq. 5.114 we can decide that $J^{(1)}_{o_1} = J^{(1)}_{o_2} = 6J$, and from the results in table 5.26 we find

$$\begin{aligned}
-0.15 &= J^{(2)}_{o_1 o_2} + J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_9}\right) \quad , \\
-0.10 &= J^{(2)}_{o_1 o_2} + J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{10}}\right) \quad , \\
0.15 &= J^{(2)}_{o_1 o_2} + J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{11}}\right) \quad , \\
0.15 &= J^{(2)}_{o_1 o_2} + J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{12}}\right) \quad , \qquad (5.116)
\end{aligned}$$

and we set up the values that allow the hyperbolic cosine to be inverted. Hence

$$\begin{aligned}
\cosh(d_9) &= \mathrm{e}^{2J^{(2)}_{o_1o_2}+2J-\ln 2+0.3} \geq 1 \ , \\
\cosh(d_{10}) &= \mathrm{e}^{2J^{(2)}_{o_1o_2}+2J-\ln 2+0.2} \geq 1 \ , \\
\cosh(d_{11}) &= \mathrm{e}^{2J^{(2)}_{o_1o_2}+2J-\ln 2-0.3} \geq 1 \ , \\
\cosh(d_{12}) &= \mathrm{e}^{2J^{(2)}_{o_1o_2}+2J-\ln 2-0.3} \geq 1 \ .
\end{aligned} \tag{5.117}$$

Now we will have to set a proper value to $J^{(2)}_{o_1o_2}$ and J to find d_{h_j}. The most restrictive condition comes from the last two equations and mean that

$$2J^{(2)}_{o_1o_2} + 2J - \ln 2 - 0.3 \geq 0 \ , \tag{5.118}$$

and therefore

$$J^{(2)}_{o_1o_2} = -J + \frac{1}{2}\ln 2 + 0.15 = -J + 0.50 \ , \tag{5.119}$$

which, once inserted back in Eq. 5.117, leads to

$$\begin{aligned}
\cosh(d_{h_9}) &= \mathrm{e}^{0.6} = 1.82 \geq 1 \ , \\
\cosh(d_{h_{10}}) &= \mathrm{e}^{0.5} = 1.65 \geq 1 \ , \\
\cosh(d_{h_{11}}) &= \cosh(d_{12}) = \mathrm{e}^{0} = 1 \geq 1 \ ,
\end{aligned} \tag{5.120}$$

and one finds

$$\begin{aligned}
d_{h_9} &= 1.21 \ , \\
d_{h_{10}} &= 1.09 \ , \\
d_{h_{11}} &= d_{h_{12}} = 0.0 \ .
\end{aligned} \tag{5.121}$$

Now we have found d_{h_j} for $j = 9, 10, 11, 12$, as the bias terms from the central unit shown in Fig. 5.25. Notice then that $J^{(2)}_{o_1o_2}$ and J are still undefined; we will set their values at the end of the calculations. Now we go for the other terms of Fig. 5.25; since we have set $J^{(1)}_{o_1} = J^{(1)}_{o_2} = 6J$, we get

$$\begin{aligned}
G^{(1)}_{o_1} &= J^{(1)}_{o_1} + \frac{d_{h_l}}{2} + d_{h_j} - 6J = \frac{d_{h_l}}{2} + d_{h_j} \ , \\
G^{(1)}_{o_2} &= J^{(1)}_{o_2} + \frac{d_{h_l}}{2} + d_{h_k} - 6J = \frac{d_{h_l}}{2} + d_{h_k} \ ,
\end{aligned} \tag{5.122}$$

and solve this system for d_{h_1}, d_{h_2}, d_{h_3}, d_{h_4}, d_{h_5}, d_{h_6}, d_{h_7} and d_{h_8}. Thus, using the values from table 5.26, we obtain the following results corresponding to the input units values as

$$\left.\begin{array}{l} 0.0 = \frac{d_{h_9}}{2} + d_{h_1} \\ 0.0 = \frac{d_{h_9}}{2} + d_{h_5} \end{array}\right\} \text{for } S_{i_1} = -1,\ \ S_{i_2} = -1\ ,$$

$$\left.\begin{array}{l} 0.0 = \frac{d_{h_{10}}}{2} + d_{h_2} \\ -0.05 = \frac{d_{h_{10}}}{2} + d_{h_6} \end{array}\right\} \text{for } S_{i_1} = -1,\ \ S_{i_2} = 1\ ,$$

$$\left.\begin{array}{l} 0.0 = \frac{d_{h_{11}}}{2} + d_{h_3} \\ 0.0 = \frac{d_{h_{11}}}{2} + d_{h_7} \end{array}\right\} \text{for } S_{i_1} = 1,\ \ S_{i_2} = -1\ ,$$

$$\left.\begin{array}{l} -0.15 = \frac{d_{12}}{2} + d_{h_4} \\ -0.15 = \frac{d_{12}}{2} + d_{h_8} \end{array}\right\} \text{for } S_{i_1} = 1,\ \ S_{i_2} = 1\ , \qquad (5.123)$$

so we get trivially

$$\begin{aligned} d_{h_1} &= -0.61\ , \\ d_{h_5} &= -0.61\ , \\ d_{h_2} &= 0.55\ , \\ d_{h_6} &= 0.52\ , \\ d_{h_3} &= 0.0\ , \\ d_{h_7} &= 0.0\ , \\ d_{h_4} &= -0.15\ , \\ d_{h_8} &= -0.15\ . \end{aligned} \qquad (5.124)$$

Now that we know the values for the bias terms, we can finally fix J and W. This values are set satisfying $|W| \gg |J| \gg \left|d_{h_j}\right|$, so the error introduced in the approximation from Eq. 5.24 is little enough. We can take for instance

$$\begin{aligned} 2W - 2J - 2\left|d_{h_j}\right| &\leq 9.3\ ,\ \ \forall j\ , \\ 2J - 2\left|d_{h_j}\right| &\leq 9.3\ ,\ \ \forall j\ , \end{aligned} \qquad (5.125)$$

thus yielding $W = 19.6$, $J = 10.3$ and $A - B = 9.3$ in

$$A - B = \min\left\{|W| - J - |d|\,, J - |d|\right\} \ , \tag{5.126}$$

from

$$\varepsilon_z \simeq \left(2^{n_i+n_o} - 2^{n_o}\right) 2^{\frac{n_\rho}{2}} \frac{1}{\mathrm{e}^{2A-2B}} \leq 0.01 \ , \tag{5.127}$$

hence the error ε_z becomes smaller than 10^{-2}.

We finally write down all the the values of the weights for this BM

$$\begin{aligned}
&-J^{(2)}_{i_1h_1} = -J^{(2)}_{i_2h_1} = -J^{(2)}_{i_1h_2} = J^{(2)}_{i_2h_2} = J^{(2)}_{i_1h_3} = -J^{(2)}_{i_2h_3} = J^{(2)}_{i_1h_4} = J^{(2)}_{i_2h_4} = 19.6 \ , \\
&-J^{(2)}_{i_1h_5} = -J^{(2)}_{i_2h_5} = -J^{(2)}_{i_1h_6} = J^{(2)}_{i_2h_6} = J^{(2)}_{i_1h_7} = -J^{(2)}_{i_2h_7} = J^{(2)}_{i_1h_8} = J^{(2)}_{i_2h_8} = 19.6 \ , \\
&-J^{(2)}_{i_1h_9} = -J^{(2)}_{i_2h_9} = -J^{(2)}_{i_1h_{10}} = J^{(2)}_{i_2h_{10}} = J^{(2)}_{i_1h_{11}} = -J^{(2)}_{i_2h_{11}} = J^{(2)}_{i_1h_{12}} = J^{(2)}_{i_2h_{12}} = 19.6 \ , \\
&J^{(2)}_{h_1o_1} = J^{(2)}_{h_2o_1} = J^{(2)}_{h_3o_1} = J^{(2)}_{h_4o_1} = 10.3 \ , \\
&J^{(2)}_{h_5o_1} = J^{(2)}_{h_6o_1} = J^{(2)}_{h_7o_1} = J^{(2)}_{h_8o_1} = 10.3 \ , \\
&J^{(2)}_{h_9o_1} = J^{(2)}_{h_{10}o_1} = J^{(2)}_{h_{11}o_1} = J^{(2)}_{h_{12}o_1} = 10.3 \ , \\
&J^{(2)}_{h_1o_2} = J^{(2)}_{h_2o_2} = J^{(2)}_{h_3o_2} = J^{(2)}_{h_4o_2} = 10.3 \ , \\
&J^{(2)}_{h_5o_2} = J^{(2)}_{h_6o_2} = J^{(2)}_{h_7o_2} = J^{(2)}_{h_8o_2} = 10.3 \ , \\
&J^{(2)}_{h_9o_2} = J^{(2)}_{h_{10}o_2} = J^{(2)}_{h_{11}o_2} = J^{(2)}_{h_{12}o_2} = 10.3 \ , \\
&J^{(2)}_{o_1o_2} = -9.7 \ ,
\end{aligned} \tag{5.128}$$

and the bias terms

$$\begin{aligned}
&J^{(1)}_{h_1} = -39.81 \ \ J^{(1)}_{h_2} = -38.65 \ \ J^{(1)}_{h_3} = -39.2 \ \ J^{(1)}_{h_4} = -39.35 \ , \\
&J^{(1)}_{h_5} = -39.81 \ \ J^{(1)}_{h_6} = -38.68 \ \ J^{(1)}_{h_7} = -39.2 \ \ J^{(1)}_{h_8} = -39.35 \ , \\
&J^{(1)}_{h_9} = -37.99 \ \ J^{(1)}_{h_{10}} = -38.11 \ \ J^{(1)}_{h_{11}} = -39.2 \ \ J^{(1)}_{h_{12}} = -39.2 \ , \\
&J^{(1)}_{o_1} = 61.8 \ \ J^{(1)}_{o_2} = 61.8 \ .
\end{aligned} \tag{5.129}$$

Notice now that the error from the approximation can become minimized if the values set at W and J are big enough. In this sense, it is possible to select any given value

provided that the error related equations are satisfied, because the probability distribution of the neural network is based upon the subtraction or addition of the values when parallel decimating.

