Investigating the Consistency and Convexity of Restricted Boltzmann Machine Learning

Author:
Bjørn Erik Juel

Supervisor:
Yasser Roudi

Norges Teknisk-Naturvitenskapelige Universitet (NTNU)

The Kavli Institute for Systems Neuroscience and Centre for Neural Computation

Summary

In this thesis we asses the consistency and convexity of the parameter inference in Boltzmann machine learning algorithms based on gradient ascent on the likelihood surface. We do this by first developing standard tools for generating equillibrium data drawn from a Boltzmann distribution, as well as analytically exact algorithms for inferring the parameters of restricted and semi-restricted Boltzmann machine architctures.

After testing, and showing, the functionality of our algorithms, we assess how different network properties effect the inferrence quality of restricted Boltzmann machines. Subsequently, we look closer at the likelihood function itself, in an attempt to uncover more rigid details about its curvature, and the nature of its convexity.

As we present results of our investigation, we discuss the findings, before suggesting possible future directions to take, improvements to make and aspects to further investigate.

We conclude that the standard, analytically exact restricted Boltzmann machine algorithm is convex up to certain permutations of the parameters, when initialized within reasonable ranges of parameter values, and given that the strength of connectivity in the underlying model is within a specified range. Additionaly, for strengths of connectivity, the distribution of Hessian eigenvalues of the likelihood function, as a funtion of the distance to a peak, may be stable both within and across network sizes.

Contents

1 Introduction **2**
- 1.1 Ising Models . . . 3
  - 1.1.1 The Inverse Problem . . . 5
- 1.2 Boltzmann Machines . . . 6
  - 1.2.1 Mathematical Model of the Boltzmann Machine . . . 9
  - 1.2.2 Permutations of the Hidden Nodes . . . 12
- 1.3 Formulating the Project . . . 12

2 Simulation Algorithms **16**
- 2.1 Metropolis Algorithm for Data Generation . . . 16
- 2.2 Boltzmann Machine Learning . . . 18
- 2.3 Defining the Standard Network Parameters . . . 19
- 2.4 Simulation Algorithm Functionality . . . 20
- 2.5 Performance of Data Generation . . . 20
- 2.6 Boltzmann Machine Inference . . . 21

3 Restricted Boltzmann Machine Learning **28**
- 3.1 Inference Sensitivity in RBM Learning . . . 28
  - 3.1.1 Sensitivity to the Data Set Size . . . 28
  - 3.1.2 Sensitivity to Initial Conditions . . . 30
  - 3.1.3 Sensitivity to the Strength of Connectivity . . . 32
  - 3.1.4 Sensitivity to the Number of Parameters . . . 35
- 3.2 Curvature of the Likelihood Surface . . . 38
  - 3.2.1 Qualitative Investigation . . . 38
  - 3.2.2 The Hessian Matrix . . . 41
- 3.3 Permutations of Hidden Nodes and Local Convexity . . . 48
  - 3.3.1 Learning with Clamped Parameters . . . 48

4 Concluding Remarks **52**
- 4.1 Possible Future Directions . . . 52
- 4.2 Conclusion . . . 54

A **60**
- A.1 Derivations . . . 60
  - A.1.1 Maximum Entropy Distribution . . . 60
  - A.1.2 Learning Rules from Log-Likelihood . . . 63
  - A.1.3 The Hessian Matrix . . . 69

A.1.4 Correlations Without Connections 73

B Sample Code **79**

C Additional Plots and figures **85**

List of Figures

1.1 Illustration of neural action 2
1.2 General architecture of an Ising model 4
1.3 Figure of the general structure of Boltzmann machines 7
1.4 Illustration of invariance in permutation of hidden nodes 11

2.1 Metropolis algorithm pseudo-code 17
2.2 Pseudocode for the learning algorithm used 18
2.3 Visualization of the precision of data generation as a function of data points using the Metropolis algorithm 21
2.4 The learning evolution of a 5x1 Boltzmann machine with restricted architecture 22
2.5 The likelihood surface of the simplest RBM, with two observed and one hidden node. 24
2.6 Learning evolution of Monte Carlo Approximation 26

3.1 Illustrating the RBM learning precision as a function of data sets . 29
3.2 Final RMS error as a function of noise in the initial conditions. . . 30
3.3 Root mean square error as a function of the connection strength scaling factor, g 33
3.4 The Likelihood functions variation as a function of distance from the inferred parameters. 34
3.5 The effect of learning precision when changing the number of nodes in the model 37
3.6 The likelihood surface of the simplest RBM, with two observed and one hidden node. 38
3.7 Surface values of the likelihood function in different areas of the parameter space 40
3.8 Histograms of all, and the maximum, eigenvalues of the Hessian matrix of the likelihood surface 42
3.9 Descriptive statistics for the eigenvalues of the Hessian matrix around the inferred point in parameter space 44
3.10 Maximum Hessian eigenvalues as a function of connection strength 45
3.11 Visualization of the positive eigenvalues of the Hessian matrix of the log Likelihood for different network sizes and data lengths . . . 47
3.12 Visualization of the convexity, up to permutations of the parameters, in the restricted Boltzmann machine learning 49

A.1 Visualization of the many equivalent solutions of the semi-restricted Boltzmann machine 77

C.1 Illustrating the RBM learning precision as a function of data sets . 85
C.2 Final RMS error as a function of noise in the initial conditions and a comparison with initial conditions independent of the ground truth parameters 85
C.3 Sensitivity of learning algorithm to initial conditions for different size RBM's 86
C.4 The the behavior of likelihood values as a function of the distance in parameter space from the inferred values 87
C.5 Figure for predicting the number of hidden variables in the model . 87
C.6 Surface values of the likelihood function in different areas of the parameter space 88
C.7 Histograms of Hessian eigenvalues in single realizations of 8-by-3 restricted Boltzmann machines 89
C.8 Descriptive statistics of the Hessian eigenvalues as a function of the distance from the inferred peak parameters. 90
C.9 Histograms of all eigenvalues used for the descriptive statistics . . 91
C.10 Visualization of the convexity, up to permutations, of the parameters in the restricted Boltzmann machine architecture for a 6-by-1 network 92

Chapter 1

Introduction

The human cortex has been estimated to contain approximately 100 billion neurons, each of which is connected to tens of thousands of other neurons on average. There are countless types of neurons with distinct structure, function or both, but nearly all of them have certain important properties in common: they receive input stimuli from external sources, most commonly through dendritic synapses from other neurons, which lead to a slight depolarization of the membrane potential. These voltage changes are propagated along the dendrites to the soma, where its response is computed based on an integration of the total input it received. The cell's range of reaction is generally limited to a binary, all-or-nothing response in the form of only generating, and propagating, an action potential along its axon if the summed membrane depolarization is above threshold. Such an action potential eventually reaches the axon terminals, where the electrical impulse generally leads to the release of chemical signal molecules, called neurotransmitters. They diffuse across the synaptic cleft to the post-synaptic cell's receptors, eliciting some intracellular response. In cases where the post-synaptic cell is a neuron as well, the intracellular response is usually a change in the membrane potential caused by opening of ion channels, restarting the cycle described above [1, 2].

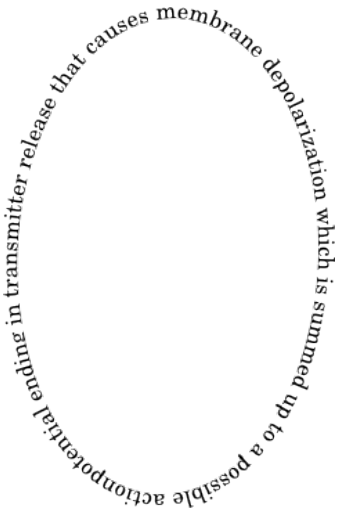


Figure 1.1: *A simple illustration of the standard properties of neural firing*

This general chain of neural actions - depolarization, summation and firing - has led to a common practice in neural modeling of disregarding the cell-type specific functional differences. Taken to the extreme, one ends up with a neuron being modeled as a binary point variable with some numerical functional connection with other cells. This sort of approximation dates all the way back to the beginning of the twentieth century, when Lapique introduced the integrate-and-fire

neuron [3], and was expanded upon in the 40's with McCullogh and Pitt's point unit neuron [4].

Obviously, if the goal is to study single cell responses or action potential generation, it makes little sense to ignore cell-specific details, and for those purposes one could apply a range of other mathematical models developed. For example, small variations of the Hodgkin and Huxley conductance based model of the membrane for action potential generation [5] and Rall's mathematical description of the electrical properties dendritic arborizations [6, 7] are are still popular for detailed description of neuron properties although they are old [8]. On the other hand, in cases where the interest lays in studying the properties and functions of populations, or networks of neurons, such drastic simplifications of the single neurons turn out to have little effect on the analysis, and works similarly to averaging out small scale, noise from larger scale network processes and essential dynamics [9, 10].

When we allow for the most drastic approximations for the data-generating network - an assembly of binary point neurons with instantaneous information transfer through numerical, functional synapses - we can take advantage of the vast methods of statistical mechanics to tell us about the network. The field was originally developed to understand the properties of large ensembles of identical particles from their individual properties and interactions, which fits the description of a population of our approximate neurons well. Such applications of the methods from statistical physics is a rapidly expanding area of the field, and is of major importance for understanding the complex networks of systems biology, such as those relevant for neuroscience [11]. As an example, Roudi *et al.* have used methods from statistical physics to reverse engineer the biological functional structure of networks of neurons, just from data correlated time-series of several single neuron spike trains [12].

As we learn more about the apparent anatomy, and functions of the brain, from the network and systems level, down through single neurons and into the realm of chemistry, computational neuroscientists are able to make more biologically realistic models to capture the complexity in the structure and architecture of arguably the most important organ in our body. Coupling simulation results and predictions with analytical models and experimental results, scientists may go several rounds back and forth between simulations, analytics and experiments to improve their model's consistency with nature [8]. However, copying or imitating biological systems is not always the end goal of computational neuroscience.

1.1 Ising Models

When viewed at the most abstract level, some complex systems can be evaluated using simple models originally developed for different problems in different fields to extract properties that remain obscure, or hard to analyse, when working with biologically relevant model systems. One such simple model, originally developed

to work with magnetization in structured materials and phase transitions in quantum gases, is known as the Ising-model [13]. It is a computational model which can be used to calculate the statistics of the stable states in macroscopic systems of identical binary units interconnected in some user defined arrangement. Even though the Ising model was originally developed for spins with nearest-neighbor connectivity in a crystalline lattice architecture, the term will be used more liberally in this thesis. Here, an Ising model of a network is simply a collection of binary nodes with some defined scheme of interconnectivity, and a related energy function defining its states of activity.

When the activity of the i'th node is denoted as s_i, and a state of the network, $\mathbf{S}$, is defined as the vector of all the network unit's activities, $\mathbf{S} = [s_1, s_2, \ldots s_i, \ldots s_N]$, we can define its energy function, $\mathscr{E} = \sum_{i,j} J_{i,j} s_i s_j$, where $J_{i,j}$ is a number quantifying the strength of connection between unit i and j. Usually, $\mathscr{E}$ will also contain a collection of terms relating some external field's influence on each node of the network, but we have omitted that here for simplicity. The energy, $\mathscr{E}$, is directly related to the probability distribution of states through the Boltzmann distribution, $P(\mathbf{S}) = \exp(-\mathscr{E}(\mathbf{S}))\mathscr{Z}^{-1}$, where $\mathscr{Z} = \sum_{states} \exp(-\mathscr{E}(\mathbf{S}))$ is known as the partition function from statistical mechanics, and is a sum over all states of the system. This means that if we know the strength of all the connections in the network, we can in theory calculate the probability for finding the system in any given state, and subsequently any statistic of the system.

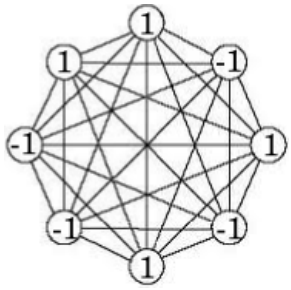


Figure 1.2: *The general structure of an Ising model, where circles denote the binary nodes, or spins, and the lines connecting them show which nodes are connected.*

Unfortunately, the number of possible states increases exponentially with the number of nodes, N, quickly making it impossible to compute the partition function exactly, since it demands summing over the entire state space. This can be partly remedied using approximations like mean field methods[14, 15, 16], or by probing the network's state space using a clever algorithm developed by Metropolis *et al.* in the 50's [17]. These approximations would give estimates of the network statistics as well as the partition function itself, a function that basically contains all information about the model, and can be used to calculate many properties of the macroscopic system from relations derived in statistical mechanics.

In neuroscience, the Ising model architecture was used by Hopfield to simulate a network of neurons recovering memories [18]. In his model, the spins are interpreted to be neurons, with the activity being 1 when it fires an action potential and -1 otherwise. Additionally, giving the parameters for connectivity a loose interpretation as functional synapses, gave the network an intuitive resemblance to the biological neural networks. Starting from any example input state of activity, a stochastic, asynchronous scheme of updating neurons to a more cost efficient

(lower energy) state would lead the system as a whole to settle in a local minimum of the energy-function, which could be interpreted as the network having retrieved a memory most resembling the input [18]. In other words, given a set of parameters for the unit interactions, it was possible to find the stable state of a network in the vicinity of any initial state.