5.3.3 Three inputs, three outputs BM

In this last example we build a BM with three input units and three output neurons. We create a system that is able to learn the probability distribution that is represented in table 5.27; notice that there are only some instances of the $2^{3+3} = 64$ total number of states that can be generated with three input and three output units. Hence, this problem is not exhaustive. This has been made for practical reasons, since an exhaustive, fully defined probability distribution requires an excessive amount of weights: there are 2^3 hidden units connecting each output with the input units for some dressed bias terms, these need 4 weights each and a bias term for a total of $3 \cdot 2^3 \cdot 4 = 96$ weights, plus the 3 bias terms of the output units; 2^3 hidden units linking each pair of output units which are represented as 3 second order dressed weights, with 4 weights each one for $3 \cdot 2^3 \cdot 5 = 120$ weights, which are added to the 3 second order terms connecting the output neurons. Finally, there is a third order dressed weight that needs 2^3 hidden units with 6 connections each, for a total of $3 \cdot 2^3 \cdot 6 = 144$ weights. In essence, this makes a total of $3 \cdot 2^3 + 3 \cdot 2^3 + 2^3 = 56$ hidden units and $99 + 123 + 144 = 366$ weights. However, since we are only considering the probability distribution associated to three different input patterns, we will use the structure is represented in Fig. 5.27 using the dressed weights notation from the previous section, this is shown in Fig. 5.28. Notice though that we will only need 3 hidden units for each dressed weight instead of the 2^3 shown above. This makes $3 \cdot 3 \cdot 4 = 36$ weights for the dressed bias terms, $3 \cdot 3 \cdot 5 = 45$ connections for the second order dressed weights and $3 \cdot 3 \cdot 6 = 54$ for the third order dressed links; for a total of 135 weights.

Again, we begin by analyzing the set of weights $G_\sigma^{(n)}$ that results from decimating all the hidden units in the neural network and joining the resulting weights by parallel

Inputs			Outputs			Probabilities
S_{i_1}	S_{i_2}	S_{i_3}	S_{o_1}	S_{o_2}	S_{o_3}	$p\left(S_{o_1}, S_{o_2}, S_{o_3} \mid S_{i_1}, S_{i_2}, S_{i_3}\right)$
-1	-1	-1	-1	-1	-1	0.15
-1	-1	-1	-1	-1	1	0.15
-1	-1	-1	-1	1	-1	0.25
-1	-1	-1	-1	1	1	0.05
-1	-1	-1	1	-1	-1	0.10
-1	-1	-1	1	-1	1	0.10
-1	-1	-1	1	1	-1	0.15
-1	-1	-1	1	1	1	0.05
-1	1	-1	-1	-1	-1	0.10
-1	1	-1	-1	-1	1	0.15
-1	1	-1	-1	1	-1	0.30
-1	1	-1	-1	1	1	0.05
-1	1	-1	1	-1	-1	0.15
-1	1	-1	1	-1	1	0.15
-1	1	-1	1	1	-1	0.05
-1	1	-1	1	1	1	0.05
1	-1	1	-1	-1	-1	0.05
1	-1	1	-1	-1	1	0.05
1	-1	1	-1	1	-1	0.05
1	-1	1	-1	1	1	0.05
1	-1	1	1	-1	-1	0.20
1	-1	1	1	-1	1	0.20
1	-1	1	1	1	-1	0.05
1	-1	1	1	1	1	0.15

Table 5.27: Non-exhaustive output probability distribution for a 3 input 3 output BM.

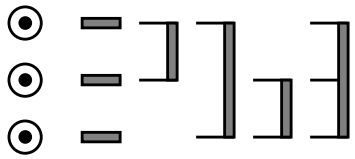

Figure 5.27: Structure with three output units and three input neurons for a non-exhaustive probability distribution with its required dressed weights.

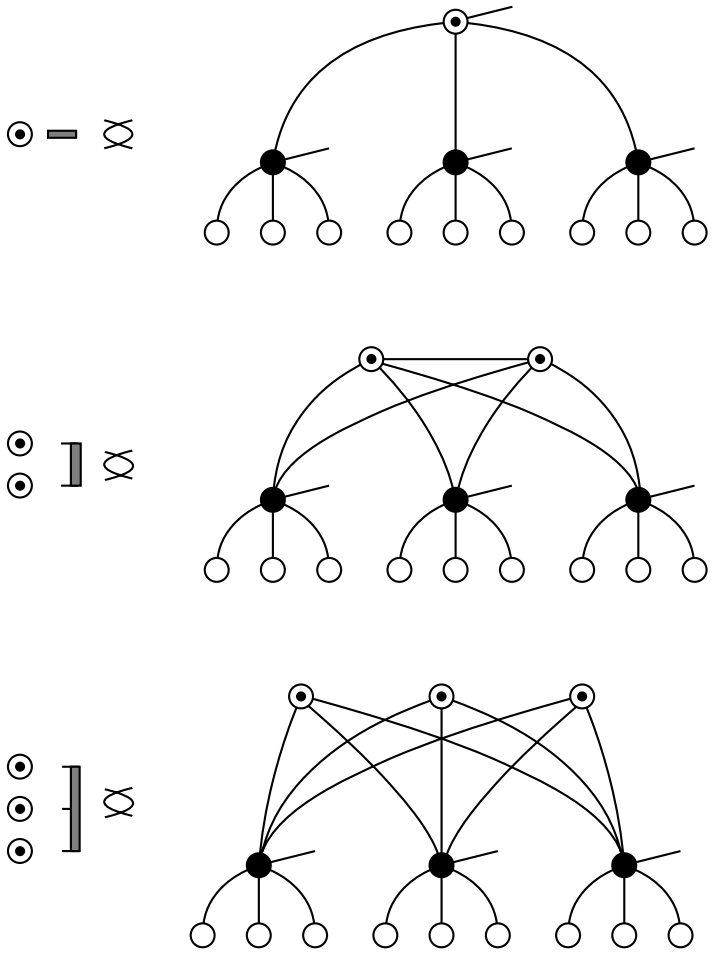

Figure 5.28: Dressed weights equivalence for the 3 inputs, 3 outputs example.

association. Assuming that $J^{(2)}_{h_j o_k} = J$, $|J^{(2)}_{i_j h_k}| = W$ and $J^{(1)}_{h_j} = -3W + d_{h_j}$, we recall

Eqs. 5.91, 5.93 and 5.114 which are now written down as

$$\begin{aligned}
\tilde{G}^{(3)}_{o_1 o_2 o_3} &= -\frac{d_{h_{j_1}}}{2} \ , \\
\tilde{G}^{(2)}_{o_1 o_2} &= \frac{J}{2} \ , \\
\tilde{G}^{(2)}_{o_1 o_3} &= \frac{J}{2} \ , \\
\tilde{G}^{(2)}_{o_2 o_3} &= \frac{J}{2} \ , \\
\tilde{G}^{(1)}_{o_1} &= \frac{d_{h_{j_1}}}{2} \ , \\
\tilde{G}^{(1)}_{o_2} &= \frac{d_{h_{j_1}}}{2} \ , \\
\tilde{G}^{(1)}_{o_3} &= \frac{d_{h_{j_1}}}{2} \ ,
\end{aligned} \tag{5.130}$$

for the active neuron S_{h_j}. The contribution from the other two hidden, inactive neurons becomes

$$\begin{aligned}
\hat{G}^{(3)}_{o_1 o_2 o_3} &= 0 \ , \\
\hat{G}^{(2)}_{o_1 o_2} &= 0 \ , \\
\hat{G}^{(2)}_{o_1 o_3} &= 0 \ , \\
\hat{G}^{(2)}_{o_2 o_3} &= 0 \ , \\
\hat{G}^{(1)}_{o_1} &= -2J \ , \\
\hat{G}^{(1)}_{o_2} &= -2J \ , \\
\hat{G}^{(1)}_{o_3} &= -2J \ .
\end{aligned} \tag{5.131}$$

Now we consider the hidden units that connect each pair of output units as

$$\begin{aligned}
\check{G}^{(2)}_{o_1o_2} &= J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{j_2}}\right) \ , \\
\check{G}^{(2)}_{o_1o_3} &= J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{j_3}}\right) \ , \\
\check{G}^{(2)}_{o_2o_3} &= J - \frac{1}{2}\ln 2 - \frac{1}{4}\ln\cosh^2\left(d_{h_{j_4}}\right) \ , \\
\check{G}^{(1)}_{o_1} &= \frac{d_{j_2}}{2} + \frac{d_{j_3}}{2} - 4J \ , \\
\check{G}^{(1)}_{o_2} &= \frac{d_{j_2}}{2} + \frac{d_{j_4}}{2} - 4J \ , \\
\check{G}^{(1)}_{o_3} &= \frac{d_{j_3}}{2} + \frac{d_{j_4}}{2} - 4J \ ,
\end{aligned} \tag{5.132}$$

Finally, we write down the values that emulate the original boolean building system as

$$\begin{aligned}
\bar{G}^{(1)}_{o_1} &= d_{h_{j_5}} - 2J \ , \\
\bar{G}^{(1)}_{o_2} &= d_{h_{j_6}} - 2J \ , \\
\bar{G}^{(1)}_{o_3} &= d_{h_{j_7}} - 2J \ ,
\end{aligned} \tag{5.133}$$

and now we associate the weights $\tilde{G}^{(n)}_{\sigma}$, $\hat{G}^{(n)}_{\sigma}$, $\check{G}^{(n)}_{\sigma}$, $\bar{G}^{(n)}_{\sigma}$ and the previously existing $J^{(n)}_{\sigma}$ set as $G^{(n)}_{\sigma} = J^{(n)}_{\sigma} + \tilde{G}^{(n)}_{\sigma} + \hat{G}^{(n)}_{\sigma} + \check{G}^{(n)}_{\sigma} + \bar{G}^{(n)}_{\sigma}$ to reach the following equations

$$\begin{aligned}
G^{(3)}_{o_1o_2o_3} &= -\frac{1}{2}d_{h_{j_1}} \ , \\
G^{(2)}_{o_1o_2} &= J^{(2)}_{o_1o_2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_{j_2}}\right) - \frac{1}{2}J \ , \\
G^{(2)}_{o_1o_3} &= J^{(2)}_{o_1o_3} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_{j_3}}\right) - \frac{1}{2}J \ , \\
G^{(2)}_{o_2o_3} &= J^{(2)}_{o_2o_3} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_{j_4}}\right) - \frac{1}{2}J \ , \\
G^{(1)}_{o_1} &= J^{(1)}_{o_1} + \frac{d_{j_2}}{2} + \frac{d_{j_3}}{2} + d_{h_{j_5}} - 8J + \frac{1}{2}d_{h_{j_1}} \ , \\
G^{(1)}_{o_2} &= J^{(1)}_{o_2} + \frac{d_{j_2}}{2} + \frac{d_{j_4}}{2} + d_{h_{j_6}} - 8J + \frac{1}{2}d_{h_{j_1}} \ , \\
G^{(1)}_{o_3} &= J^{(1)}_{o_3} + \frac{d_{j_3}}{2} + \frac{d_{j_4}}{2} + d_{h_{j_7}} - 8J + \frac{1}{2}d_{h_{j_1}} \ .
\end{aligned} \tag{5.134}$$