Elsewhere, similar large scale systems of interacting variables, being studied using a network regime, can be found in several fields of research. For example, in molecular biology, genetic patterns of expression are studied as networks controlling protein synthesis [19], which in turn interact in networks of pathways to control cellular function [20]. On a different level, but with a similar structure, dynamical systems in the financial world [21], as well as in the study of our climate system [22], use complex network models with large numbers of variables to predict future developments of the system. In other words, understanding composition and function of dynamic networks is important for the understanding of complex systems in general, and biology and neuroscience specifically.

1.1.1 The Inverse Problem

The Ising model is interesting for studying physical systems, but a inverse implementation can also be quite appealing. It allows us to gain knowledge about the underlying network's structure and dynamics based on the observed information provided by experimentalists. Assuming that the observations are samples from an equilibrium distribution of an Ising model, one can iteratively reconstruct a statistically plausible structure of the effective interactions in the network as a whole [23]. This type of reverse engineering of networks can be used to functional connections of any system, satisfying some limitations on the structure. Already and it has been used to model both protein-protein [24] and genetic interactions [25] from high throughput method data in molecular biology, as well as functional connections from neural activity [26].

One way to reverse engineer a statistically relevant model of the network, without being overwhelmed by the vast dimensionality of the problem, is to use a parametric model which approximates the statistics of the real distribution over the state space well, even with a relatively small number of variables [27]. Now, there are an infinite number of potential models consistent with the measured momenta of the data, though most will over-fit them and not generalize to new samples. This is a problem which is generally caused by including too many parameters in the model. A good way to avoid such over-fitting is putting hard constraints on the number, or type, of parameters, or introducing some cost for having more of them. In statistical physics it is popular to choose the model which maximizes entropy, under the constraints of fitting only low order moments, such as means and pairwise correlations [27, 28]. An important motivation for using such models is that they are the most informative, yet least structured, model consistent with the lower order moments, without assuming the existence of higher order interactions. It is known as the maximum entropy distribution [29, 30].

Looking for the maximum entropy model can seem reasonable when considering the second law of thermodynamics, which says that systems in equilibrium tend towards the distribution of maximum entropy [31]. On the other hand, there is no *a priori* reason why we should only include second order correlations in our model, except for the sake of simplicity. As mentioned above, it is, in theory, possible to construct a maximum entropy model consistent with any K'th order moment, where $1 < K < N$ and N is the number of nodes in the network. Demanding the model be consistent with all N orders of interaction leads to a model able to exactly replicate the data statistics, while choosing an order $K = 1$ would make the assumption that every node is independent, and the model would only regenerate the unit means seen in the data [28].

In 2003, Schneidmann *et al.* [29] studied the difference in entropy in, or information carried by, systems when including different orders of correlations in their description of the underlying model for real spiking data. They used the concept of mutual information, $I_K = S_{K-1} - S_K$ in their analysis, to quantify the K'th order correlation's contribution to the total information in the model, calculated as the difference between the difference in entropy S between the K'th and the $(K-1)$'th order model. The total amount of correlation in the network is measured by $I_N = \sum_K I_K = S_1 - S_N$, and can be used to explore whether the correlation terms we include are sufficient or not for a good description of the network model. For example, the ratio $I_{(2)}/I_N$ quantifies the proportion of total information gained from pairwise correlation terms [29].

When applying this method to two sets of real retinal spike data, Schneidmann *et al.* [28] found that 90% of the correlational information could be extracted from the second and first order terms. Their findings implied that pairwise models give a good picture of the true distribution of neuronal couplings and that "the network is much more than the sum of its parts, but a nearly complete model can be derived from all its pairs" [28]. This dominance in explanatory power was shown by Roudi *et al.* [12] to be an effect of considering only few neurons, all having low firing rates, and it is not generalizable.

Most inference methods in use for systems biology at present only consider a network of the observed units. This can be a serious problem, as a large part of the systems being studied is not directly observed in the data, such as unrecorded neurons or gene products, but still interact with the network of interest [32]. As an example, most of the spike data from cortical networks are assumed to stem from excitatory cells, even though it is known that the effects of inhibitory cells is an essential factor for the cortical network dynamics [33].

1.2 Boltzmann Machines

The reverse engineering of underlying model parameters is an important part of the field of machine learning. It is a part of the broader discipline known as artificial

intelligence, and is basically a set of statistical methods in which computer programs learn to do some task optimally, without being explicitly instructed in how to do it. Some of the problems they tackle can be set up using a framework known as graphical models - basically networks of nodes with statistical interdependencies conveyed by connections between them. These have been shown to be useful when working on the types of networks interesting for researchers in systems biology [34].

Methods and algorithms from machine learning can be used for neuroscientific purposes in different ways. For example, they can be used to assist with data mining and statistical analysis, such as connecting spikes with neurons in electrode data, or more directly as a source for theories on how the brain is structured, or functions [34]. Here we will focus on one family of graphical models, and assess the strength of inverse problem of learning the connections.

The graphical model we will be assessing is a popular network model which was developed when Ackley, Hinton and Sejnowski set out to find an adaptive system that would learn the structure of a model, without requiring a lot of problem specific information. This led them to what we now know as Boltzmann machines (BM), a general network model that "is capable of learning the underlying constraints of a domain simply by being shown examples from the domain" [35]. In other words, like in the inverse Ising problem, the Boltzmann Machine is able to find a set of parameters that characterizes the network that generated the data, just based on the data sampled from it [35].

Learning the set of parameters best suited for reconstructing the statistics of the presented test-data, within some given model framework, is a detrimental part of the field of machine learning in general, and a strong property of the Boltzmann machine specifically. Hinton called this application of the Boltzmann Machine the 'learning problem', in contrast to the so called 'search problem', in which the parameters, or connection weights, are set, and the Boltzmann Machine is fed an input vector and used to search the solution space for the optimal output vector [36]. These two applications show a link to the previously discussed Ising models and Hopfield networks. In the search problem, finding the best output vector can be compared with memory retrieval in Hopfield networks [37], while the learning problem is similar to the reverse Ising problem of constructing

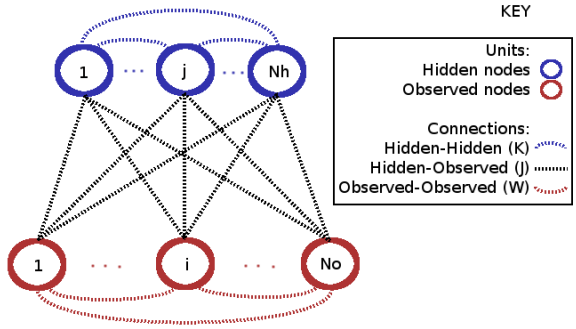


Figure 1.3: *The architecture of a general Boltzmann Machine, containing two layers of nodes (can also be referred to as spins and neurons) and three distinct sets of connections. There are No observed and Nh hidden nodes, of which some general un-numbered node is denoted as the i'th or j'th respectively.*

connections from data statistics.

Before we delve deeper into the properties of Boltzmann machine learning, we should get familiar with a few variations of the graphical model used to describe the network. In figure 1.3 the general architecture of a Boltzmann machine can be seen, a structured group of symmetrically connected nodes. Normally, the nodes (also called spins or neurons) are divided into a set of observed, with which the data can be related, and a distinct set of hidden elements. Following this division, the connections between distinct nodes can also be grouped - there are two sets of within-layer and one set of between-layer connections. The grouping into observed and hidden variables is visualized by color coding in the figure, with red being related to the observed, and blue with the hidden.

The grouping of different types of connections draws up natural guidelines for specifying different types of Boltzmann machine architectures. The general Boltzmann machine displayed in 1.3 is fully connected, with every node directly interacting every other node. This gives a vast number of parameters, and a network in which the activity of all nodes is conditionally dependent on all others. With such a high number of parameters, and heavy interconnectivity, also between the unobserved nodes in the network, we will have a very hard time inferring the correct model parameters generating the data we observe. This is because there is likely to be several different solutions for parameters that match the pairwise correlation constraints. Additionally, this network gets tedious to work with because we need to solve sets of self-consistent equations repeatedly to even be able to use our learning algorithms.

The simpler, semi-restricted Boltzmann machine (sRBM) removes the connections between the hidden nodes. This makes them independent, and even though they are still indirectly connected through the observed layer, we can separate the terms containing each hidden node in the energy function, $\mathscr{E}$. This still seems to be under-constrained, as there are many parameters to fit only to match pairwise correlations, but we will look further into this in the next chapter.

The simplest system, and the one we will be studying the closest is the completely restricted Boltzmann machine (RBM). Here, we set all observed-to-observed node connections to zero as well. As long as we keep the number of hidden nodes below, or near, the number of observed nodes, we should not face over-fitting problems when finding the optimized set of parameters. Even though it is an old model, this type of network is still used a lot in machine learning.

This is especially true in a framework known as Deep Belief networks, which were introduced some years ago by Hinton *et al.*, where large RBM's are stacked vertically to function as feature detectors in a pattern recognition framework [38]. Such deep architecture networks can describe complex, non-linear concepts using much fewer nodes than shallow architectures with only one or two layers. However, it was, generally considered impractical to use, until Bengio *et al.* developed an efficient algorithm for training in which one could approach the parameter inference

in a layer-wise fashion [39]. This made the deep architecture networks much more approachable, and when the RBM's are trained using an algorithm based on contrastive divergence developed in the beginning of last decade [40], even networks with large numbers of nodes in each layer could be learned in a reasonable fashion.

The contrastive divergence method is basically a truncated Monte Carlo method from which a low variance estimate of the model distribution is obtained [40]. Other approximations for large systems have been developed as well, for example using mean field approaches, both naive and expanded with self-reaction terms, to handle the popular large Boltzmann machines [41, 14, 42]. But even though these great approximations do very well for large systems, and has been studied in detail, it is not easy to find exhaustive studies of Boltzmann machines at the most basic levels.

1.2.1 Mathematical Model of the Boltzmann Machine

The BM family got its name because the probability density over the space of its states of activity is described by the Boltzmann distribution. This can be seen quickly when we work with a symmetric coupling matrix, which we will. It ensures the dynamics of the network "satisfies detailed balance" - a property which can be shown to lead directly to an equilibrium distribution of the correct form [32]. We encountered it in the previous section, and we saw that the probability for observing some given state, $\mathbf{S}$, of the Ising model was related to that states energy, $\mathscr{E}(\mathbf{S})$. Since the Boltzmann machine contains two distinct sets of nodes, one with No visible nodes and one with Nh hidden, it is useful to separate them in the state vector. We use the notation, v_i and h_j for the i'th visible and j'th hidden node, respectively, and we write the full network state as $\mathbf{S} = [v_1, \ldots, v_{No}, h_1, \ldots, h_{Nh}]$.

In the the appendix A.1.1 we show the derivation of the maximum entropy distribution for the sRBM using Lagrange multipliers constraining the model to match data means and correlations. The derivation generalizes easily to be used on the fully connected Boltzmann machine as well, giving

$$-\mathscr{E} = \sum_{ij} J_{ij} v_i h_j + \sum_{ik} W_{ik} v_i v_k + \sum_{jl} K_{jl} h_j h_l + \sum_i f_i v_i$$

$$\implies P(\mathbf{v}, \mathbf{h}) = \exp\big(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_{ik} W_{ik} v_i^s v_k^s + \sum_{jl} K_{jl} h_j^s h_l^s + \sum_i f_i v_i^s\big)\mathscr{Z}^{-1}$$

$$where \quad \mathscr{Z} = \sum_s \exp\big(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_{ik} W_{ik} v_i^s v_k^s + \sum_{jl} K_{jl} h_j^s h_l^s + \sum_i f_i v_i^s\big)$$

Since we do not know the values of any of the hidden variables, h_j, we would like to find a distribution over the observed states, $\mathbf{v}$, by marginalizing out the h's. This is not straight forward for the fully connected case, but for the more restricted variations, the hidden variables are independent, making them separable in the exponent. The system we will end up studying closest in the following chapters is the RBM with a zero external field, $\mathbf{f}$, and for it the PDF over the observed states

turns out to be

$$P(\mathbf{v}) = \sum_{\mathbf{h}} P(,\mathbf{h}) = \sum_{\mathbf{h}} \frac{\exp(\mathscr{E}(\mathbf{v},\mathbf{h}))}{\mathscr{Z}} = \frac{\sum_{\mathbf{h}} \exp(\sum_{i,j} J_{ij} v_i h_j)}{\sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(\sum_{i,j} J_{ij} v_i h_j)}$$
$$= \frac{\prod_j \left(\exp(\sum_i J_{ij} v_i) + \exp(-\sum_i J_{ij} v_i) \right)}{\sum_{\mathbf{v}} \prod_j \left(\exp(\sum_i J_{ij} v_i) + \exp(-\sum_i J_{ij} v_i) \right)}$$

$$\implies P(\mathbf{v}) = \frac{\prod_j \cosh\left(\sum_i J_{ij} v_i \right)}{\sum_{\mathbf{v}} \prod_j \cosh\left(\sum_i J_{ij} v_i \right)} \tag{1.1}$$

This is the distribution we are actually seeing data from, but realizing that it does contain all the same parameters, it could be sufficient to describe the full distribution. Having access to this, quite compact, formula may also come in handy when discussing the properties and shape of the density function.

Inferring the Connections in Boltzmann Machines

Looking at the appendix A.1.1 where the derivation maximum entropy model for the Boltzmann machine is found, there is an important thing to notice; we were only able to derive the general structure of the formula, and there are still parameters that need fitting. For the RBM, we also constrain the formula to be consistent with with the data means and correlations, but at the time of the derivation we did not have any data to use. However, when we have received a data set of the observed activity, we are ready to find the rest of the model parameters. That is, given some set of data we can try to reconstruct the parameters of the underlying, generative model by adjusting the parameters of some test model, such that it can recreate statistics comparable to those found in the dataset [27].

One way of updating to find these parameters is by trying to maximize a measure quantifying how likely a model is to generate the data observed, given some suggested set of parameters. The likelihood function, $L = \prod_d P(\mathbf{S}_d)$, is such a measure, and maximizing the logarithm, $\mathscr{L} = \log(L)$, of it leads to a set of iterative learning rules first introduced by Ackley *et al.* in 1985 [35]. For the Boltzmann machine the rules take the following form

$$\Delta W_{ik} = \eta_W \big(\langle v_i v_k \rangle_{data} - \langle v_i v_k \rangle_{model} \big)$$
$$\Delta J_{ij} = \eta_J \big(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \big)$$
$$\Delta K_{jl} = \eta_K \big(\langle h_j h_l \rangle_{data} - \langle h_j h_l \rangle_{model} \big)$$
$$\Delta f_i = \eta_f \big(\langle v_i \rangle_{data} - \langle v_i \rangle_{model} \big)$$

Here, the angled brackets denote calculating the expected values of variables inside them, with respect to the distribution indicated in the suffix. The first expected value is calculated directly from the data set, while the right is calculated under the model distribution, using the current guess as parameters. The η is a learning

rate, which can also be optimized; it should be small enough for the momentary gradient to be a decent approximation of the slope in the entire step, while being big enough to allow the learning to be reasonably fast.

Notice that the update rule for J_{ij} seems to require knowledge of the hidden node activity in the data, to calculate the correlations. In the appendix A.1.2, we show the derivations explicitly, and also discuss the apparent problem of needing the knowledge of the h_j's. It turns out that an equivalent expression for the update rule is

$$\Delta J_{ij} = \eta_J \big(\langle v_i \tanh(\sum_i J_{ij} v_i) \rangle_{data} - \langle v_i \tanh(\sum_i J_{ij} v_i) \rangle_{model} \big) \tag{1.2}$$

Interestingly, the expression we use in place of h_j is equivalent to the expected value of h_j given a observed state of activity, $E(h_j|\mathbf{v^s}) = \tanh(\sum_i J_{ij} v_i^s)$, which gives an intuitive explanation for the substitution.

We can use the above learning rules in an iterative manner to progressively learn parameters that fit our data better. Since using them moves our guessed parameters in the direction of most increasing likelihood, and the step size is proportional to the same gradient, the parameters will keep changing until they are in, or arbitrarily near, a stationary point of the likelihood surface, with a higher value than the initial guess. If this stationary point is a maximum of the surface, it yields, by definition of the likelihood, the local parameters most likely to generate the data we are analyzing.

These same rules have been derived in other ways as well, for example by taking a different objective function such as the Kullback-Liebler divergence, a measure quantifying the difference between two distribution, as a starting points [43]. Minimizing the difference between the full data distribution, and the distribution depending on our guess of weights, leads to the exact same learning rules as the ones presented above [35]. This tells us that the parameters we learn are not only the ones most likely to generate the data observed, but also the parameters which make the best pairwise approximation of any underlying distribution used for generating the data.

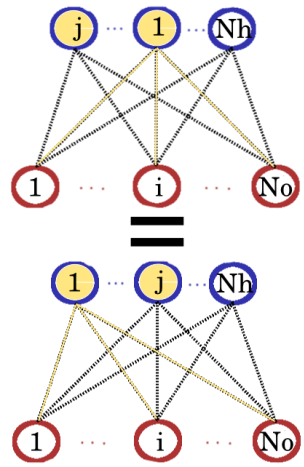


Figure 1.4: *A simple illustration of the insensitivity to permutations of hidden nodes in the RBM architecture. The observed nodes do not notice the difference between the two permutations of hidden nodes, as long as the connections permute along with them (visualized by the yellow lines).*

1.2.2 Permutations of the Hidden Nodes

An interesting property of the distribution over the observed states in 1.1, is that the simplest RBM is invariant to permutations of hidden nodes. The concept is sketched in figure 1.4, and it attempts to show why switching the position of two hidden nodes will not effect the probability of the observed state, $\mathbf{v}$. As long as the connections, $\mathbf{J}$, are switched along with the node, every argument in the hyperbolic cosines of 1.1 remains the same, and the probabilities are unchanged. Another way to see that the PDF should be invariant to permutations is to view it as an issue with labeling the unobserved nodes; the observed states do not care which of the hidden nodes is the first or last, as long as every visible node sees the same labels.

When it comes to the learning algorithm, we seem to be let of the hook when using the rules of the form presented in 1.2. Again, the argument of the hyperbolic function is just a sum over all the visible nodes; the order of the terms in the hyperbolic tangent makes no matter, as long as each argument gets one, and only one, contribution from every observed node. This could lead to different results when inferring model parameters, but since the value of $\mathscr{L}$ is nothing but a product of the data point probabilities, the likelihood values for the different solutions should be equivalent.

Another symmetry present in the network is in the sign of the parameters. This can most easily be understood by the symmetry of the hyperbolic cosine function; switching sign on all connections $J_{i,X}$ to the X'th hidden node, has no effect on the value of the X'th term in the numerator of 1.1. The fact that we can neither label nor know the true sign of the hidden nodes should not worry us to much, however, as they are indeed hidden, and therefore cannot be told apart for real data sets in any case. When it comes to testing our algorithms, we need to keep the labeling problem in mind, though.

1.3 Formulating the Project

In the previous sections, we mentioned why choosing a maximum entropy model of our data would be a good idea when trying to reconstruct the underlying parameters of a network in equilibrium. We also presented some reasons only to include lower order statistics in the formulation of the model when studying neural spike data. However, in this thesis we will not be studying real, experimental spike data, but rather the Boltzmann machine learning algorithm itself, and its precision and reliability, to assess whether it is a good model to use for statistical parameter inference when faced with data that is believed to have been drawn from an equilibrium maximum entropy pairwise distribution.

To do this, we write the necessary simulation algorithms for generating appropriate data, and also for inferring model parameters. The reasoning for not using biological data in this project is related to the need to know the true underly-

ing parameters of the data generating model if we are going to have any way of measuring the precision of the inference algorithm. Therefore we generate data synthetically, from known parameters using the Boltzmann distribution with pairwise interactions. This also ensures that the model we sample from formula is indeed the maximum entropy distribution, and so some pitfalls of making a bad assumption of the underlying model framework are avoided.

In what follows, we present our algorithms and assess their precision for a range of network sizes and varying parameter values. We asses the stability and consistency we can expect from the parameter inference algorithms in Boltzmann machines with hidden nodes, and investigate its sensitivity to varying numbers and distributions of the generative model parameters. We also research the surface of log likelihood function, $\mathscr{L}$, using both qualitative and quantitative methods, to see what can be said about its convexity. Finally, even though it seems clear from the mathematical form of probability distributions that the surface of the likelihood is not globally convex, ultimately we would like to produce an answer to whether the learning problem of the RBM is locally convex and whether the individual peak parameters are all relevant to the true parameters of the model.

Chapter 2

Simulation Algorithms

To research the precision of Boltzmann machine learning, and the convexity of the likelihood function used in it, we need functioning computer algorithms. We derive and write two commonly used general algorithms to asses the problems of interest: one to generate an equilibrium sample from the Boltzmann distribution, and one to infer the most likely parameters of the networks used to generate those data points. In the following, the general structure of those two algorithms will be presented, followed by a definition of some standards we use for the networks and a presentation of simulation results showing their functionality.

2.1 Metropolis Algorithm for Data Generation

First of all, every parameter learning simulation in this project used synthetically generated data. This is useful since a main goal of our research has been to explore when the parameters of the model generating our data can be reconstructed using a restricted Boltzmann machine (RBM). In other words, when we know the actual ground truth parameters of our model, it is easy to compare our algorithm's suggested parameters, and make a measure of its predictive ability.

Given the connectivity of the RBM, and the definition that each spin can only take on one of the values ± 1, the Metropolis algorithm lends itself perfectly to generate a set of data sampled from its equilibrium distribution of states [17]. It is widely used to sample probability distributions due to its simple nature, logical structure and independence of the, often hard to calculate, partition function.

Starting from some random state in a user defined system, the algorithm iteratively attempts to move to an adjacent state by perturbing the current state of the system. By comparing the probabilities of the previous and the perturbed states, the Metropolis algorithm decides whether to accept or reject the change. Since this comparison depends only on the ratio between the new and old state probabilities, and that they are both drawn of the same equilibrium distribution,

Metropolis Algorithm for the Boltzmann Distribution

Guess some initial state of the system, $\mathbf{s_1}$
calculate the energy, $\mathscr{E}(\mathbf{s_1})$, of the state

For a given length of data:
    Flip the activity in a randomly chosen spin
    calculate the energy, $\mathscr{E}(\mathbf{s_2})$, of the new state, $\mathbf{s_2}$

    If $\log\left(\frac{P(\mathbf{s_2})}{P(\mathbf{s_1})}\right) = \log\left(\frac{\exp(\mathscr{E}(\mathbf{s_2}))}{\exp(\mathscr{E}(\mathbf{s_1}))}\right) = \mathscr{E}(\mathbf{s_2}) - \mathscr{E}(\mathbf{s_1}) > \log(randomnumber)$:
        Accept the change of state and set $\mathscr{E}(\mathbf{s_1}) = \mathscr{E}(\mathbf{s_2})$
    Else :
        Reject the change, reset the activity to the previous
        state and keep $\mathscr{E}(\mathbf{s_1})$ unchanged

    Record and save the state activity in a data matrix

Figure 2.1: *Pseudo-code for the Metropolis algorithm when the probability density is given by the Boltzmann distribution,* $P(\mathbf{s}) = \exp(\mathscr{E}(\mathbf{s}))$.

the partition function $\mathscr{Z}$ cancels out. A short pseudo-code for how the Metropolis algorithm works can be found in fig 2.1.

Looking closer at the test for accepting a suggested change of state, we see that whenever the new energy, $\mathscr{E}(\mathbf{s_2})$, is smaller than the old $\mathscr{E}(\mathbf{s_1})$, the change is accepted. However, there is a non-zero chance to move to lower probability states whenever the difference is larger than the randomly generated parameter as well. This allows for the algorithm to explore the entire state space while retaining the model's probability mass differences in the samples. The term 'energy' is used here mainly because the methods were originally developed for physical systems which prefer low energy states, and is kept here for a lack of a more descriptive term.

In general, any data set size is insufficient to perfectly sample the models probability density function, but the precision of the approximated statistics should increase with the sample size. This increase can be quantified, and be used as a sanity check for whether our simulations are working. In general, the statistical sampling error, ϵ, depends on the number of samples, D, in the following way

$$\epsilon = \sqrt{\frac{\sum_s^S P(s)(1-P(s))/D}{S}} \propto 1/\sqrt{D}$$

So, if the sampling error falls off linearly, with a slope $-1/2$ in a log-log scale, the algorithm can be expected to work.

Learning Algorithm

Guess some initial set of parameters

While parameters keep changing:
    Calculate necessary statistics from data
    Calculate corresponding statistics using the parametrized model
    Apply learning rules to update model parameters

Figure 2.2: *Pseudocode for the Learning algorithm by maximizing the log Likelihood*

2.2 Boltzmann Machine Learning

Given a dataset generated from the algorithms discussed above, a set of learning rules and the number of variables in the model, the algorithm to be presented aims to find the maximum likelihood parameters within the assumed model framework.

In section 1.2 we introduced the Boltzmann machine as a statistical model of inference, and we saw a set of learning rules developed which we can use to reach the most likely set of parameters. Because of the form of the pairwize Boltzmann distribution, the potentially complicated formula for the gradient of the log likelihood turns out to be relatively simple, only requiring the calculation of lower order statistics.

As can be seen by the derivation in A.1.2, these rules are developed to move the parameters in the direction of the gradient of the log likelihood, $\mathscr{L}$, and will therefore seize to progress when a stationary point is found. In other words, we should be able to guess any set of initial parameters, and our algorithm will terminate arbitrarily close a point in parameter space with a greater or equal value of likelihood compared to the initial, and a derivative of zero. Whether those parameters comprises the global maximum likelihood solution, is a more complicated question, and will be explored in the next chapter.

Following the structure outlined in the short pseudo-code in figure 2.2, which shows the basic structure of the algorithm, the learning of parameters starts by making an initial guess. The most important part is the iterative section of the algorithm, in which the parameter learning takes place, is a loop which continues until some user defined rules are reached. Optimally, this would be when the algorithm finds a stationary point in the cost function, but this is not always viable because of the form of the learning rules. Since they are proportional to the derivative in each dimension of the parameter space, and since the derivative of smooth functions tend to decrease as a peak is approaching, the learning will keep slowing down as it nears its conclusion. Since one could theoretically let learning continue *ad infinitum*, while the parameters inch closer to their optimum, we have implemented some limiting conditions for stopping learning algorithm when the parameters are virtually unchanging over several iterations.

As the learning progresses, values of interest can be calculated for the intermediate parameters to track how the learning evolved, to be visualized in plots and figures by the end of the simulation. These are not really part of the learning algorithm per se, but are an integral part to analyze the system at hand.

In the appendix B we show some example code using functions developed for our research and investigation, including functional matlab codes for the algorithms discussed above.