At this point we should now solve the backwards problem for each one of the output

vectors. For the input vector $S_{i_1} = S_{i_2} = S_{i_3} = -1$ these read as

$$\begin{aligned}
\ln 0.15 &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.15 &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.25 &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.10 &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.10 &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.15 &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ ,
\end{aligned} \tag{5.135}$$

while for input vector $S_{i_1} = -S_{i_2} = S_{i_3} = -1$ we obtain

$$\begin{aligned}
\ln 0.10 &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.15 &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.30 &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.15 &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.15 &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ .
\end{aligned} \tag{5.136}$$

Finally, the system of equations for $S_{i_1} = -S_{i_2} = S_{i_3} = 1$ is

$$\begin{aligned}
\ln 0.05 &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} - G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} - G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.20 &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} - G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.20 &= G^{(0)} + G^{(1)}_{o_1} - G^{(1)}_{o_2} + G^{(1)}_{o_3} - G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.05 &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} - G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} - G^{(2)}_{o_1o_3} - G^{(2)}_{o_2o_3} - G^{(3)}_{o_1o_2o_3} \ , \\
\ln 0.15 &= G^{(0)} + G^{(1)}_{o_1} + G^{(1)}_{o_2} + G^{(1)}_{o_3} + G^{(2)}_{o_1o_2} + G^{(2)}_{o_1o_3} + G^{(2)}_{o_2o_3} + G^{(3)}_{o_1o_2o_3} \ .
\end{aligned} \tag{5.137}$$

The results for these systems of equations can be seen in table 5.28.

$G^{(n)}_{\sigma}$	$S_{i_1} = S_{i_2} = S_{i_3} = -1$	$S_{i_1} = -S_{i_2} = S_{i_3} = -1$	$S_{i_1} = -S_{i_2} = S_{i_3} = 1$
$G^{(1)}_{o_1}$	-0.1652	-0.1733	0.4839
$G^{(1)}_{o_2}$	-0.1094	-0.2747	-0.2092
$G^{(1)}_{o_3}$	-0.3385	-0.1733	0.1373
$G^{(2)}_{o_1o_2}$	0.0375	-0.2747	-0.2092
$G^{(2)}_{o_1o_3}$	0.0639	0.1733	0.1373
$G^{(2)}_{o_2o_3}$	-0.3385	-0.2747	0.1373
$G^{(3)}_{o_1o_2o_3}$	-0.0639	-0.2747	-0.1373

Table 5.28: Solution to the backwards problems for the three inputs, three outputs non-exhaustive system.

Now we take Eq. 5.134 to find the set of weights $J^{(n)}_{\sigma}$, and write it down for the

$S_{i_1} = S_{i_2} = S_{i_3} = -1$ configuration

$$
\begin{aligned}
G^{(3)}_{o_1 o_2 o_3} &= -\frac{d_{h_1}}{2} \ , \\
G^{(2)}_{o_1 o_2} &= J^{(2)}_{o_1 o_2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_2}\right) - \frac{J}{2} \ , \\
G^{(2)}_{o_1 o_3} &= J^{(2)}_{o_1 o_3} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_3}\right) - \frac{J}{2} \ , \\
G^{(2)}_{o_2 o_3} &= J^{(2)}_{o_2 o_3} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_4}\right) - \frac{J}{2} \ , \\
G^{(1)}_{o_1} &= J^{(1)}_{o_1} + \frac{d_2}{2} + \frac{d_3}{2} + d_{h_5} - 8J + \frac{d_{h_1}}{2} \ , \\
G^{(1)}_{o_2} &= J^{(1)}_{o_2} + \frac{d_2}{2} + \frac{d_4}{2} + d_{h_6} - 8J + \frac{d_{h_1}}{2} \ , \\
G^{(1)}_{o_3} &= J^{(1)}_{o_3} + \frac{d_3}{2} + \frac{d_4}{2} + d_{h_7} - 8J + \frac{d_{h_1}}{2} \ ,
\end{aligned}
\tag{5.138}
$$

so we can fix $J^{(1)}_{o_1} = J^{(1)}_{o_2} = J^{(1)}_{o_3} = 8J$. Now we fix the second order terms; we begin with $J^{(2)}_{o_1 o_2}$ as we did in the previous example

$$
\begin{aligned}
0.0375 &= J^{(2)}_{o_1 o_2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_2}\right) - \frac{J}{2} \ , \\
-0.2747 &= J^{(2)}_{o_1 o_2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_9}\right) - \frac{J}{2} \ , \\
-0.2092 &= J^{(2)}_{o_1 o_2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\cosh\left(d_{h_{16}}\right) - \frac{J}{2} \ ,
\end{aligned}
\tag{5.139}
$$

we obtain

$$
\begin{aligned}
\cosh\left(d_{h_2}\right) &= \mathrm{e}^{2J^{(2)}_{o_1 o_2} - \ln 2 - 0.075 - J} \geq 1 \ , \\
\cosh\left(d_{h_9}\right) &= \mathrm{e}^{2J^{(2)}_{o_1 o_2} - \ln 2 + 0.5494 - J} \geq 1 \ , \\
\cosh\left(d_{h_{16}}\right) &= \mathrm{e}^{2J^{(2)}_{o_1 o_2} - \ln 2 + 0.4148 - J} \geq 1 \ ,
\end{aligned}
\tag{5.140}
$$

so we have to satisfy the following constraints

$$
\begin{aligned}
2J^{(2)}_{o_1 o_2} - \ln 2 - 0.075 - J &\geq 0 \ , \\
2J^{(2)}_{o_1 o_2} - \ln 2 + 0.5494 - J &\geq 0 \ , \\
2J^{(2)}_{o_1 o_2} - \ln 2 + 0.4148 - J &\geq 0 \ ,
\end{aligned}
\tag{5.141}
$$

being the most restrictive solution $2J^{(2)}_{o_1 o_2} = \ln 2 + 0.075 + J$, hence $J^{(2)}_{o_1 o_2} = \frac{\ln 2}{2} + 0.0375 + \frac{J}{2}$.

Now we find

$$\begin{aligned} d_{h_2} &= 0 \ , \\ d_{h_9} &= \text{acoshe}^{0.075+0.5494} = 1.237 \ , \\ d_{h_{16}} &= \text{acoshe}^{0.075+0.4148} = 1.072 \ , \end{aligned} \tag{5.142}$$

and repeat this process with the remaining $J^{(2)}_{o_1o_3}$, $J^{(2)}_{o_2o_3}$, d_{h_3}, d_{h_4}, $d_{h_{10}}$, $d_{h_{11}}$, $d_{h_{17}}$ and $d_{h_{18}}$ terms

$$\begin{aligned} J^{(2)}_{o_1o_3} &= \tfrac{\ln 2}{2} - 0.0320 + \tfrac{J}{2} \ , & J^{(2)}_{o_2o_3} &= \tfrac{\ln 2}{2} + 0.169 + \tfrac{J}{2} \ , \\ d_{h_3} &= 0 \ , & d_{h_4} &= 0 \ , \\ d_{h_{10}} &= 0.476 \ , & d_{h_{11}} &= 0.361 \ , \\ d_{h_{17}} &= 0.389 \ , & d_{h_{18}} &= 1.054 \ . \end{aligned} \tag{5.143}$$

We proceed now with the d_{h_1}, d_{h_8} and the $d_{h_{15}}$ terms, which are directly found by using the third order term that one can obtain through the backwards problem solution $G^{(3)}_{o_1o_2o_3} = -\frac{1}{2}d_{h_j}$, and so

$$\begin{aligned} d_{h_1} &= 0.1278 \ , \\ d_{h_8} &= 0.5494 \ , \\ d_{h_{15}} &= 0.2746 \ . \end{aligned} \tag{5.144}$$

We conclude the example by finding the proper values for the terms d_{h_5}, d_{h_6}, d_{h_7}, $d_{h_{12}}$,

$d_{h_{13}}$, $d_{h_{14}}$, $d_{h_{19}}$, $d_{h_{20}}$ and $d_{h_{21}}$ through the following equations

$$\begin{aligned}
-0.1652 &= J_{o_1}^{(1)} + \frac{d_{h_2}}{2} + \frac{d_{h_3}}{2} + d_{h_5} - 8J + \frac{d_{h_1}}{2} \ , \\
-0.1094 &= J_{o_2}^{(1)} + \frac{d_{h_2}}{2} + \frac{d_{h_4}}{2} + d_{h_6} - 8J + \frac{d_{h_1}}{2} \ , \\
-0.3385 &= J_{o_3}^{(1)} + \frac{d_{h_3}}{2} + \frac{d_{h_4}}{2} + d_{h_7} - 8J + \frac{d_{h_1}}{2} \ , \\
-0.1733 &= J_{o_1}^{(1)} + \frac{d_{h_9}}{2} + \frac{d_{h_{10}}}{2} + d_{h_{12}} - 8J + \frac{d_{h_8}}{2} \ , \\
-0.2747 &= J_{o_2}^{(1)} + \frac{d_{h_9}}{2} + \frac{d_{h_{11}}}{2} + d_{h_{13}} - 8J + \frac{d_{h_8}}{2} \ , \\
-0.1733 &= J_{o_3}^{(1)} + \frac{d_{h_{10}}}{2} + \frac{d_{h_{11}}}{2} + d_{h_{14}} - 8J + \frac{d_{h_8}}{2}, \\
0.4839 &= J_{o_1}^{(1)} + \frac{d_{h16}}{2} + \frac{d_{h_{17}}}{2} + d_{h_{19}} - 8J + \frac{d_{h_{15}}}{2} \ , \\
-0.2092 &= J_{o_2}^{(1)} + \frac{d_{h_{16}}}{2} + \frac{d_{h_{18}}}{2} + d_{h_{20}} - 8J + \frac{d_{h_{15}}}{2} \ , \\
0.1373 &= J_{o_3}^{(1)} + \frac{d_{h_{17}}}{2} + \frac{d_{h_{18}}}{2} + d_{h_{21}} - 8J + \frac{d_{h_{15}}}{2} \ ,
\end{aligned} \tag{5.145}$$

and so

$$\begin{aligned}
d_{h_5} &= -0.2291 \ , \\
d_{h_6} &= -0.1733 \ , \\
d_{h_7} &= -0.4024 \ , \\
d_{h_{12}} &= -1.3045 \ , \\
d_{h_{13}} &= -1.3484 \ , \\
d_{h_{14}} &= -0.8665 \ , \\
d_{h_{19}} &= -0.3839 \ , \\
d_{h_{20}} &= -1.4095 \ , \\
d_{h_{21}} &= -0.7215 \ .
\end{aligned} \tag{5.146}$$

Notice however that to complete the example it requires J and W to be fixed. To this purpose, we define an arbitrary error $\varepsilon_z \leq 0.01$, and choose $J = 11.2$ and $W = 20.9$, according again to Eq. 5.110.