2.3 Defining the Standard Network Parameters

When running simulations, we have tried to keep the choice of parameters as stable as possible. This is done to minimize any unwanted bias in the result, and to be able to compare different simulations with each other. Therefore we have certain standard values, and relations, we have used consistently across trials.

When generating a network, the parameters, or connection weights, are drawn from a normal distribution with a standard deviation related to the number of nodes they connect in the network. For example, the J's connect No observed with Nh hidden nodes, and are therefore drawn from $\mathcal{N}(0, 1/\sqrt{No * Nh})$.[1] The reason for doing this is to normalize the contribution of each parameter type in the energy function; we sum over all pairs of nodes connected, which has a total of $No*Nh$ terms in the case of the between-layer connections, $\mathbf{J}$. Because the weights are independent Gaussian variables, and since we know that $\sigma_{\sum_i X_i} = \sqrt{\sum_i \sigma_i^2}$ from statistics, we get that $\sigma_{\mathbf{J}} = \sqrt{\sum_i^{No*Nh} (1/\sqrt{No * Nh})^2} = 1$. This type of normalization makes sure our energy function gets an equal contribution from all parameter types on average, and we avoid a possible source of bias. The external fields $\mathbf{f}$ are chosen in a similar fashion when they are present, but standard deviation of their distribution is $1/N$.

In general, there is no *a priori* reason why we should aim for a standard deviation of one in the total contribution of a certain parameter type, as long as they are consistent for all types. Therefore we also have a scaling factor, g, in our model. This g-value can be thought of as a substitute for the inverse temperature commonly found in statistical physics, and should make the probability density across state space more or less uniform when it is decreased, or increased, respectively. It is one of the parameters we will be testing our model for in later sections, but otherwise it will be kept to one unless explicitly stated.

[1] Very late in the writing process, it was brought to the author's attention that it may be a better choice to draw the parameters from another distribution dependent on the total number of nodes. $\mathcal{N}(0, \sqrt{(No + Nh)/No * Nh})$ was suggested as a good factor, as it would make the the energy function act as a thermodynamic entity, scaling linearly with the system size, as it should. This should be taken into account in any figure or result where two systems of different sizes have been compared.

When it comes to the number of nodes in the network we have been working in a regime with a larger number of observed nodes. More specifically, we have $\approx 80\%$ of the network nodes as observable when generating data. This is because we prefer to study the simpler systems initially, and a choice we made to make the inference more precise since we will marginalize out less information when only using observed statistics. Additionally, we did most of the simulations on fairly small systems, usually with less than 20 nodes in total, to be able to calculate the energies and partition function exactly.

In the Boltzmann learning algorithm, we also have certain set standards. For the learning rate, we use $\eta = 0.5$. This is based on trial and error, and was small enough to avoid oscillations, but large enough to approach solutions relatively quickly. Since the algorithm is unlikely to ever converge completely on a stationary point, we decided to set two hard limits on the learning. First, we implement a rule that when the parameters have changed less than $\Delta = 10^{-8}$ over the last 50 iterations, the algorithm is considered to have converged. Additionally, if the learning does not reach this limit after 10000 iterations, we stop the algorithm, and consider it to have not converged.

These standards were initially chosen by empirical, yet mostly qualitative testing, but was continuously monitored, to avoid biases based on them.

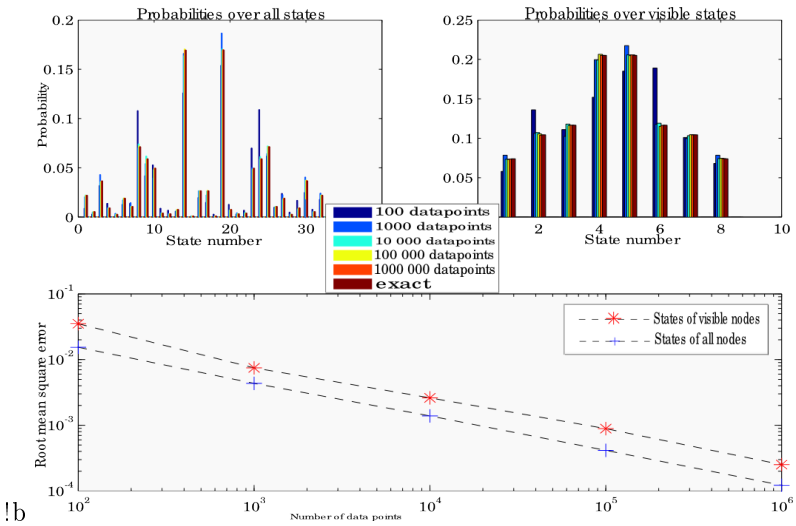
2.4 Simulation Algorithm Functionality

Here we go through some results of the algorithms introduced above, which were used to make sure that the codes and simulations used are functioning properly. If the Metropolis algorithm for generating data sets, for example, does not generate data from the equilibrium Boltzmann distribution, we will probably not get any useful results, regardless of how advanced our analysis is. Similarly, if the Boltzmann learning algorithms are dysfunctional, our conclusions about the curvature of the likelihood and the vicinity of peaks would likely be flawed.

2.5 Performance of Data Generation

First we asses the performance of our Metropolis algorithm for sampling data from the distribution of interest. We know from statistics that the data generated probability density function's (PDF) deviation from the true model distribution should decrease proportionally to $N^{-\frac{1}{2}}$ as the sample size, N, increases. In figure 2.3 we visualize the precision of the data generated PDF as compared to the true for several different data lengths using histograms and a log-log plot of the root mean square (RMS) error of the states of the system.

Figure 2.3 shows the results of the data generation from one example network, a semi-restricted Boltzmann machine (sRBM) with three observed and two hidden



!b

Figure 2.3: *Precision of the Metropolis algorithm visualized through a comparison of the Probability density function from the sampled data and the analytical calulation. Top left: histogram of the number of observations of all states in the network. Top right: Histogram of he observations of observed node states. The different colors in the histograms correspond to different numbers of iterations in the Metropolis algorithm (observations in the data set). Bottom: The root mean error in the data PDF as a function of data points. The two lines correspond to the two top plots.*

nodes and no external field. The behavior seen is representative of that seen in any other network tested, and we can see some of the known properties of the PDF. For example, the symmetry of the PDF seen in the histograms is expected, and disappears when external fields are introduced.

In the histograms, the states are set up and numbered as if they were a statistical truth table, in which state 1 has all negative nodes. From there, it goes through all permutations of the states by flipping the spins in an ordered fashion (N'th spin flipped for every state, $N-1$ flipped every other, and so on), until all spins are positive in the 2^N'th state. Inspecting the difference in the height of the bars as we move from blue to red, or small to large data sets, we can see that the proportions of of steps spent in a state gets closer to the real, dark red, distribution. This behavior can be seen quantitatively by looking at the bottom plot, which shows the root mean square error in the proportion as a function of data length in a log-log scale. As expected it shows a linear decrease with a slope of $-1/2$.

2.6 Boltzmann Machine Inference

Moving on to check the inference of model parameters, we use a figure showing the convergence of learning in a sample system as an example of the general progress in the algorithm. Figure 2.4 shows the evolution of training in an RBM with ten

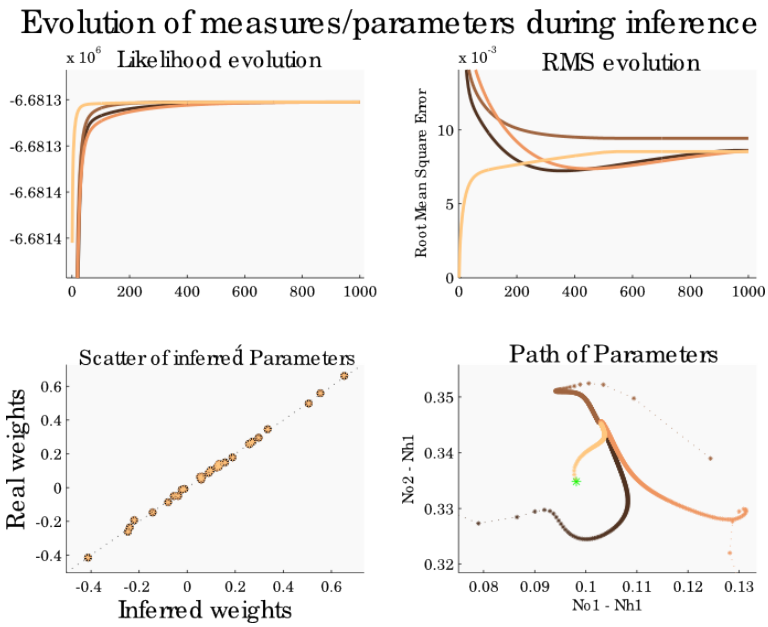


Figure 2.4: *The evolution of parameter optimization with four different sets of initial conditions, separated by color (gold is initialized at the generative parameters, while the rest are initialized at random points in parameter space). Top left: the likelihood value as learning progresses. Top right: the root mean square error of the parameters as learning progresses. Bottom left: scatter-plot of the final, inferred model parameters versus the real, generative values. Bottom right: path through a plane in parameter space as learning progresses.*

observed and two hidden variables. It presents and compares simulations from four different sets of initial conditions, and the plots contain most of the aspects found in such a simulation.

The two top plots of figure 2.4 show the two main measures we used to visualize the convergence of the learning process, the log likelihood value, $\mathscr{L}$ and the root mean square (RMS) error of parameters respectively. The development of the log likelihood value by itself tells us something about the gradient of the surface at the current point in parameter space, but it tells us nothing about the correctness of the parameters found. Supplementing with a plot of the error, such as the top right plot here, gives more of an indication of whether the parameters learned have a connection to the generative parameters.

The RMS is a standard measure for the error ε when fitting parameters cite?, and it gives a quantity of their average absolute deviation from the underlying model's true parameters. It is given by the formula

$$\varepsilon = \sqrt{\frac{1}{N}\sum_{i}^{N}(x_i - X_i)^2}$$

where x_i and X_i are the i'th inferred and true parameters, and N is the number of parameters in the model. Assuming that the algorithm infers the optimal parameters to the data, one could expect the error to depend on the data size, and it is possible to show that this deviation also declines as $D^{-1/2}$. It should be mentioned that, in the error presented here, the possible permutations in $\mathbf{J}$'s have been controlled for. In fact, after the learning has converged, we have a function finding the permutation yielding the lowest RMS, and we use that ordering of weights for calculating all values of the RMS presented in the figure.

The bottom left plot further investigates whether the learning led to the correct model, by making a scatter-plot of the inferred versus the generative parameters. When the points fall on the diagonal line, the inferred parameters coincide with the underlying, ground-truth model parameters, and the algorithm has found the correct solution. The scatter and the RMS figure lose some of their value when we do not actually know the parameters, but are helpful in this case for checking the algorithm, and they could be useful for checking consistency in results over several runs.

The final, bottom right, plot in figure 2.4 shows how the value of J_{11} and J_{21} change during learning, showing how the weights move through parameter space as learning progresses. The parameters tend to take some unintuitive route through parameters space to find its solution, and it indicates that it may not straight forward to predict the final solution permutation, even if we know the true solution and the initial guess parameters.

Taking all the information available in the figure, paints a picture of how sensitive the learning process is to the initialization of the system, but simultaneously how robust the resulting set of parameters is across trials. Even though this figure only shows a single simple example network, the result indicates a functioning algorithm.

Inference with a Semi-Restricted Architecture

Originally, we also wanted to do an in-depth assessment of the of the semi-restricted Boltzmann machine (sRBM) learning, but we quickly encountered problems with consistency in the parameter inference. The algorithm was highly sensitive to initial conditions.

Figure 2.5 is used to show the sRBM algorithms learning precision and its inconsistency and sensitivity to initial conditions. The sensitivity is probably most easily seen in the second row of the figure matrix, where the final parameters of inference are scattered against each other for two different initial conditions. The figure shows that there is almost no correspondence between the suggested parameters, even though these are results from the largest data sets. This is especially clear for the between layer connections in the right plot, and indicates problems with the algorithm. Either there are close-to-flat areas on the likelihood surface which cause the algorithm to remain virually unchanged for many iterations, or the likelihood surface is indeed flat, close to it or contains several, not permuta-

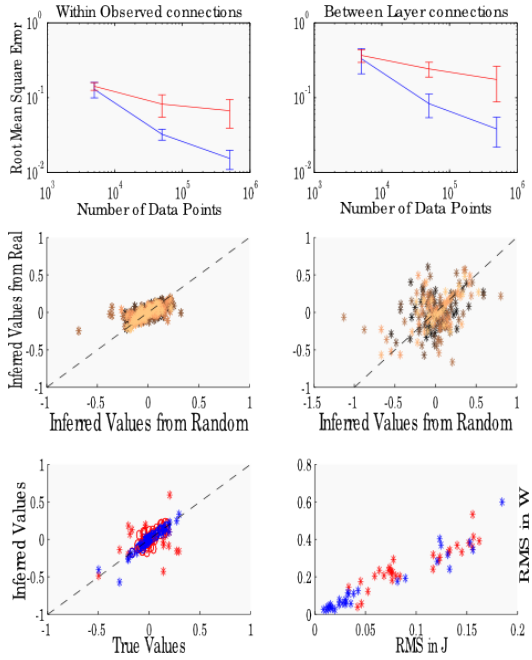


Figure 2.5: *An overview of the inference precision in the semi Restricted case. In the first two rows, the left column of plots concern the* $\mathbf{W}$*'s, connections within the observed layer, while the right column concerns the between layer connections,* $\mathbf{J}$*. The top row shows the final RMS error for different sizes of data, initializing with random (red) and true (blue) parameters. The second row contains scatter plots of the inferred values after initializing at the two different conditions. The fifth figure, bottom row to the left, shows a scatter plot for the inferred values of both* $\mathbf{W}$ *(circles) and* $\mathbf{J}$ *(stars), from one of the runs used in the first plot. Again, red marks random, while blue marks true, initial conditions. The sixth plot shows the covariance of the RMS error in between-layer and within layer inference. The same relation holds for both random (red) and true (blue) initial conditions.*

tionally related, peaks that are seemingly unrelated to the preferred optimum.

In the top two panels of the figure we plot the RMS error for within observed layer connections, $\mathbf{W}$, on the left, and the between layer connections, $\mathbf{J}$, on the right as a function of the sample data set size. Both are plotted for random and true initial conditions, and the same trends can be seen in both plots; the error is decreasing, but not necessarily in a linear fashion, and the slope depends on the type of initial condition. Even though the inference with both types of initial conditions are getting better when the data set size increases, the RMS value of the inference from random initial conditions seems like it will never get as good as inference from the true solution. This is problematic, as it means the inference is sensitive to initial conditions, and we cannot trust the parameters suggested by the algorithm.

The possibility that the relationship between RMS and data length can be non-linear in the log-log scale gives a chance that the error will eventually saturate, unlike what was seen for the RBM. Taking the divergence and possible saturation together may suggest that our algorithm is unable to perfectly infer the connectivity of an sRBM, even for infinite data set. Unfortunately, we were unable to research whether the error saturates, as the simulations could not handle significantly large data sets.

The bottom left panel of figure 2.5 shows a comparison of the end result of inference for one single realization of the sRBM, but for both initial conditions. Here we can see again that the resulting parameters of inference starting from true

valued (blue) and random (red) initial conditions are not the same. It is clear that the true initial condition yields a better result as compared to the true parameters, but that is not surprising, as the typical behavior of the learning was to just shift slightly from their initial guess, and then get stuck. As such, if one starts near the correct solution, one will tend to end near the correct solution.

From the last figure, we plot the final RMS in $\mathbf{J}$'s against the RMS in $\mathbf{W}$'s, to see how they covary. And it does seem like a clear pattern, when the J's are poorly inferred, so are the W's. This may not be surprising, since there is no fundamental difference between the two types of connections here, even though we have defined one subset as hidden, but is worth while to mention. In any case, this figure's hints at a lack of consistency in our sRBM inference was seen consistently, across trials, and led us to the decision to focus on the restricted architecture initially. In appendix A.1.4, we do however show how the smallest possible sRBM can be mathematically proven to be non-convex, and we hypothesize how this property could scale with network sizes.

Exactness of Monte Carlo

To be able to look at larger networks, we need to approximate the probability density function when calculating the correlations in the learning rules. This is because the number of states in the model grows exponentially with the number of nodes in the network, and it quickly gets impossible to exactly calculate the partition function, $\mathscr{Z}$, which contains one term for every state. Of course, this problem of an exponential number of states also makes the chance of properly sampling the distribution in the data generation exceedingly small. Therefore we should not expect to make a good reconstruction of the real parameters in any case, but we can see if there are some properties of the learning, and the error within it, that scales with the size of the system.

Monte Carlo (MC) methods are exact in the limit of infinite samples, and here we test the algorithm to make sure it works properly. In figure 2.6a we compare it with the exact algorithm using a small and simple network, checking for two initial conditions - true and random. From the scatter plot (bottom left) we can see that both algorithms converge on, more or less, the same solution whether the initial conditions were random or the underlying, generative parameters of the model. The slight differences in the inferred values is due to using a finite number of samples in each step of the MC learning. It can be seen more clearly in the bottom right panel, where we see that the MC method learning follows a similar, but noisy, path as the exact algorithm towards the inferred solution. While the exact algorithm settles on an optimum solution quite quickly, the MC method search around restlessly for some peak in the likelihood after reaching the vicinity of the right solution. This is to be expected though, as every learning iteration needs a new set of samples to approximate the correlations used in our learning rules, and the uncertainty in the MC sampling makes it unlikely for each of these samples to yield the same exact collection of correlations. Therefore each step will likely move the current guess in a different direction in parameter space. This means

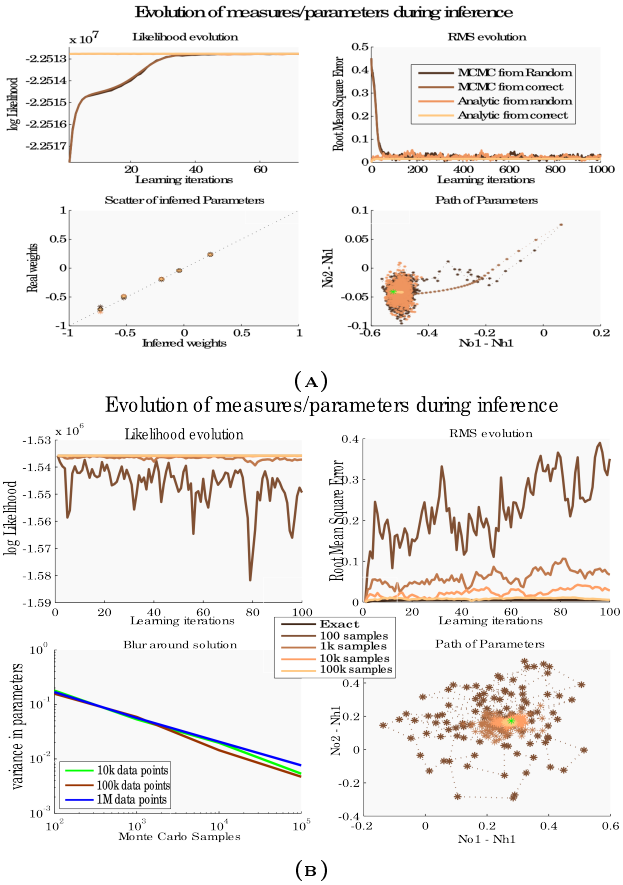


Figure 2.6: *A look at the learning evolution using the Monte Carlo sampling approximation for calculations of momenta, again using 5 observed and 1 hidden node.*
In (a) we show a comparison of the learning progression of the analytical algorithm and the Monte Carlo sampling approximation for two different initial conditions, black and brown have a random set of parameters, while orange and gold start at the real parameter values. Top left: Likelihood values during learning, zoomed in to the first 70 iterations of learning. Top right: Root mean square error during learning. Bottom left: Scatter plot of the final inferred parameters. Bottom right: Path through parameter space during learning for J_{11} and J_{21}.
In (b) we show the effect of increasing the number Monte Carlo samples. Here, the network was initialized with the correct set of parameters. Top left: The likelihood of the points suggested by the algorithm during learning. Top right: The RMS error of parameters for each iteration in learning. Bottom left: The standard deviation of the parameters suggested, as a function of MC samples, for different size data sets. Bottom right: A visualization of the path taken through parameter space for J_{21} and J_{11}, as learning progresses.

that the algorithm will keep changing its suggested optimal parameters, and never settle on a definitive best solution.

Even though the algorithm keeps its suggestions confined to the same area of the parameter space, it is hard to set some threshold for when to stop the learning. If we stop learning after an arbitrary number of steps of searching being restricted to the same area, we would expect the error to depend on the arbitrarily chosen number of MC samples. The relationship between the standard deviation from the solution and the number of MC samples is investigated further in figure 2.6b.

As discussed in the section about data generation, which uses the same type of algorithm, we expect to see a linear relationship between the standard deviation of the error and the number of MC samples on a log-log scale. This relationship can be seen in the bottom left panel of 2.6b, in which we also see that noise in suggested parameters when the algorithm is close to the optimum seems to be independent of the size of the data set. This could possibly be taken advantage of in the learning algorithm, by adaptively increasing the MC sample size when the parameters are deemed to be in the peak area. This could let us dip into the precision of large, with the speed of the small, sample sizes.

Of course, the distance from the generative parameters to the optimum in the likelihood *does* depend on the size of the data set, as presented in the section about the data generation, but that is besides the point as our algorithm can't be expected to learn anything beyond the data it is presented with. All in all, the MC algorithm seems to work well, and its precision is only restricted by the numbers of MC samples used to approximate the correlations, as expected. Unfortunately, we know that the sizes of data needed to properly sample a PDF is related to the dimensionality of underlying network, and we would need exponentially larger MC samples every iteration for comparable results when increasing the network size. As such, even though the MC methods is exact in the limit of large sample sizes, we were unable use it in networks that were significantly larger than the systems we could simulate using analytical calculations. We decided to focus our attention on these analytically solvable problems for the rest of this project.

Chapter 3

Restricted Boltzmann Machine Learning

In this chapter we make a more thorough investigation of the Boltzmann machine learning algorithm. We asses how sensitive the precision of inference and likelihood surface curvature is to changing the number and values model parameters. We aim to investigate the surface of the log likelihood function, $\mathscr{L}$, both in a qualitative and quantitative fashion, to find out what we can say about its curvature and convexity. Finally, we look into the apparent permutational invariance of the probability distribution over observed states, and how the algorithm evolves to end up in one solution rather than another.

3.1 Inference Sensitivity in Restricted Boltzmann Machine Learning

In the first section of this chapter, we assess how the learning algorithm is affected by varying different aspects of it. Initially we check its dependence on number of samples in the data set, and how well it regenerates the distribution we draw the true model parameters from. Secondly, we research the algorithm's sensitivity to its initial conditions, and ask whether we can expect to regenerate the ground truth parameters if we, theoretically, had no prior knowledge about how they may be valued. Before moving on to the investigation of the likelihood function itself, we investigate how the inference sensitivity depends on the number of nodes, both hidden and observed, in the network, and the strength of the connections between them.

3.1.1 Sensitivity to the Data Set Size

Since our goal with the restricted Boltzmann machine (RBM) learning algorithm is to approximate the model parameters of a probability density over some space of observable states, we would expect the power of it to depend on the amount of data

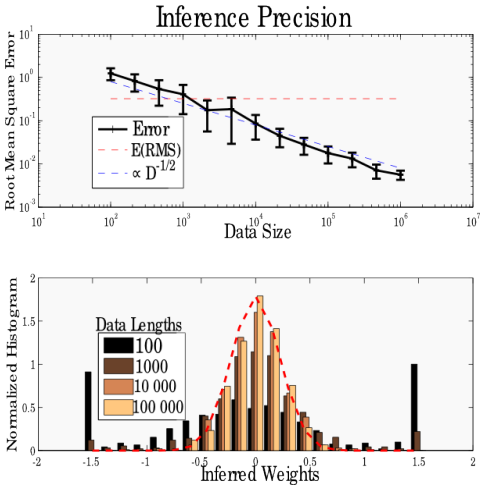


Figure 3.1: *Illustrating the RBM learning precision as a function of data sets. The top plot shows the log-log scale linear reduction in inference error as a function of data length, while the red dashed line shows the expected RMS of a random set of parameters drawn from the same distribution as the true parameters. The bottom plot shows a histogram of the inferred parameters of 30 distinct runs and four data set sizes. They are compared to the red dashed line which represents the true underlying distribution of weights. In each simulation we used a 8-by-3 network with a standard distribution of the weights ($g = 1$).*

we have available. In figure 3.1 the observed, along with the expected, reduction in inference error as a function of data set size, can be seen in the top plot. They falls of linearly with a slope of $-1/2$ in the log-log scale and is fairly consistent across trials. For very small data sets, however, the inference error exceeds the expected error if one was to draw completely random parameters, $E(RMS)$, marked by the red dashed line.

The fact that our algorithm does a worse job than randomly guessing parameters, seemed strange at first, but we soon realized a possible explanation for it. There is no inherent reason why the algorithm should suggest parameters drawn from the same distribution that the ground truth parameters were drawn from, and in the bottom plot, we can see the distribution of inferred parameters in histograms for four different data lengths. It is quite clear that the parameters inferred from small sets of data seem to be from a different distribution than the true parameters, visualized by the dashed red line.

The heavier tails seen in the black and brown histograms (cumulated in the outermost bars in the figure) are due to having too few data points. This leads to skewed distributions in which certain states are sampled too frequently relative to their true probability. Such an over-representation of some states will lead to observing too strong correlations in the data, and since the learning rules depend directly on these correlations the algorithm is bound to suggest too strong weights. Taking this to the extreme, with only one sample in the data set, the issue becomes more obvious. In that case, all nodes will be perfectly correlated or anti-correlated, which would lead to suggested weights of $-\infty$ and ∞ for all parameters, independent of what the true model parameters really were.

In the supplementary figure C.1 we show the relation between the distribution of the inferred weights as a function of the data length in a more rigid way. With the Küllback-Liebler divergence, $D_{KL}(P||Q) = \sum_s P_s \log(P_s/Q_s)$, mentioned in

an earlier section, we have a quantitative formula for comparing two distributions P and Q. Using this formula, we compare the distribution used to draw the true weights, $P = \mathcal{N}(0, 1/\sqrt{NoNh})$, with the distribution of the inferred weights, Q. The figure shows that the KL-divergence decreases as we increase the size of the data for both the 10-by-2 node, and the 5-by-1, networks.