Chapter 6

Summary and conclusions

In this work, we have studied new aspects of the learning process, the dynamics and the capacity of Boltzmann Machines (BM) and their extension to High Order Boltzmann Machines (HOBM) where weights can connect more than two units at a time. The Boltzmann Machine neural network is a system with the ability of learning and extrapolating probability distributions. However, the exhaustive computational cost and the large amount of time associated to the learning process have prevented widespread usage of this model. Though there are several authors who have proposed different methods to reduce the learning time, the BM is still better known as the parallel implementation of the Simulated Annealing (SA) algorithm than as a neural network useful in solving practical, real-life problems. Up to now, the existing relations between a BM, a probability distribution (p.d.f.) and a High Order Boltzmann Machine were the ones shown in Fig. 6.1.

In its standard form, learning in Boltzmann Machines is carried out using a gradient descent algorithm where in each iteration weights are modified according to the update rules

$$\begin{aligned} \Delta w_i^{(1)} &\propto \left(\langle S_i \rangle^* - \langle S_i \rangle\right) \ , \\ \Delta w_{ij}^{(2)} &\propto \left(\langle S_i S_j \rangle^* - \langle S_i S_j \rangle\right) \ . \end{aligned} \tag{6.1}$$

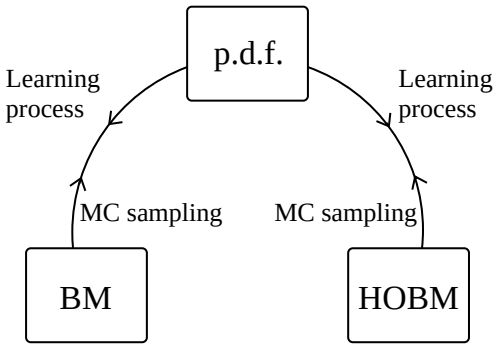

Figure 6.1: The BM, its probability distribution and the HOBM.

Being a gradient descent algorithm, this leads to the local minimum of the Kullback-Leibler distance closest to the departing state. Though this is the standard procedure, statistically exact methods such as Simulated Annealing optimization could be used to achieve better results. These considerations also apply to the High Order Boltzmann Machine, whose weights are updated according to the rule

$$\Delta w^{(3)}_{i_1 i_2 i_3 \ldots} \propto \left(\langle S_{i_1} S_{i_3} S_{i_3} \cdots \rangle^* - \langle S_{i_1} S_{i_3} S_{i_3} \cdots \rangle\right) \quad . \tag{6.2}$$

On the other hand, if the weights of a BM are known the p.d.f. reproduced by the network can estimated by Monte Carlo (MC) simulation employing the Metropolis algorithm. Once again, the same thing applies to the HOBM model.

So far this describes the state of the art prior to this work. In this thesis we have made some extensions of that, introducing new relevant aspects that improve the performance of the dynamics and learning in BMs and HOBMs.

In chapter 3 the original decimation algorithm that was presented in Ref. [Saul and Jordan, 1994] and further extended in Ref. [Rüger et al., 1996] is described. Decimation was conceived as a procedure that can be used in sparsely connected BMs to analytically compute the statistical moments of Eq. 6.1. Decimation of a unit is a process that eliminates it and produces a new network with one less neuron and additional connections between the remaining ones, keeping their probability distribution unaffected. Decimation of several units was obtained by successive application of this algorithm to each one of the units to be removed. A serious drawback of the method proposed was that it could

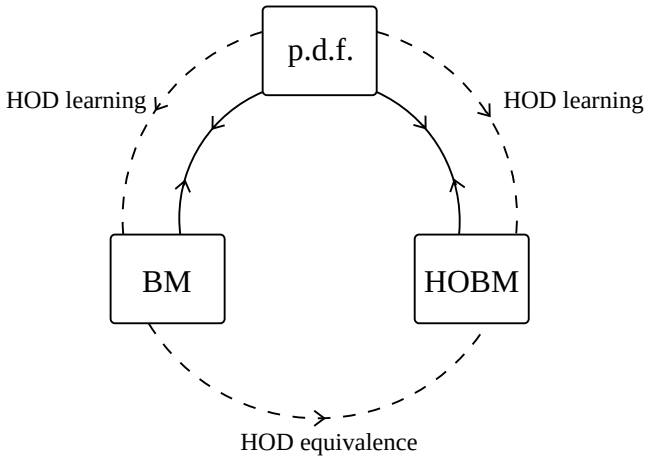

Figure 6.2: New connections between the BM and HOBM, due to the HOD equivalence; and the p.d.f. and the BM/HOBM models when using HOD to carry out a learning process.

not be used when units are connected to more than three other units. We have derived an extension of this procedure that overcomes this problem, thus allowing the unit to be decimated to connect to an arbitrary number of units in the network. This has been referred to as *High Order Decimation* (HOD), where the moments required to update the weights of any BM or HOBM can be computed analytically (instead of using the standard MC based algorithm), at the expense of producing as a result high order weights connecting the remaining units. When used on a HOBM, this method is more precise that other popular algorithms like the high order Mean Field (MF) approximation proposed in Ref. [Kuroki et al., 1999]. In this sense, and as a first contribution, we have added additional links relating the p.d.f. to both the BM and the HOBM to the scheme of Fig. 6.1, as shown in Fig. 6.2.

In particular, High Order Decimation can be used to decimate all hidden units in a BM, producing a HOBM with only visible units. In this way, we have shown that hidden units in a BM can be replaced by a set of high order connections, keeping the probability distribution of the visible units unaffected. This is schematically represented in Fig. 6.3. This process relates the BM and the HOBM as shown by the lower link in Fig. 6.2.

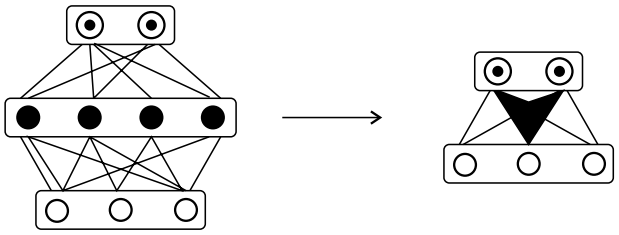

Figure 6.3: Schematic representation of the equivalence between hidden units in a BM and high order weights (represented as a solid pattern) on a HOBM.

Furthermore, an extension of the decimation process, where any number of units is decimated at once, has also been proposed and checked to work. This method, referred to as *Multiple Decimation* (MD), is also shown to be faster than reiterated application of the HOD procedure. HOD and MD lead to the same decimated system. Additionally, we have also tested the efficiency of the HOD method when applied to the learning process of a BM in classifying problems. When compared to other well-known classifying algorithms, the HOD method is shown to be competitive at least in accuracy. With this method we have solved real-life problems such as the *balance* and the *tic-tac-toe* from Ref. [Newman et al., 1998], the *gene* problem from Ref. [Prechelt, 1994], and the benchmarking *Monk's problem* from Ref. [Thrun et al., 1991].

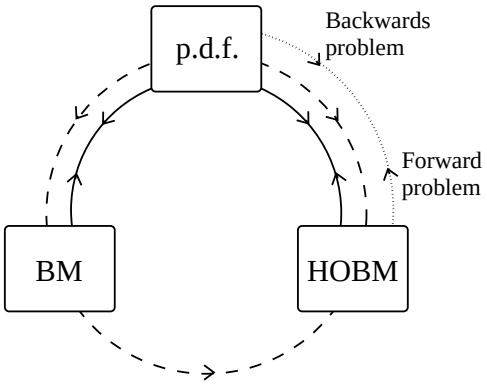

Figure 6.4: New link established between the analysis of the p.d.f. and the HOBM model.

In chapter 4 we have discussed a representation of the set of equations connecting the high order weights of the network and the p.d.f. implemented by a HOBM, in terms of

Hadamard matrices. This representation turns out to be quite useful since the particular properties of Hadamard matrices allow for a neat determination of the weights of the network when the whole p.d.f. of the system is known. This has been referred to as the *backwards problem*, which is the inverse of the much simpler *forward problem* where one knows all the weights of the network and computes the values of the probabilities. The backwards problem has been shown to be exactly solvable for a p.d.f. that is fully known, that is, for a p.d.f. where one knows the probability of every state of the network. However when only some probabilities are known (as happens in real problems), there is an infinite set of solutions that can reproduce them. We have not discussed in detail the whole family of solutions, but have analyzed a specific solution based on an LU factorization of the Hadamard matrix of the system. This establishes an additional link between the p.d.f. to be learnt and the HOBM model as shown with a dotted line in Fig. 6.4.

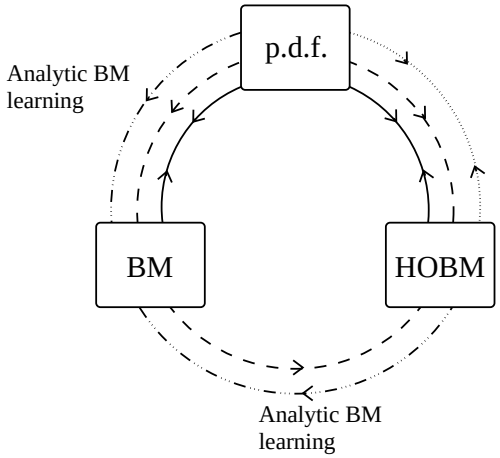

Figure 6.5: New link established between the analysis of the p.d.f. and the HOBM model.

Finally, in chapter 5 a specific BM with a fixed topology has been devised in such a way that one can directly find the values of the weights linking the different units when the whole p.d.f. of the system is know. In this sense, a solution to the backwards problem for that specific topology has been given. This topology has been adopted in order to prove the existing equivalence between hidden units in a BM and high order weights in a HOBM. Starting from the known p.d.f. a HOBM is built by solving the corresponding backwards

problem. Once the high order weights are known, the high order decimation equations for the adopted BM are inverted and the associated two-body weights are obtained. In this way, a second order BM with a fixed topology is shown to be able to reproduce any p.d.f. that does not assign zero probability to any state.