We hypothesized that D_{KL} would saturate at some data set size, due to the fact that we only do a finite number of inference runs, which would leave us with an incomplete sampling of the distribution. This, as with most sampling issues, will yield an approximation error, which would not be overcome by increasing the size of the data used for inference. Signs of saturation can be seen for the smaller system, and if we tilt our heads and squint it can also be seen for the 10-by-2 network. We did, however, not quantify this error, and leave it as an exercise for the interested reader.

Here we have discussed, and argued for, why the learning algorithm we use is sensitive to the length of data observed. Its error in inference falls off as a power law, having a scale-free decay with a slope of $-1/2$ when plotted in a log-log scale against the length of data, and it regenerates the statistics of the parameter distribution well when the data is sufficiently large.

3.1.2 Sensitivity to Initial Conditions

In the previous section, we initialized our learning algorithm with the true values of the parameters. This gave us confidence in that the result we got would be the best possible inference the algorithm could do, but it is not a very realistic scenario in real-world applications, where the true model parameter values are unknown. Therefore, in the following section, we aim to investigate the RBM algorithm's strength and consistency by assessing its sensitivity to initial conditions. To do

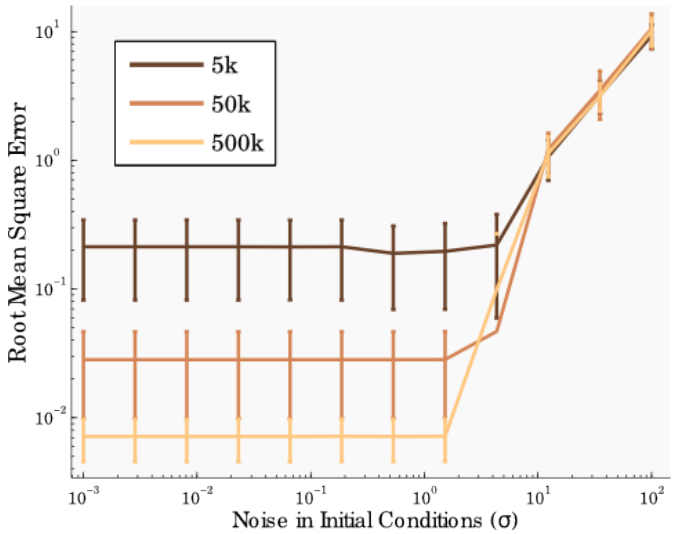


Figure 3.2: *The root mean square error of inference as a function of the amount of noise in the initial conditions. The standard deviation of the random variable used to add noise is varied along the x-axis, and the values marking the axis are the different standard deviations of the noisy σ used.*

this, we draw a random number, σ, and add it to the true values of the weights, and use these noisy parameters as the initial conditions. Incrementing the standard deviation of σ systematically, and doing the inference of a network several times with different initial conditions, lets us visualize how the inference error depends on the accuracy of our initial guess.

To get figure 3.2, we inferred the parameters of 30 different RBM's with ten observed and two hidden nodes, for three different sizes of data sets. The learning was initialized at noisy initial conditions using several different, logarithmically spaced values of noise, and we recorded the error in the parameter reconstruction. We initialized each of the network parameters in $J_{ij}^{guess} = J_{ij}^{true} + \sigma$, where J_{ij}^{true} is the ground truth value of the connection between the i'th observed and the j'th hidden node, and σ is a random variable, scaled by $1/\sqrt{NoNh}$ to make it comparable to the true parameters. We varied the standard deviation of σ from very small ($\mathcal{O}(10^{-3})$) to values ≈ 100 times larger than the standard deviation of the distribution we draw the true parameters from. The resulting root mean square error of the parameter inference as a function of the standard deviation of σ is presented in figure 3.2.

The figure shows how stable the inference is as long as the initial conditions are near the real solution. In fact, it seems as though the algorithm is completely insensitive to the initial conditions of inference as long as the initial conditions are of the same order of magnitude as the true parameters, or smaller. This was true for all sizes of networks tested, as can be seen from supplementary figures in C.3. However, from noise levels slightly bigger than the standard deviation of the true parameter distribution, we see that the error increases quickly.

This behavior for big values of noise is a signal of warning for learning weights in cases where we do not know the ground truth parameters of the model. In the supplementary figure C.2 we look at the inference error when initializing the learning at random values, independent of the true parameters as well. In it one can see that the error is stable, and comparable to the noisy initial conditions, as long as the initial guess has relatively small parameter values. This indicates that the inference is stable independent of the initial conditions, as long as they are not large compared to the standard deviation of the true model.

From this we could be tempted to use origo as the initial condition for all learning, but unfortunately, initializing all weights to zero in the RBM will lead to the parameter learning getting stuck there, and never learn. This is because origo is a stationary point of the distribution, as can be seen from the form of the learning rules and the likelihoods gradient. Origo is an unstable stationary point however, so it could be safe to initialize all parameters at values near, but strictly not equal to, zero to be safe from over-estimating parameters.

The instability of the learning algorithm can be due to several things. Some possibilities we would like to mention are the following: there could be peaks in the likelihood in which some parameters are large, while the others are small, but

independent of the true variables; alternatively, the likelihood could be flat, or near-flat, when certain parameters have large values, leading to an immature termination of the algorithm due to thresholding when the change is sufficiently slow; lastly, the algorithm could eventually converge, but since it starts so far away it needs to travel further to get to the right parameters, and it just has not gotten there yet, and was terminated du to taking too many iterations.

The two first suggested explanations cannot easily be separated, and by investigating properties of the learning evolution, it is unclear which of our possible explanations is more likely. The gradients at the endpoints were very small, and we never reached the limit of $10k$ learning steps, so we can say quite certainly that there are more or less flat areas outside the regions of the optimal parameter values. Whether the algorithm would eventually converge is hard to say, but when we removed the threshold for stopping learning when the parameters change very slowly over 100's of iteration, and let the algorithm run for $100,000$ iterations, it did not look promising. The parameters were still updating, but over the last ten thousand iterations the total change was $\mathcal{O}(10^{-10})$, and the parameters had been virtually unchanged for the last $99,000$ iterations.

Taking the figures and arguments of this section together, it seems like the learning algorithm is consistent in its inferred parameters as long as the initial conditions do not have parameters which are large compared to the true model parameters. If we initialize our parameters cleverly, in relatively small non-zero values, the expected inference precision seems to be dictated by the length of data, and its trial-to-trial variability is governed by the user defined cutoff-value in the algorithm.

3.1.3 Sensitivity to the Strength of Connectivity

Since we focus on restricted networks with external fields, f, set to zero, the only model parameters to fit are the weights connecting the observed and hidden layer. Since these parameters completely describe the probability distribution, we should make sure the values we use are well chosen. We saw, in the previous section, that having large parameters in the initial conditions of our learning process spelled disaster for the precision of inference. However, in the examples we saw, the true parameters were drawn from $\mathcal{N}(0, 1/\sqrt{NoNh})$. Will we see similarly drastic effects when changing the size of the model parameters?

To answer this, we will, in the following section, investigate how the precision of inference depends on the scaling of connection strengths. We use the parameter g to change the distribution from which we draw our model parameter as follows, $\mathbf{J} = \mathcal{N}(0, g/sqrtNoNh)$.

In figure 3.3, we show how the reconstruction error in a 10-by-2 RBM scales with the strength of the model connections for two data set sizes. The top figure shows the absolute error, while the bottom shows the relative error, normalized by the value of g. Both plots send the same message: there is an optimal range for the value of g if the goal is to be able to reconstruct the network connectivity. It seems

that choosing a g to be $\mathcal{O}(1)$ will lead to a slightly better inference precision, and this result seemed quite stable for all network sizes, when drawing parameters from $\mathcal{N}(0, g/\sqrt{NoNh})$. This shows that our choice of $g = 1$ was a good one, though we could possible have done even better with the inference on average for a slightly bigger factor.

The reason for the optimal range found for the values of g can be understood by considering the effect of strong and weak connections in the model. In the lower extreme, having a connection $J_{ij} = 0$ makes the nodes it connect, v_i and h_j, independent; they have no preference to be in the same or opposite states, and the model correlations between them would be zero. In the other end, when $J_{ij} = \pm\infty$, the two are completely determined by each other, they will always be in the same or opposite state of activity, depending on the sign of the weight, and both their data and model correlations are exactly ± 1. Now, if all the model parameters are small or large, artifacts of these effects show up in the data sampling, we will get an unsatisfactory sampling of the true distribution, and we will be unable to reconstruct the model parameters correctly.

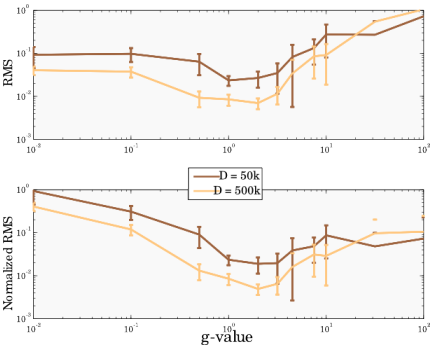


Figure 3.3: *Plotting both the actual, and normalized, root mean square error of inference as a function of the connection strength scaling factor, g, used when drawing the true model parameters. Using 10 observed and 2 hidden nodes and two data set sizes to assess the inference error when the true parameters are drawn from* $\mathcal{N}(0, g/\sqrt{(NoNh)})$. *The normalized RMS is scaled by* $g^{-1/2}$ *to control for possible bias from parameters being relatively bigger.*

For the small values of g, the PDF of the system will be virtually uniform across the state space, the Metropolis algorithm will accept most spin flips, and the small deviations from uniformity will be indistinguishable from the noise in data sampling. The learning algorithm will understand that the weights are more or less independent, and it will suggest small weights for the reconstruction, but the values suggested are hardly relevant. The fact that the inferred weights are small makes sure we get quite a small absolute RMS, but it falls through when the error is normalized by g.

When the underlying parameters are drawn from distributions with a large standard deviations, we get an opposite situation, where nodes depend strongly on the activities of the others. This gives rise to a heavily peaked PDF in which certain states have a significantly higher chance of being observed than others. When the Metropolis ends up in one of these high probability states, any perturbation will be very unlikely to be accepted, which may lead to getting trapped in certain

states over disproportionately long periods of time, making it over-represented in our data set. This is translated into too large data correlations, which skews the optimum of the learning algorithm accordingly.

The trough represents a sort of Goldilocks size of parameters, which has large enough valued parameters to make the PDF significantly non-uniform, but still small enough to avoid freezing into single states for too long. This leads to quite strong correlations which are significantly different from noise, but still small enough to avoid freezing into single states of activity for long times during sampling. This should lead to a more proper sampling the space.

Effect of g on the Likelihood Surface Near Peak Parameters

Our reason for having both an absolute and a normalized measure of RMS is that we thought certain properties of likelihood surface landscape may scale with the strength of connectivity. For example the region of convexity could be wider in an absolute measure for stronger connectivities, and as such lead to worse results when compared absolutely, but being relatively similar. This could help explain why the root mean square error is larger for those g's, as our stopping criterion may be reached further away from the functions actual optimum.

In figure 3.4 we show how averaged likelihood values depend on the distance from the inferred parameters for a range of different values of g. There are mainly three things that stand out here. First, the peaks seem to have more or less the same width for all strengths of connectivity, as can be seen by the lines all starting to drop off around at distances around $\mathcal{O}(1/\sqrt{NoNh})$ from the inferred point. This tells us that the top plot in figure 3.3 is the most relevant to use. Secondly, results for the three smallest g-values show no visible difference in the likelihood peak structure, both considering where the the values start dropping off, the slope as the parameters get far from the inferred values and the maximum values sampled.

Lastly, there seems like the strongly connected networks have larger areas with

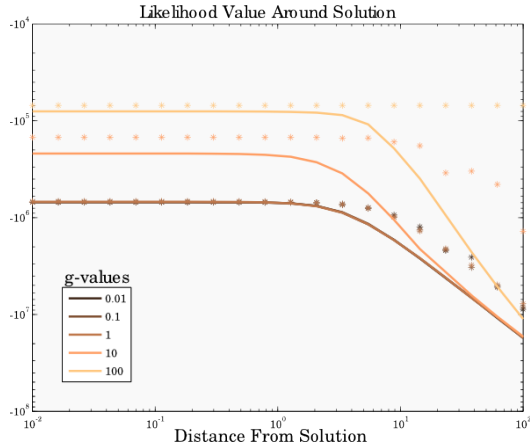


Figure 3.4: *Variation in the log likelihood value as a function of the distance, in parameter space, from the inferred values of learning a 10-by-2 RBM. The x-axis is denominated by $1/\sqrt{NoNh}$, so the distances are proportional to the parameter's standard deviation. The lines are averages of several sample points around the peak of the likelihood, and the stars show the maximum of the sampled values.*

near maximum likelihood. This is illustrated by the colored stars, marking the maximum likelihood value calculated of all the random points sampled at the given distances from the peak. In other words, changing the strength of connectivity by the scaling factor, g, has little effect on the average distance to the 'edge' of the peak. However, for large g's there seems like there are certain dimensions with a, more or less, flat likelihood. This can be understood in light of how the probability distribution responds to large parameters; they will make up a dominating term in the energy function, and as long as it is significantly bigger than the others, its exact value does not really matter.

The gold stars in figure 3.4, showing that there are many values far from the true parameters with near maximum values of $\mathscr{L}$, may help explain the resulting RMS of large noise values seen in 3.2. The near maximum values necessarily imply that there are significant areas of almost flat likelihood surface, in other words, it is not shocking that the learning algorithm stops at erroneous parameter values.