Appendix

Properties of Hadamard matrices

In this appendix, we briefly describe what Hadamard matrices are and how they are related to the systems of equations that are discussed in this thesis. We will only discuss the concepts that are needed to understand the ideas presented in the text.

This appendix has been structured in two sections: first, we discuss Hadamard matrices in general, and their most relevant mathematical properties. The next section focuses on the specific type of Hadamard matrices that are used in this work, we also discuss an alternative Hadamard matrix creation rule that better suits our requirements, and prove that they can be generated with a slight modification of the general recursive rule.

A.1 General properties of Hadamard matrices

Hadamard matrices were first presented by J. J. Sylvester in Ref. [Sylvester, 1867]. They are binary valued square $m \times m$ matrices H_m with $\{-1, +1\}$ entries whose rows are pairwise orthogonal [Hedayat and Wallis, 1978], in other words

$$H_m \cdot H_m^T = mI \ . \tag{A.1}$$

Consequently, one immediately derives the following properties

$$\det\{H_{m\times m}\} \neq 0 \ , \tag{A.2}$$

$$H^{-1}_{m\times m} = \frac{1}{m} H^{T}_{m\times m} \ , \tag{A.3}$$

$$H_{m\times m} \cdot H^{T}_{m\times m} = H^{T}_{m\times m} \cdot H_{m\times m} = mI \ , \tag{A.4}$$

I being the identity matrix. Property A.2 is a direct consequence of the fact that all row vectors are orthogonal. Furthermore, since the determinant of the product of square matrices equals the product of their respective determinants, and the determinant of the transpose equals the determinant of the original matrix, one readily infers that

$$\det\{H_{m\times m}\} = \pm\sqrt{m} \ . \tag{A.5}$$

In this way, $H^{-1}_{m\times m}$ is guaranteed to exist. On the other hand, multiplying Eq. A.1 on the left by $H^{-1}_{m\times m}$ we find property A.3. We now multiply Eq. A.3 on the right by $H_{m\times m}$

$$I = \frac{1}{m} H^{T}_{m\times m} \cdot H_{m\times m} \ , \tag{A.6}$$

thus proving Eq. A.4.

It is not obvious that Hadamard matrices of any dimensionality do actually exist. It has been conjectured in Ref. [Paley, 1933] that, if $m = 1$, $m = 2$ or either m is divisible by 4 and is of the form

$$m = 2^c \left(p^k + 1\right) \quad k, c \in \mathbb{Z} \ , \ \ k > 0 \ , \ \ c \geq 0 \ , \tag{A.7}$$

where p is a prime number different from 2, there always exist Hadamard matrices of order m. Still, Hadamard matrices for $m = 2^k$, $\forall k \in \mathbb{Z} \geq 0$ are confirmed to exist [Hedayat and Wallis, 1978]. There are several operations that can be carried out on a Hadamard matrix which will still preserve the Hadamard properties [Orrick, 2008], and therefore yield another Hadamard matrix. Examples of such operations are transposition, the permutation of rows or columns, or changing the sign of any number of rows or columns.

Hadamard matrices are nowadays extensively used in cryptography [Lipmaa, 2002], error detection [Fenwick et al., 1977], spectrography [Gentry et al., 2006], modulation

[Nyström and Popovic, 1998] and signal correlation [Horadam, 2006]. They are generated in many different ways, depending on the value of m [Kharaghani and Tayfeh-Rezaie, 2004, Bouyukliev et al., 2005, Doković, 2008]. In this thesis, we are only interested in values of the form $m = 2^N$, N being (possibly a subset of) the number of neurons in the network. These Hadamard matrices can be built via the Sylvester rule [Sylvester, 1867]

$$H_{2m} = \begin{pmatrix} H_m & H_m \\ H_m & -H_m \end{pmatrix} \ , \ H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \ . \tag{A.8}$$

We now show by induction that any Hadamard matrix that is generated through this rule satisfies Eq. A.2. It is obvious that the row vectors of H_2 are orthogonal. We now assume that all row vectors in H_m are orthogonal, and show that this is also true for H_{2m}. We start assuming that the row vectors in H_m are orthogonal. Take two row vectors $\vec{u}_i^{(2m)}$ and $\vec{u}_j^{(2m)}$ from H_{2m} with $i \neq j$. These are of the general form

$$\vec{u}_i^{(2m)} = \begin{pmatrix} \vec{u}^{(m)} \\ \pm\vec{u}^{(m)} \end{pmatrix} \ , \ \vec{u}_j^{(2m)} = \begin{pmatrix} \vec{v}^{(m)} \\ \pm\vec{v}^{(m)} \end{pmatrix} \ , \tag{A.9}$$

with $\vec{u}^{(m)} \neq \vec{v}^{(m)}$ and $\vec{u}^{(m)} \cdot \vec{v}^{(m)} = 0$, since $\vec{u}^{(m)}$ and $\vec{v}^{(m)}$ are orthogonal vectors from H_m. Then

$$\vec{u}_i^{(2m)} \cdot \vec{u}_j^{(2m)} = \vec{u}^{(m)} \cdot \vec{v}^{(m)} \pm \vec{u}^{(m)} \cdot \vec{v}^{(m)} = 0 \ . \tag{A.10}$$

We have shown that Sylvester's rule always produces Hadamard matrices.

A.2 Use of Hadamard matrices in HOBMs

In this work, we have used Hadamard matrices to build the systems of equations that define the HOBM model. However, we do not use Sylvester's rule directly. In this section, we write explicitly the equations of the HOD process and the backwards problem using Hadamard matrices. We begin this discussion analyzing a simple example consisting in the serial association of a two-unit network, as shown in Fig. A.1a, which after decimation produces A.1b.

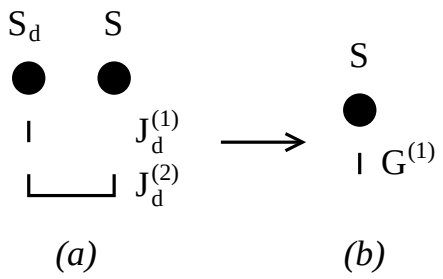

Figure A.1: Serial association to obtain a bias term.

The decimation equations for S_d are

$$\begin{aligned} \ln\cosh\left(J_d^{(1)} + J_d^{(2)}\right) &= G^{(0)} + G^{(1)} \qquad \text{for} \quad S = 1, \\ \ln\cosh\left(J_d^{(1)} - J_d^{(2)}\right) &= G^{(0)} - G^{(1)} \qquad \text{for} \quad S = -1. \end{aligned} \tag{A.11}$$

In matrix notation this system becomes

$$\begin{pmatrix} \ln\cosh\left(J_d^{(1)} + J_d^{(2)}\right) \\ \ln\cosh\left(J_d^{(1)} - J_d^{(2)}\right) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} G^{(0)} \\ G^{(1)} \end{pmatrix} \ , \tag{A.12}$$

where $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ fulfills properties A.2 to A.3 and is therefore of the Hadamard type. Actually this matrix is H_2 in Sylvester rule. Notice that Eq. A.12 could also be represented in matrix form using the alternative matrix $H_{2\times 2} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$ by simply reversing the order of the equations in Eq. A.11. Since this matrix does also fulfill properties A.2 to A.3, it is of the Hadamard type. We thus see that one can find at least two different Hadamard matrices describing the system. As we shall see below, this statement is general and applies to all the networks analyzed in this text.

Now we work out the first non-trivial example, corresponding to the star-triangle transformation shown in Fig. A.2. The system of equations related to the decimation

process of unit S_d reads

$$\begin{aligned}
&\ln\cosh\left(J_d^{(1)} - J_{d1}^{(2)} - J_{d2}^{(2)} - J_{d3}^{(2)}\right) = A_0 = \\
&\quad = G^{(0)} - G_1^{(1)} - G_2^{(1)} - G_3^{(1)} + G_{12}^{(2)} + G_{13}^{(2)} + G_{23}^{(2)} - G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} - J_{d1}^{(2)} - J_{d2}^{(2)} + J_{d3}^{(2)}\right) = A_1 = \\
&\quad = G^{(0)} - G_1^{(1)} - G_2^{(1)} + G_3^{(1)} + G_{12}^{(2)} - G_{13}^{(2)} - G_{23}^{(2)} + G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} - J_{d1}^{(2)} + J_{d2}^{(2)} - J_{d3}^{(2)}\right) = A_2 = \\
&\quad = G^{(0)} - G_1^{(1)} + G_2^{(1)} - G_3^{(1)} - G_{12}^{(2)} + G_{13}^{(2)} - G_{23}^{(2)} + G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} - J_{d1}^{(2)} + J_{d2}^{(2)} + J_{d3}^{(2)}\right) = A_3 = \\
&\quad = G^{(0)} - G_1^{(1)} + G_2^{(1)} + G_3^{(1)} - G_{12}^{(2)} - G_{13}^{(2)} + G_{23}^{(2)} - G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} + J_{d1}^{(2)} - J_{d2}^{(2)} - J_{d3}^{(2)}\right) = A_4 = \\
&\quad = G^{(0)} + G_1^{(1)} - G_2^{(1)} - G_3^{(1)} - G_{12}^{(2)} - G_{13}^{(2)} + G_{23}^{(2)} + G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} + J_{d1}^{(2)} - J_{d2}^{(2)} + J_{d3}^{(2)}\right) = A_5 = \\
&\quad = G^{(0)} + G_1^{(1)} - G_2^{(1)} + G_3^{(1)} - G_{12}^{(2)} + G_{13}^{(2)} - G_{23}^{(2)} - G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} + J_{d1}^{(2)} + J_{d2}^{(2)} - J_{d3}^{(2)}\right) = A_6 = \\
&\quad = G^{(0)} + G_1^{(1)} + G_2^{(1)} - G_3^{(1)} + G_{12}^{(2)} - G_{13}^{(2)} - G_{23}^{(2)} - G_{123}^{(3)} \\
&\ln\cosh\left(J_d^{(1)} + J_{d1}^{(2)} + J_{d2}^{(2)} + J_{d3}^{(2)}\right) = A_7 = \\
&\quad = G^{(0)} + G_1^{(1)} + G_2^{(1)} + G_3^{(1)} + G_{12}^{(2)} + G_{13}^{(2)} + G_{23}^{(2)} + G_{123}^{(3)} \ , \qquad \text{(A.13)}
\end{aligned}$$

where A_γ stands for

$$\begin{aligned}
&A_\gamma = \qquad \text{(A.14)} \\
&\quad = G^{(0)} + G_1^{(1)} S_1 + G_2^{(1)} S_2 + G_3^{(1)} S_3 + G_{12}^{(2)} S_1 S_2 + G_{13}^{(2)} S_1 S_3 + G_{23}^{(2)} S_2 S_3 + G_{123}^{(3)} S_1 S_2 S_3 \ .
\end{aligned}$$

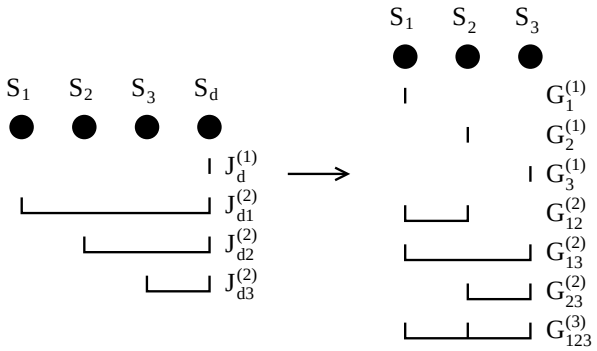

Figure A.2: High order star-triangle association.

This system of equations can also be expressed in matrix form as

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{pmatrix} = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} G^{(0)} \\ G_1^{(1)} \\ G_2^{(1)} \\ G_3^{(1)} \\ G_{12}^{(2)} \\ G_{13}^{(2)} \\ G_{23}^{(2)} \\ G_{123}^{(3)} \end{pmatrix} . \qquad \text{(A.15)}$$

Notice that the matrix in Eq. A.15 is of the Hadamard type, though it has not directly been generated through Sylvester's rule. This system results from the specific order in which the equations have been written (defining the order of the rows in Eq. A.15) and the order chosen for the different terms entering in each equation (defining the order of the columns in Eq. A.15). One could, for instance, obtain an equivalent matrix representation of the system of equations permuting rows 1 and 3, which would correspond to write, in Eq. A.13, first the third equation, then the second one, then the first one, and finally all the others in the same order they appear. In this way, all permutations of rows are allowed, since they correspond to a different order in which the equations are being presented. Furthermore, one could also change columns in the matrix, and that would correspond to a rearrangement of the different terms entering in the energy functional. The order in

which the terms appear in Eq. A.13 corresponds to writing the energy functional in the *standard* form

$$E = \tag{A.16}$$
$$= G^{(0)} + G_1^{(1)} S_1 + G_2^{(1)} S_2 + G_3^{(1)} S_3 + G_{12}^{(2)} S_1 S_2 + G_{13}^{(2)} S_1 S_3 + G_{23}^{(2)} S_2 S_3 + G_{123}^{(3)} S_1 S_2 S_3 \ ,$$

where one writes first the zero order weight $G^{(0)}$, next all the first order (bias) terms, then the second order (two-body weights) terms, and so on until the last, unique N-th order connection. For instance, a permutation of columns two and five in Eq. A.15 corresponds to changing the second and fifth terms in this previous expression, leading to

$$E = \tag{A.17}$$
$$= G^{(0)} + G_{12}^{(2)} S_1 S_2 + G_2^{(1)} S_2 + G_3^{(1)} S_3 + G_1^{(1)} S_1 + G_{13}^{(2)} S_1 S_3 + G_{23}^{(2)} S_2 S_3 + G_{123}^{(3)} S_1 S_2 S_3 \ ,$$

which obviously represents the same energy. We can now take advantage of the fact that permutation of rows and/or columns are allowed, as we have just discussed, to build an equivalent Hadamard matrix describing this very same system. In the system matrix of Eq. A.15, change rows as follows: $1 \leftrightarrow 8$, $2 \leftrightarrow 7$, $3 \leftrightarrow 6$, and finally $4 \leftrightarrow 5$, to produce

$$\tilde{H}_{2^3 \times 2^3} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 \end{pmatrix} . \tag{A.18}$$

Next change columns as follows: column 2 goes to column 5, column 4 goes to column 2, column 5 goes to column 7, and column 7 goes to column 4. The outcome of the whole

process is

$$H_{2^3\times 2^3} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} , \tag{A.19}$$

which is a Hadamard matrix that has been directly generated through Sylvester rule, as can be easily checked. Once again, the decimation process can be described in terms of a Hadamard matrix.

Now we prove the general statement that the system of equations describing the result of any HOD process can be expressed in terms of a Hadamard matrix. We recall the expression of the decimation equations (single or multiple) corresponding to an N-th order, N units neural network

$$A_\gamma = -J^{(0)} - \sum_{i_1} J^{(1)}_{i_1} S_{i_1} - \sum_{i_1<i_2} J^{(2)}_{i_1 i_2} S_{i_1} S_{i_2} - \sum_{i_1<i_2<i_3} J^{(3)}_{i_1 i_2 i_3} S_{i_1} S_{i_3} S_{i_3} - \dots \ , \tag{A.20}$$

where A_γ stand for the logarithm terms involving the weights of the network prior to decimation. The matrix associated to this system of equations is usually written in the form

$$\hat{H}_{2^N\times 2^N} = \tag{A.21}$$
$$\left[\{1\} \ \{S_1\} \ \{S_2\} \ \dots \ \{S_N\} \ \{S_{i_1} S_{i_2}\} \ \dots \ \left\{S_{i_1} S_{i_2} \cdots S_{i_{N-1}}\right\} \ \{S_1 S_2 \cdots S_N\}\right] \ .$$

The first column $\{1\}$ in $\hat{H}_{2^N\times 2^N}$ represents a column vector with every component set equal to 1. The next N terms $\{S_1\}$, $\{S_2\}$ to $\{S_N\}$ stand for N column vectors of 2^N components each one. Their values are drawn from a 2^N rows $\times$ N columns matrix corresponding to the different N-bit words (organized in rows), representing the integer

numbers $0, 1, 2, \ldots, 2^N - 1$ in ascendant order with the lowest value 0 on top and the highest value $2^N - 1$ on the bottom, and with every 0 replaced by -1. All other elements in $\hat{H}_{2^N \times 2^N}$ are obtained multiplying the values of units $S_{i_1}, S_{i_2}, \ldots, S_{i_N}$ taken from columns 2 to $N+1$ in the same row. In this way and following the previous example corresponding to a neural network with $N = 3$ units, one starts from

	S_1	S_2	S_3		S_1S_2	S_1S_3	S_2S_3	$S_1S_2S_3$
1	-1	-1	-1		1	1	1	-1
1	-1	-1	1		1	-1	-1	1
1	-1	1	-1		-1	1	-1	1
1	-1	1	1	to build	-1	-1	1	-1
1	1	-1	-1		-1	-1	1	1
1	1	-1	1		-1	1	-1	-1
1	1	1	-1		1	-1	-1	-1
1	1	1	1		1	1	1	1

.

Joining all the columns generated in this way one ends up with the full $2^3 \times 2^3$ matrix of Eq. A.19.

All in all, what we end up with is a matrix formed by columns corresponding to all possible products of units, ranging from zero units (this is the first column of 1's) to the product of all the N units (last column). The order in which the different columns appear is irrelevant, and the same applies to the rows. One can build a simple rule that generates iteratively all the required rows and columns. In order to do so, one starts writing a 2×2 matrix $H_{2\times 2}$ corresponding to the elements $\{1\}$ and $\{S_1\}$ with $S_1 = 1$ on top and $S_1 = -1$ on the bottom

$$H_{2\times 2} = \begin{bmatrix} 1 & S_1 \end{bmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{A.22}$$

Now we can describe a system with a second neuron building a $2^2 \times 2^2$ matrix $H_{2^2\times 2^2}$, multiplying $H_{2\times 2}$ by $\{1\}$ and $H_{2\times 2}$ by $\{S_2\}$, with $S_2 = 1$ on top and $S_2 = -1$ on the

bottom

$$
\begin{aligned}
H_{2^2\times 2^2} &= [\, H_{2\times 2}\times 1 \;\; H_{2\times 2}\times S_2 \,] = \left[\, [1 \;\; S_1] \;\; [1 \;\; S_1]S_2 \right] \\
&= \begin{bmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times 1 & \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times (S_2 = 1) \\ \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times 1 & \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times (S_2 = -1) \end{bmatrix} \\
&= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} .
\end{aligned} \tag{A.23}
$$

We can now include a third neuron S_3 to generate $H_{2^3\times 2^3}$, by following the same procedure

$$
\begin{aligned}
H_{2^3\times 2^3} &= [\, H_{2^2\times 2^2}\times 1 \;\; H_{2^2\times 2^2}\times S_3 \,] \\
&= \begin{bmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \times 1 & \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \times (S_3 = 1) \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \times 1 & \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \times (S_3 = -1) \end{bmatrix} ,
\end{aligned} \tag{A.24}
$$

thus arriving to

$$H_{2^3\times 2^3} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} . \tag{A.25}$$

This process can obviously be iterated in order to obtain $H_{2^N\times 2^N}$, which fulfills

$$\begin{aligned} H_{2^N\times 2^N} &= [\ H_{2^{N-1}\times 2^{N-1}} \times 1 \ \ H_{2^{N-1}\times 2^{N-1}} \times S_N\ ] = \\ &= \begin{bmatrix} H_{2^{N-1}\times 2^{N-1}} \times 1 & H_{2^{N-1}\times 2^{N-1}} \times (S_N = 1) \\ H_{2^{N-1}\times 2^{N-1}} \times 1 & H_{2^{N-1}\times 2^{N-1}} \times (S_N = -1) \end{bmatrix} , \end{aligned} \tag{A.26}$$

which matches the Sylvester rule for Hadamard matrix generation. Since $H_{2\times 2}$ is already Hadamard, and Sylvester rule applied to Hadamard matrices is known to produce new Hadamard matrices, $H_{2^N\times 2^N}$ is guaranteed to be of the Hadamard type. Therefore, the system of linear equations that is derived from this algorithm can always be solved. The same argument applies to the equations of the backwards problem that appear in chapter 4.

A.2.1 The Walsh-Hadamard transform

It has been shown that both the HOD method and the backwards problem use Hadamard matrices to carry out the decimation process over a given BM topology. Actually, the Hadamard matrices, other than a set of binary values, are the standard numerical representation of the Walsh functions [Walsh, 1923], as seen in Fig. A.3. This functions conform an orthogonal set that can be used to generate a Fourier-like transform which is known as Walsh-Hadamard transform [Shanks, 1969].