Even though this is only shown here for a single network, we show the same type of plots for two other networks in the supplementary figure appendix C.4, and similar properties are apparent in both of them. The width of the peaks are comparable for all values of g, weak connection strengths have indistinguishable peaks and maximal likelihood values and strongly connected networks can have maximally valued points far from the inferred parameters.

Taken together, these results show that there is a region of connection strengths which allow for better inference of the underlying model parameters due to more complete, and less biased or noisy, sampling of the state space. Additionally, we can say that the width of peaks in the likelihood function seem to be independent of the standard deviation of the weights, if we loosely define the end of a peak by the distance at which the average likelihood value is visibly lower.

3.1.4 Sensitivity to the Number of Parameters

Since the number of parameters in our model is directly dependent on the number of nodes we have in our network, one could expect the inference of some general network to be less precise for increasing network complexity given a set number of data points. As such, we wish to assess how sensitive our algorithm is to changing the amount of observed or hidden nodes present.

We are restricted to only look into small systems, though, since an RBM with No observed and Nh hidden nodes has 2^{No+Nh} available states. In addition to the number of states demanding an exponentially large data set to get a sufficient sampling of the space, this means that the number of terms in the partition function increases exponentially with the number of nodes. In the following we have therefore limited the investigation to containing no more than ten observed and seven hidden nodes.

In figure 3.5 we have plotted the RMS error of the inferred parameters after learn-

ing the connectivity of several different sized networks using differing sizes of data sets and a few properties pop out. First, from the left plot, it seems like the error in parameter inference is independent of the number of observed nodes, at least in the case with two hidden nodes, which is what we show here. However, this is only true for the case when inference starts from the true parameters, and we see that the final RMS seems to depend on both data length and number of nodes when the initial conditions are random (dashed lines). This observation is somewhat surprising, as most other simulations tended to be insensitive to the initial conditions. Interestingly, the deviation between true and random initial conditions is also most clearly visible for the larger data sets; a counter intuitive result. As the number of observed nodes increases, however, this deviation disappears, and the algorithm seems to be insensitive to initial conditions again.[1]

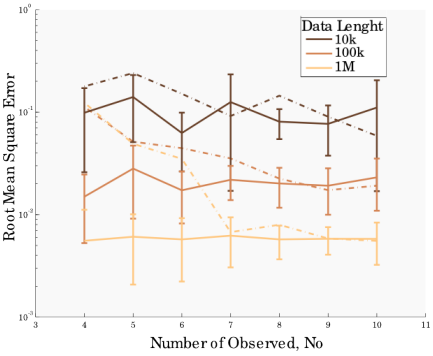
In the right plot of figure 3.5, the RMS seems to increase significantly with increasing the number of hidden nodes, for all sizes of data, but it also looks as if it will saturate at some point. The saturation is only really seen for the smallest data sets in figure 3.5b, and when we compare the error with the expected error from drawing random parameters, we see that they overlap quite well. We remember from earlier, that the statistical distribution of the reconstructed parameters can be biased towards larger values when the data sets are small, and this can explain why it saturates at levels above the red dashed line. I would expect the saturation levels of the brown and gold lines (larger data) to be at slightly lower errors than the black, but still above the red due to them sampling the distribution more correctly. This would be consistent with the analysis in 3.1, but we were unable to do this assessment do to limitations in computer storage capacity.

Taking the results of inference from random conditions in 3.5a and those from changing number of hidden nodes in 3.5b together, we could get the idea that it is the ration No/Nh that determines the precision it is possible to achieve with our algorithm. However, our preliminary tests showed no clear trends in how the reconstruction precision behaved for different No/Nh-ratios which could not be deduced from the plots already shown.

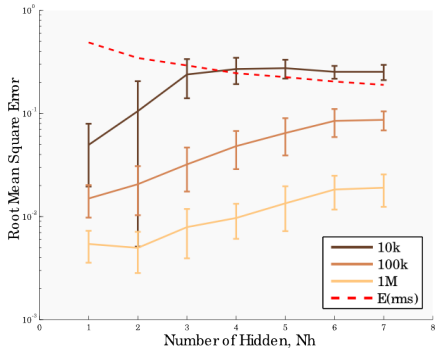
Predicting the Correct Number of Hidden Nodes in the Network

In this thesis, we are looking at systems where we have knowledge of not only the connectivity, but also the number of hidden units in the network. In a real-world example, however, where data is provided by some experimentalist's results, this information is obviously not given, due to the very nature of hidden variables. Therefore it would be interesting to see whether there is anything we can do to predict the existence and number of hidden. Using the correct number of nodes will not generally do a better job at regenerating the observed statistics [44], but it could yield a numerically more relevant result if each node is to be interpreted

[1] We are not sure how to interpret these results, and as the figures take day to make, and we first discovered the sensitivity to initial conditions days before the thesis deadline, we were unable to generate the required figures to back up any conclusion. It could be mentioned, however, that the independence of number of nodes when initializing the network at the correct solution was seen in networks in which $Nh = 4$ as well.



(A) Change in inference precision with increasing numbers of observed nodes in the model, keeping the number of hidden nodes stable at two. The dotted lines show the RMS after starting from random connections, while the solid are from the true parameters



(B) Change in inference precision with increasing numbers of hidden nodes in the model, keeping the number of observed nodes stable at eight. The red line shows the RMS for a random set of weights drawn from the same distribution as the generative parameters.

Figure 3.5: *The effect of inference precision when changing the number of nodes in the model while keeping the number of data points stable. The different colored lines correspond to different numbers of sampled points in the data set.*

as some entity, such as a neuron.

Dunn and Roudi [45] found that, by varying the number of hidden units in the reconstruction, a normalized likelihood had a maximum for the correct value of hidden units. Furthermore, the correct value of hidden nodes is accompanied by a minimum in the reconstruction error in the observed-to-observed weights. As such, one can predict what number of hidden nodes must have been present in the generation of the observed data, given that the scheme of connectivity is relevant. These results were achieved in a fully connected, kinetic Boltzmann Machine, and they used an advanced mean field method in their inference, but it would be interesting to see if we could get similar results in our simpler system.

We did some preliminary studies, looking for ways to predict the number of hidden nodes used to generate the data. Using standard information measures, such as Akaike's and Bayesian information criterion, as well as differently normalized likelihood measures, we were unfortunately not able to say anything useful about the number of hidden units from our investigation. The results can be seen in the supplementary figure C.5, and they did not look immediately promising. Therefore we leave this endeavor here for now, but note it as being an interesting problem to assess should we be confronted with real data for which the RBM seems to fit well as a parametrized model.

The results from this review of the effect of the number of nodes on the inference precision of the RBM seems to imply that we need to separate between the

effect of adding hidden and observed nodes to the network. The strength of inference seems to be reduced when our model has more hidden nodes, while adding an observed node either increases, or has no effect on, our precision in inferring the model connectivity depending on how many nodes we could previously observe. However, we have not been able to find a strong property to be used in predicting the number of hidden nodes in our model.

To sum up our findings concerning the sensitivity of the restricted Boltzmann machine learning, we can say that the most detrimental factor to precisely reconstruct the network connectivity is having a sufficiently large set of data. If the true model parameters are generated with parameters drawn from a distribution with $g \in [1, 10]$ we are likely to to do slightly better in our parameter reconstruction do to a more appropriate sampling of the state space. And, finally, having few hidden nodes in the data generating network seems preferable for the the inference algorithm.

3.2 Curvature of the Likelihood Surface

The likelihood function is the bread and butter of our version of Boltzmann machine learning. Since the learning rules are derived from this very function, and because they are designed to shift our parameter guess in the direction of its gradient, the likelihood surface geometry is of interest. As such assessing the shape of the likelihood surface is critical for understanding the strengths and weaknesses of our learning algorithm. If the function is not convex in relevant areas of the parameter space, has large flat areas or saddle points, our approach of using gradient ascent to find the maximum likelihood values will be futile.

We have already seen that the RBM learning algorithm is more or less only restricted by how completely the data samples the state space of the network. In the following we will focus on the function used to learn the parameters, and we will start by making a qualitative assessment of the surface curvature. We move on to use a qualitative measure, the Hessian matrix, to describe the curvature more rigorously, and investigate how this measure depends on the same properties we varied in the previous section.

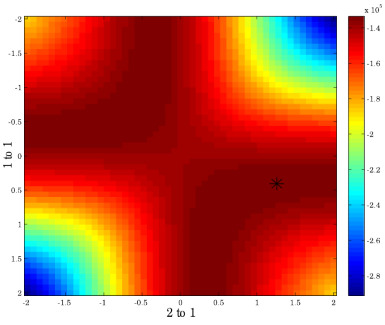


Figure 3.6: *The likelihood surface of the simplest RBM, with two observed and one hidden node. The black star marks the generative model parameter values.*

3.2.1 Qualitative Investigation

To get an intuition for the shape of the likelihood, we start by a purly observational inves-

tigation of the surface. Since $\mathscr{L}$ is a function of model parameters conditional on some data set, the dimensionality of it depends on the number of variables present in the model we want to infer the parameters of. As such, the only network with a likelihood surface we can visualize completely in a figure is the 2-by-1 RBM. It only has two parameters in its model, making it two-dimensional, and we can therefore potentially calculate and plot the likelihood value for every point in its entire, two dimensional parameter space in the paper plane.

In figure 3.6 we have done this for the area around the true values of parameter space, marked by the black star. The initial result is not promising with regards to function convexity; there seems to be a large region of the likelihood surface that has a similar value of $\mathscr{L}$ as the true parameters. This means that if we just follow the gradient of the function from some random initial condition until it is virtually zero, we have no way of predicting whether we end up in the correct solution.

In fact, it can be shown analytically that the simplest RBM network is not convex. Considering the true weights J_{11} and J_{21}, we can easily show that there are alternative sets of weights, $J11^*$ and J_{21}^*, which has the exact same probability distribution over the observed states as the true parameters. Starting from equation 1.1, and using properties of hyperbolic functions, some algebra and arithmetic gives us

$$\begin{aligned} P_{2x1}(v^s) &= \frac{1}{4}\big(1 \pm \tanh(J_{11})\tanh(J_{21})\big) \\ \implies \tanh(J_{11}^*) &= \frac{\tanh(J_{11})\tanh(J_{21})}{\tanh(J_{21}^*)} \end{aligned}$$

which has a solution for any, non-zero J_{21}^*. In other words, this RBM is not only non-convex, it has an infinite number of points in parameter space with exactly the maximum likelihood value. In fact it also has an infinite number of valid solutions when $J_{ij}^* = 0$, given that one of the original weights was also zero.

Luckily, we know that properties of simple systems do not always carry over to more complex and high dimensional versions of the problem [12], so there may be hope for convexity in higher dimensional problems. Unfortunately, it gets hard to survey the likelihood surface by eye when the network contains more than two parameters, as we have restricted tools for visualization. In any case, we have tried to give an idea of the shape of a likelihood function surface in an RBM with eight observed and three hidden nodes. The resulting plots of this can be seen in figure 3.7.

The three surface plots are generated in a similar way to 3.6, but because there are so many more variables in the model visualized here, we have done one thing differently: every parameter except two has been clamped to certain values while we systematically sweep the two last two parameters over a range of values, calculating and plotting $\mathscr{L}$ for every point. This lets us inspect a two dimensional slice of the multidimensional likelihood hyperplane. In 3.7 we hold the clamped variables at different values for each plot, effectively sampling three different, but

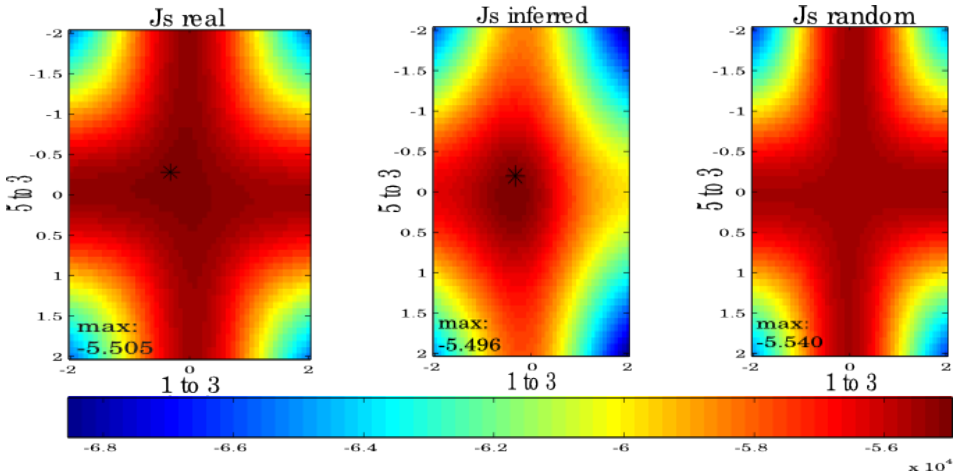


Figure 3.7: *Surveying the Likelihood surface of an RBM with 8 observed and 3 hidden nodes in three different areas of the parameter space. In all figures we sweep over a range of values for the weights between observed number 1 and hidden node 3, and observed number 5 and hidden node 3. In the left figure, the remaining parameters are held at their correct values - the values which they had during data generation. In the central figure, the remaining weights are held at the inferred values. In the right figures, all parameters were given a random value drawn from the same distribution as the real parameters. The black stars mark the true and inferred values, of J_{53} and J_{13}, in the left and center plot respectively. The value in the bottom left corner of each plot denotes the maximum likelihood value encountered during each local sweep.*

parallel plains in parameter space. From left to right, the parameters are clamped at the inferred, the true and some random set of parameters.