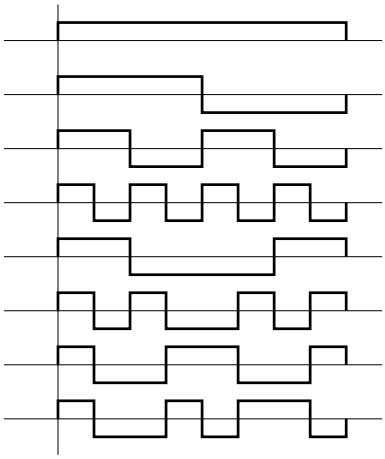

Figure A.3: Walsh functions.

The Walsh-Hadamard transform $W(k)$ of a given sequence $x(t)$ of length 2^N is defined as

$$W(k) = H_{2^N \times 2^N} \cdot x(t) \quad , \tag{A.27}$$

where $H_{2^N \times 2^N}$ is the Hadamard matrix of order 2^N. This transform can be applied to solve complex Boolean functions [Langevin and Zanotti, 2005] and image compression and signal processing [Pichler, 2004], since it returns a numerical sequence comparable to the one generated by a Discrete Fourier transform [Tallia et al., 1984]. In this sense, and close to the Fast Fourier transform algorithm, there is a Fast Walsh Hadamard transform algorithm that can be implemented in order to speed up the process [Shanks, 1969]. Notice now that the high order Decimation algorithm is equivalent to carrying out a WHT over a given sequence, being it the set of weights of the neural network.

Bibliography

[Aarts and Korst, 1987] Aarts, E. and Korst, J. (1987). Boltzmann Machines and their applications. In *Parallel architectures on PARLE: Parallel Architectures and Languages Europe, Volume I*, pages 34–50, London, UK. Springer-Verlag.

[Aarts and Korst, 1989] Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines*. New York: Wiley, 2 edition.

[Abe, 1989] Abe, S. (1989). Theories on the Hopfield neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 557–564.

[Abu-Mostafa and Jacques, 1985] Abu-Mostafa, Y. and Jacques, J. S. (1985). Information capacity of the Hopfield model. *IEEE Transactions on Information Theory*, 31(4):461–464.

[Ackley et al., 1985] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for Boltzmann Machines. *Cognitive Science*, 9:147–169.

[Albizuri et al., 1997] Albizuri, F. X., d'Anjou, A., Graña, M., and Larrañaga, P. (1997). Structure of the high-order Boltzmann Machine from independence maps. *IEEE Transactions on Neural Networks*, 8(6):1351–1358.

[Albizuri et al., 1996] Albizuri, F. X., d'Anjou, A., Graña, M., and Lozano, J. A. (1996). Convergence properties of high-order Boltzmann Machines. *Neural Networks*, 9(9):1561–1567.

[Albizuri et al., 1995] Albizuri, F. X., d'Anjou, A., Graña, M., Torrealdea, F. J., and Hernández, M. C. (1995). The high-order Boltzmann Machine: Learned distribution and topology. *IEEE Transactions on Neural Networks*, 6(3):767–770.

[Amit, 1989] Amit, D. J. (1989). *Modelling brain function: The world of attractor neural networks*. Cambridge University Press, 40 West 20th Street, New York NY 10011-4211, USA.

[Arnold, 1997] Arnold, V. I. (1997). *Mathematical Methods of Classical Mechanics*. Springer.

[Baldi, 1988] Baldi, P. (1988). Neural networks, orientations of the hypercube, and algebraic threshold functions. *IEEE Transactions on Information Theory*, 34(3):523–530.

[Baldi and Venkatesh, 1993] Baldi, P. and Venkatesh, S. S. (1993). Random interactions in higher order neural networks. *IEEE Transactions on Information Theory*, 39(1):274–283.

[Bear et al., 2006] Bear, M. F., Connors, B., and Paradiso, M. (2006). *Neuroscience: Exploring the Brain*. Lippincott Williams and Wilkins; 3Rev Ed edition.

[Beiu et al., 1992] Beiu, V., Ioan, D. C., and Dumbrava, M. C. (1992). Continuous Boltzmann Machines: theoretical aspects and applications. In *Proceedings of the CompEuro '92, Computer Systems and Software Engineering*, pages 193–198.

[Bouyukliev et al., 2005] Bouyukliev, I., Fack, V., and Winne, J. (2005). Hadamard matrices of order 36 and double-even self-dual [72,36,12] codes. In *DMTCS Proceedings, 2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05)*, pages 93–98.

[Boyd et al., 1994] Boyd, S., Ghaoui, L. E., Feron, E., and Balakrishnan, V. (1994). *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM).

[Breedlove et al., 2007] Breedlove, S. M., Rosenzweig, M. R., and Watson, N. V. (2007). *Biological Psychology: An Introduction to Behavioral, Cognitive, and Clinical Neuroscience, Fifth Edition*. Sinauer Associates, Inc.

[Bryson, Jr. and Ho, 1969] Bryson, Jr., A. E. and Ho, Y. (1969). *Applied optimal control: optimization, estimation and control*. Blaisdell, Buffalo, New York.

[Burshtein, 1998] Burshtein, D. (1998). Long term attraction in higher order neural networks. *IEEE Transactions on Neural Networks*, 9(1):42–50.

[Cardy, 1996] Cardy, J. (1996). *Scaling and Renormalization in Statistical Physics*. UK: Cambridge University Press.

[Culloch and Pitts, 1943] Culloch, W. S. M. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133.

[Cybenko, 1988] Cybenko, G. (1988). Continuous valued networks with two hidden layers are sufficient. Technical Report.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoid fucntion. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

[DeGloria et al., 1993] DeGloria, A., Faraboschi, P., and Olivieri, M. (1993). Efficient implementation of the Boltzmann Machine algorithm. *IEEE Transactions on Neural Networks*, 4(1):159–163.

[Dembo et al., 1991] Dembo, A., Farotimi, O., and Kailath, T. (1991). High-order absolutely stable neural networks. *IEEE Transactions on Circuits and Systems*, 38(1):57–65.

[Doković, 2008] Doković, D. Ž. (2008). Hadamard matrices of order 764 exist. *Combinatorica*, 28(4):487–489.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience Publication, John Wiley & Sons, INC., 2nd edition.

[Ercegovac et al., 1998] Ercegovac, M. D., Lang, T., and Moreno, J. H. (1998). *Introduction to Digital Systems*. Wiley.

[Evans and Robertson, 1968] Evans, C. R. and Robertson, A. D. J. (1968). *Cybernetics: Key Papers*. University Park Press.

[Farguell et al., 2006] Farguell, E., Mazzanti, F., and Gómez-Ramírez, E. (2006). Increasing Boltzmann Machine learning speed and accuracy with the High Order Decimation method. In *International Conference on Neural Networks and Associative Memories (NNAM)*, México, D.F. Instituto Politécnico Nacional.

[Farguell et al., 2007] Farguell, E., Mazzanti, F., and Gómez-Ramírez, E. (2007). *Boltzmann Machines learning using High Order Decimation*, volume 208, pages 21–42. Springer Berlin / Heidelberg.

[Farguell et al., 2008] Farguell, E., Mazzanti, F., and Gómez-Ramírez, E. (2008). Boltzmann Machines reduction by High Order Decimation. *IEEE Transactions on Neural Networks*, 19(10):1816–1821.

[Fenwick et al., 1977] Fenwick, D. M., Steele, R., and Vasanji, N. (1977). Error detection and correction of dpcm signals using a walsh hadamard transform technique. In *Conference on Digital Processing of Signals in Communications*, pages 225–268.

[Freeman and Skapura, 1993] Freeman, J. A. and Skapura, D. M. (1993). *Redes neuronales: Algoritmos, Aplicaciones y Técnicas de Programación*. Addison-Wesley.

[Gentry et al., 2006] Gentry, S. M., Wehlburg, C. M., Wehlburg, J. C., Smith, M. W., and Smith, J. L. (2006). US patent 6996292 - Staring 2-D Hadamard transform spectral imager.

[Giles and Maxwell, 1987] Giles, C. L. and Maxwell, T. (1987). Learning, invariance, and generalization in high order neural networks. *Applied Optics*, 26(23):4972–4978.

[Graña et al., 1997] Graña, M., d'Anjou, A., Albizuri, F. X., Hernández, M., Torrealdea, F. J., de la Hera, A., and González, A. I. (1997). Experiments of fast learning with high order Boltzmann Machines. *Applied Intelligence*, 7(4):287–303.

[Hagiwara, 1992] Hagiwara, M. (1992). Acceleration for both Boltzmann Machine learning and mean field theory learning. In *International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 687–692.

[Hazewinkel and Vinogradov, 1995] Hazewinkel, M. and Vinogradov, I. (1995). *Encyclopaedia of Mathematics*. Dordrecht, Kluwer Academic, London.

[Hebb, 1949] Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

[Hedayat and Wallis, 1978] Hedayat, A. and Wallis, W. D. (1978). Hadamard matrices and their applications. *Annals of statistics*, 6(6):1184–1238.

[Hertz et al., 1991] Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Addison-Wesley Publishing Company, Santa Fe Institute, Santa Fe.

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proc. Natl. Acad. Sci.*, pages 2554–2558.

[Hopfield, 1984] Hopfield, J. J. (1984). Neurons with graded response have collective computational abilities like those of two-state neurons. In *Proc. Natl. Acad. Sci.*, volume 81, pages 3088–3092.

[Horadam, 2006] Horadam, K. J. (2006). *Hadamard Matrices and Their Applications*. Princeton University Press, Princeton.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Itzykson and Drouffe, 1991] Itzykson, C. and Drouffe, J. (1991). *Statistical field theory*. Cambridge University Press.

[Jankowski et al., 1996] Jankowski, S., Lozowski, A., and Zurada, J. M. (1996). Complex-valued multistate neural associative memory. *IEEE Transactions on Neural Networks*, 7(6):1491–1496.

[Kandel et al., 1995] Kandel, E. R., Schwartz, J. H., and Jessell, T. M. (1995). *Essentials of Neural Science and Behavior*. Appleton & Lange.

[Kappen, 1993] Kappen, B. (1993). Using Boltzmann Machines for probability estimation. In Gielen, S. and Kappen, B., editors, *Proceedings of the International Conference on Artificial Neural Networks ICANN '93*, pages 521–526. Springer-Verlag.

[Kappen, 1995] Kappen, H. J. (1995). Deterministic learning rules for Boltzmann Machines. *Neural Networks*, 8:537–548.

[Kappen and Rodriguez, 1998] Kappen, H. J. and Rodriguez, F. B. (1998). Efficient learning in Boltzmann Machines using Linear Response Theory. *Neural Computation*, 10(5):1137–1156.

[Kappen and Wiegerinck, 2001] Kappen, H. J. and Wiegerinck, W. (2001). Second order approximations for probability models. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13, pages 238–244.

[Kharaghani and Tayfeh-Rezaie, 2004] Kharaghani, H. and Tayfeh-Rezaie, B. (2004). A Hadamard matrix of order 428. *Journal of Combinatorial Designs*, 13(6):435–440.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

[Klahr and Siegler, 1978] Klahr, D. and Siegler, R. S. (1978). *The Representation of Children's Knowledge*, pages 61–116. New York: Academic Press.

[Koenig et al., 1992] Koenig, A., Wehn, N., and Glesner, M. (1992). Partitioning on Boltzmann Machines. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 324–327.

[Kosko, 1992] Kosko, B. (1992). *Neural networks and fuzzy systems*. Prentice Hall international editions.

[Kosmatopoulos and Christodoulou, 1994] Kosmatopoulos, E. B. and Christodoulou, M. A. (1994). The Boltzmann g-RHONN: a learning machine for estimating unknown probability distributions. *Neural Networks*, 7(2):271–278.

[Kosmatopoulos and Christodoulou, 1995] Kosmatopoulos, E. B. and Christodoulou, M. A. (1995). High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Crcuits and Systems II: Analog and Digital Signal Processing*, 42(9):592–603.

[Kosmatopoulos et al., 1995] Kosmatopoulos, E. B., Polycarpou, M. M., Christodoulou, M. A., and Ioannou, P. A. (1995). High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, 6(2):422–431.

[Kullback, 1959] Kullback, S. (1959). *Information theory and statistics*. New York: Wiley, 2nd edition.

[Kuroki et al., 1999] Kuroki, T., Tanaka, T., and Taki, M. (1999). Examination of effectiveness of higher-order mean field Boltzmann Machine learning based on linear response theorem. In *International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1442–1445.

[Langevin and Zanotti, 2005] Langevin, P. and Zanotti, J. P. (2005). Nonlinearity of some invariant boolean functions. *Designs, Codes and Cryptography*, 36(2):131–146.

[Leisink and Kappen, 2000] Leisink, M. A. R. and Kappen, H. J. (2000). Learning in higher order Boltzmann Machines using Linear Response. *Neural Networks*, 13(3):329–335.

[Lin and Lee, 1995] Lin, C. T. and Lee, C. S. G. (1995). A multi-valued Boltzmann Machine. *IEEE Transactions on Systems, Man and Cybernetics*, 25(4):660–669.

[Lipmaa, 2002] Lipmaa, H. (2002). *On Differential Properties of Pseudo-Hadamard Transform and Related Mappings*, volume 2551, pages 48–61. Springer Berlin / Heidelberg.

[Matheus and Rendell, 1989] Matheus, C. J. and Rendell, L. A. (1989). Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 645–650, Detroit, MI. Morgan Kaufmann.

[Thrun et al., 1991] Thrun et al., S. (1991). The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA.

[McEliece et al., 1987] McEliece, R., Posner, E., Rodemich, E., and Venkatesh, S. (1987). The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33(4):461–482.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *An introduction to Computational Geometry*. MIT Press.

[Muezzinoglu et al., 2003] Muezzinoglu, M. K., Guzelis, C., and Zurada, J. M. (2003). A new design method for the complex-valued multistate Hopfield associative memory. *IEEE Transactions on Neural Networks*, 14(4):891–899.

[Newman et al., 1998] Newman, D. J., Hettich, S., Blake, C. L., and Merz, C. J. (1998). UCI repository of machine learning databases.

[Nijman and Kappen, 1996] Nijman, M. J. and Kappen, H. J. (1996). Efficient learning in sparsely connected Boltzmann Machines. In *Proceedings of the Artificial Neural Networks - ICANN 96 6th International Conference*, volume 1112 of *Lecture Notes in Computer Science*, pages 41–46.

[Noordewier et al., 1991] Noordewier, M. O., Towell, G. G., and Shavlik, J. W. (1991). Training Knowledge-Based Neural Networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, volume 3. Morgan Kaufmann.

[Nyström and Popovic, 1998] Nyström, J. and Popovic, B. (1998). US patent 6526091 - communication methods and apparatus based on orthogonal hadamard-based sequences having selected correlation properties.

[Orrick, 2008] Orrick, W. P. (2008). Switching operations for Hadamard matrices. *SIAM J. Discrete Math*, 22(1):31–50.

[Oyama, 1993] Oyama, T. (1993). Fault section estimation in power system using Boltzmann Machine. In *Proceedings of the Second International Forum on Applications of Neural Networks to Power Systems (ANNPS)*, pages 3–8.

[Paley, 1933] Paley, R. E. A. C. (1933). On orthogonal matrices. *J. Math. Phys*, 12:311–320.

[Parisi, 1988] Parisi, G. (1988). *Statistical field theory*. Frontiers in Physics, 66. Addison-Wesley.

[Parra and Deco, 1993] Parra, L. and Deco, G. (1993). Continuous Boltzmann Machine with Rotor Neurons. In *Proceedings of 1993 International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1397–1400.

[Pazienza et al., 2007] Pazienza, G. E., Gómez-Ramírez, E., and Vilasís, X. (2007). *Polynomial Cellular Neural Networks for Implementing the Game of Life*, volume 4668, pages 914–923. Springer Berlin / Heidelberg.

[Peretto and Niez, 1986] Peretto, P. and Niez, J. J. (1986). Long term memory storage capacity of multiconnected neural networks. *Biological Cybernetics*, 54(1):53–63.

[Peterson and Anderson, 1987] Peterson, C. and Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, 1(5):995–1019.

[Pichler, 2004] Pichler, F. (2004). Walsh-functions: early ideas on their application in signal processing and communication engineering. In *The 2004 International TICSP Workshop on Spectral Methods and Multirate Signal Processing Proceedings, SMMSP2004*, volume 25.

[Plefka, 1982] Plefka, T. (1982). Convergence condition of the TAP equation for the infinite-ranged ising spin glass model. *J. Phys. A*, 15:1971–1978.

[Prechelt, 1994] Prechelt, L. (1994). PROBEN1 - A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, MA.

[Press et al., 1993] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1993). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge.

[Rager, 1992] Rager, J. E. (1992). Complex Boltzmann networks and one stage learning. In *International Joint Conference on Neural Networks (IJCNN)*, volume 4, pages 791–795.

[Ramón y Cajal, 2008] Ramón y Cajal, S. (2008). *Histología del sistema nervioso del hombre y de los vertebrados*. Consejo Superior de Investigaciones Científicas (CSIC).

[Rosenblatt, 1961] Rosenblatt, F. (1961). *Principles of Neurodynamics. Perceptrons and the theory of brain mechanisms*. Cornell Aeronautical Lab, Inc., Buffalo, New York.

[Rubinstein, 1981] Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo method*. John Wiley & Sons, Inc., New York, NY, USA.

[Rüger, 1997] Rüger, S. M. (1997). Decimatable Boltzmann Machines for diagnosis: Efficient learning and inference.

[Rüger et al., 1996] Rüger, S. M., Weinberger, A., and Wittchen, S. (1996). Decimatable Boltzmann Machines vs. Gibbs Sampling. Technical Report 96-29, Informatik, Technische Universität, Berlin.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, pages 45–76.

[Saul and Jordan, 1994] Saul, L. and Jordan, M. I. (1994). Learning in Boltzmann trees. *Neural Computation*, 6(6):1174–1184.

[Schneider and Card, 1993] Schneider, C. R. and Card, H. C. (1993). Analog CMOS deterministic Boltzmann circuits. *Solid-State Circuits, IEEE Journal of*, 28(8):907–914.

[Sejnowski, 1987] Sejnowski, T. J. (1987). High-order Boltzmann Machines. In *AIP Conference Proceedings 151 on Neural Networks for Computing*, pages 398–403, Snowbird, Utah, E.U.A.

[Shanks, 1969] Shanks, J. L. (1969). Computation of the fast Walsh-Fourier transform. *IEEE Transactions on Computers*, 18(5):457–459.

[Slotine and Weiping, 1991] Slotine, J. J. and Weiping, L. (1991). *Applied Nonlinear Control*. Prentice Hall.

[Stone, 1977] Stone, M. (1977). Asymptotics for and against cross-validation. *Biometrika*, 64:29–35.

[Storkey, 1997] Storkey, A. (1997). Increasing the capacity of a Hopfield network without sacrificing functionality. In *International Conference on Artificial Neural Netwroks (ICANN)*, pages 451–456.

[Sylvester, 1867] Sylvester, J. J. (1867). Thoughts on inverse orthogonal matrices, simultaneous sign successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers. *Philosophical Magazine*, 34:461–475.

[Tallia et al., 1984] Tallia, R., Morello, P., and Castellano, G. (1984). The Walsh-Hadamard transform: An alternative means of obtaining phase and amplitude. *Journal of Nuclear Medicine*, 25(5):608–612.

[Tanaka, 1999] Tanaka, T. (1999). Estimation of third-order correlations within mean field approximation. In Usui, S. and Omori, T., editors, *Proceedings of the Fifth International Conference on Neural Information Processing '98 Kitakyushu*, volume 1, pages 554–557s.

[Thathachar and Arvind, 1999] Thathachar, M. A. L. and Arvind, M. T. (1999). Global Boltzmann perceptron network for online learning of conditional distributions. *IEEE Transactions on Neural Networks*, 10(5):1090–1098.

[The Mathworks^{TM}, 2008a] The Mathworks^{TM} (2008a). Matlab Help. The Mathworks^{TM}, Release 14.

[The Mathworks^{TM}, 2008b] The Mathworks^{TM} (2008b). MATLAB® 7.6 - The Language of Technical Computing. Neural Networks Toolbox^{TM}.

[Walsh, 1923] Walsh, J. L. (1923). A closed set of normal orthogonal functions. *Amer. J. Math*, 45:5–24.

[Xiang et al., 1994] Xiang, Z., Bi, G., and Le-Ngoc, T. (1994). Polynomial perceptrons and their applications to fading channel equalization and co-channel interference suppression. *IEEE Transactions on Signal Processing*, 42(9):2470–2480.

[Younes, 1994] Younes, L. (1994). Learning algorithms for extended models of Boltzmann Machines. In *Proceedings of the 12th IAPR International conference on pattern recog-*

nition, 1994. Vol. 2 - Conference B: Computer Vision & Image Processing, volume 2, pages 602–604.