At first glance, they all look quite similar, with the same ranges of values and comparable geometries. However, the central image stands out a little bit with what looks to be more of a peak near the black marker. This is somewhat comforting, as the remaining parameters were clamped at their inferred values as our program sweeps over values for J_{13} and J_{53} near their own inferred solution. Even though there does not seem to be any peaks in the other two plots, the value written in the bottom left corner of each image shows that the local maximum likelihood. Those values do indeed seem to be higher in the inferred solution than elsewhere in parameter space, even though only by a few percent.

Another property to notice, which was also typical for this type of analysis, is that there does seem to be areas with a virtually flat likelihood surface, stretching quite far from the true solution. Even though they are not true maxima, or indeed saddle points (as the gradient is not quite zero), they may cause problems for our learning algorithm. If we, for example, end up in one of these near flat areas, the learning will slow down significantly due to its update rate being proportional to the gradient and a small, static learning rate, η, and may even be considered to have converged. This is very hard to control for without close inspection on a network to network basis, and is very difficult when the ground truth of the connectivity is unknown.

From this qualitative overview, it seems like there is indeed a peak near the true model parameters. Unfortunately, there seems to be large flat areas, in other areas of parameter space, or near-flat dimensions stretching out far from the true values of the weights. We give another example like figure 3.7 from another slice through the parameter space in the supplementary figure C.6, in which the flat areas are even more apparent. These properties can give our algorithm trouble, making it slow at best, and completely stuck in at worst.

3.2.2 The Hessian Matrix

Now, this type of qualitative assessment is obviously not exhaustive, nor rigid enough to draw strong conclusions about the general curvature of the log likelihood surface. A quantitative measure, relevant to the issues studied here, is the so called Hessian matrix and its eigenvalues. The Hessian is a matrix containing all different partial second derivatives of a multi-variate function, F, making it an n-dimensional matrix (n is the number of variables, X, in the function), where each element takes the form

$$\mathscr{H}_{i,j} = \frac{\partial^2}{\partial X_i \partial X_j} F(X_1, X_2, ..., X_n)$$

For our purposes the general function $F(X_1, ..., X_n)$ will be the likelihood, $\mathscr{L}$, and the derivations of the exact form of the matrix for RBM's and sRBM's can be found in the appendix A.1.3. For the RBM architecture, which is what we will be reviewing here, the elements of the Hessian matrix take the form

$$\mathscr{H}_{xy} = D * \begin{cases} \langle v_i v_a \rangle_d - \langle v_i v_a \rangle_m - \langle v_i v_a h_j h_b \rangle_d + \langle v_a h_b \rangle_m \langle v_i h_j \rangle_m & \text{if } j = b \\ \langle v_a h_b \rangle_m \langle v_i h_j \rangle_m - \langle v_i v_a h_j h_b \rangle_m & \text{if } j \neq b \end{cases}$$

where $x = i + j$ and $y = a + b$.

Our Hessian matrix elements tell us something about the curvature of the likelihood function in any chosen point in parameters space. Using standard techniques from linear algebra, it is possible to transform the matrix in such a way that its only non-zero elements are on the diagonal. The values on diagonal after such a transformation are known as the eigenvalues, and each of them is associated with a certain eigenvector, given by the characteristic equation. In the case of the Hessian matrix of the likelihood function, the eigenvalues quantify the magnitude of the curvature of the likelihood in the direction of the eigenvector through parameter space. In other words if the Hessian has relatively large eigenvalues for a set of parameters, the curvature of in the corresponding point in parameter space is sharp. On the other hand, if the eigenvalues are near zero, there is very little curvature, and the function is close to linear.

If the Hessian eigenvalues of a function are all negative, we say that the function has negative curvature in that point, and if that very point is a stationary point of the function, we can identify it as a peak. Additionally, if all the eigenvalues are negative for every point in the parameter space, we say that the function

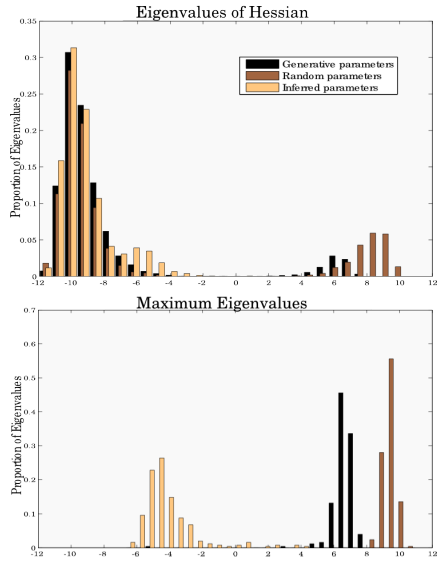


Figure 3.8: *Normalized histograms showing the Eigenvalues of the Hessian matrix of the Likelihood surface for an 8-by-3 RBM. After 50 realizations of the same network, the top figure shows all the eigenvalues of the likelihood in the inferred, true and random parameters (separated by color coding). The bottom figure shows the maximum eigenvalue in each of the three points for every realization of the network. We used our standard $g = 1$, we used $\approx 100k$ data points in every simulation, and the eigenvalues were calculated using inbuilt functionality in matlab. The random parameters are drawn from the same distribution as the true model parameters.*

is globally convex, since it can only have a single peak and has negative curvature over its entire parameter space. Since we wish to investigate the convexity of the likelihood function, it is a good idea to look at the distribution of eigenvalues for various points in its parameters space, to see if it lets us draw any strong conclusions about its curvature.

To do this, we implemented the formulas presented above, and used inbuilt matlab functionality to find the eigenvalues. Doing this let us plot histograms to review the eigenvalue distribution for a given size or type of Boltzmann machine. In figure 3.8 such distributions are presented for three different points in parameter space, after 50 distinct realizations of Boltzmann machine learning, where we used 8-by-3 RBM's.

We can see that the distribution for the three kinds of points in parameter space - generative, random and inferred plotted in black, brown and gold respectively - have certain quite similar properties. They all seem to have a similar number of negative eigenvalues, with a lighter tail of distinctly larger values. Histograms of the trial-to-trial variation of this distribution of eigenvalues is investigated in the appendix, in figure C.7, and is seen to be quite stable across the separate realizations of the system.

The most striking difference between the eigenvalue distributions for the different types of points is the magnitude of the values in the tail. It is large positive for the random, smaller positive for the true and mostly negative for the inferred parameters. If it is the case that the eigenvalues calculated using inferred parameters are indeed contained in its entirety on the negative values of the x-axis, it would mean that the curvature in the stationary point found by our algorithm is completely negative, in other words that it is a peak on the surface.

To investigate whether they are indeed all below zero, the lower figure shows only the maximum eigenvalues for each of the 50 learning simulations in each point. Though the vast majority of the Hessian eigenvalues calculated from the parameters inferred have a negative maximum eigenvalue, some are seen having positive values on our x-axis. It is important to mention here that we transformed the eigenvalues to have better separation of the relatively smaller values, using a logarithmic scaling. Since there may be both positive and negative eigenvalues, a standard logarithmic axis will not work, so we decided on keeping the axis linear, while transforming the values themselves using $\log(|Eig|) * sign(Eig)$. This transformation is an unambiguous mapping as long as $\log(|Eig|) > 0$ for all eigenvalues, but it is impossible to separate < 1 positive from complementary negative eigenvalues (f.ex. $\log_{10}(|-0.1|) * sign(-0.1) = \log_{10}(|10|) * sign(10)$), and vice versa. In other words, what looks like positive eigenvalues can be either very small negative, or the corresponding positive, eigenvalue.

Since the eigenvalues tend to be on the same order of magnitude as the size of the data, it is unlikely to find eigenvalues of an absolute value below 1, and we never ran into issues here. Therefore, the gold values with positive values, are indeed positive, which means that the inferred set of parameters seems not yield a true maximum of the likelihood. This was not true for most simulations, and having positive eigenvalues in the inferred parameters became more prevalent for more complex systems. However, in tests where we removed the threshold limit in the learning, and let the inference continue for a large number of iterations ($\mathcal{O}(10^5)$), we never had issues with positive eigenvalues in the inferred parameters, making it likely to just be an artifact of the threshold, or iteration limits, used to stop learning.

Effect of Distance from Inferred Parameters on Hessian Eigenvalues

Even though the Hessian matrix is most informative in stationary points of the function, it can be interesting to explore its eigenvalues in the vicinity of such points too. It can give us an idea of how the curvature changes as we move away from the peak, and quantify how far away from the peak the curvature of the likelihood surface could stay strictly negative. In the following we try to visualize how the Hessian eigenvalues depend on the distance from the inferred parameters.

In figure 3.9 we show some descriptive statistics about the eigenvalues of the Hessian matrix around the inferred parameters of an RBM with eight observed an three hidden nodes. Having sampled 50 random points in parameter space at several distances from the inferred solution, the eigenvalues were only strictly negative in the very close vicinity of the peak. Already at a distance 10^{-2} times smaller than the standard deviation of the distribution we sample the model parameters from, we see the overall maximum eigenvalue sneaking into positive values.

This means that the first positive second derivative occurs at points very near the optimum in parameter space. In other words, the convex region of the likelihood seems to be quite small, and cover only narrow areas of the state space. We

see similar properties in the supplementary figure C.8, where we study two other RBM's, with a different numbers of nodes. In both of those figures, we have a small range of noise levels around the inferred points in which all eigenvalues are negative. Both the systems presented in the supplementary figures also display the quite sudden transition into a region with large positive eigenvalues, but the transition happens at slightly different distances from the inferred values.

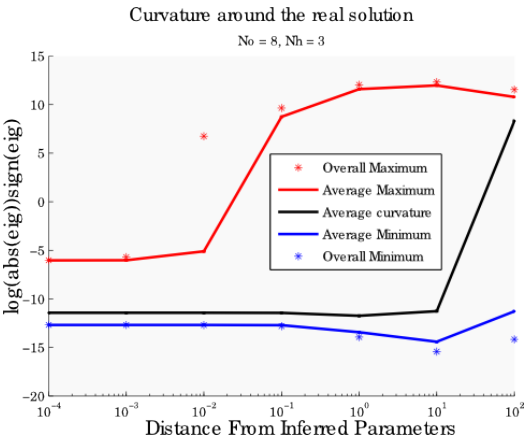


Figure 3.9: *Descriptive statistics for the eigenvalues of the Hessian matrix, normalized by the number of data points used, around the inferred point in parameter space for a 8-by-3 RBM. Again we have done the logarithmic transformation of the eigenvalues to have better control over the separation of relatively smaller values. We draw 50 random points the surface of hyper-spheres with a given radius centered at the inferred parameters, and calculate the eigenvalues there. Incrementing the radius systematically, we plot the overall maximum eigenvalue (red star), the average of the maximum eigenvalues from each point (red line), the overall average eigenvalue (black), the average of the minimum eigenvalues from each point (blue line) and the overall minimum eigenvalue (blue star) from that all the points sampled at each radius.*

Here, in figure 3.9 we see a sudden shift in the average curvature at the largest distances, a property that is not seen in the supplementary figure. This could just be due to a sampling issue, it can be an effect of the number of nodes in the system, or it can be that it will happen at larger distances for the systems presented in the appendix. When we increase the noise further, beyond standard deviations of $\mathcal{O}(100)$, however, we run into problems with computer tolerance of big numbers in the exponent when calculating the Hessian, and it gets difficult to study the behavior in depth.

From the general shape of the curves, the eigenvalues seem to be quite consistently distributed for different systems. This is seen more clear in the supplementary figure C.9 we present the histograms of all the eigenvalues used to produce both figure 3.9 and C.8. It shows that there is quite a standard development of the eigenvalue distribution as we move the suggested parameters away from the inferred peak. Going from a, more or less, unimodal distribution completely contained in the negative values, there is a steady trickle of eigenvalues switching sign (but keeping their magnitude) as the distance from the inferred point increases. When the distribution reaches a symmetric, bi-modal distribution, the magnitudes of the eigenvalues appear to start decaying. After this, we are unable to follow the development, as our algorithms cannot handle such large parameters.

It seems clear, however, that there is a region of strict convexity around the in-

ferred parameters. It appears to be small in general, but it's radius may depend on the number of nodes in the network. Taking the figures presented here, together with the distributions seen in figure 3.8 and the histograms presented in the supplementary figure C.9, we can suggest that the distribution of eigenvalues are quite consistent across networks, even though their scaling as a function of distance from the inferred peak may vary.

Effect of Connection Strength on Hessian Eigenvalues

In a previous section, we saw that the width of the convex region of the likelihood surface seemed to be independent of the connection strength scaling factor, g. In that case, however, we only looked at averaged likelihood values, and we could not say anything specifically about the curvature's dependence on the strength of connectivity. We have also seen that when the model parameters are sampled from a certain range, $g \in [1, 10]$, the inference precision is somewhat better than for models using other values for g. Here, we inspect the Hessian eigenvalues for the true and inferred parameters for the same range of g's previously investigated, to see whether they have an impact of the likelihood surface curvature as well.

In figure 3.10 we plot the maximum eigenvalues, averaged over $\approx$ 100 inference runs, calculated in the true (dashed line) and inferred (solid line) points of parameter space. We left out the error bars here, as they mostly clogged up the figure, without giving much information of interest. In the inferred points, the eigenvalues rarely exceeded zero[2] but were relatively small for both large and small values of g. This indicates that the algorithm did find a peak, but that there was at least one dimension in which the surface was more or less flat.

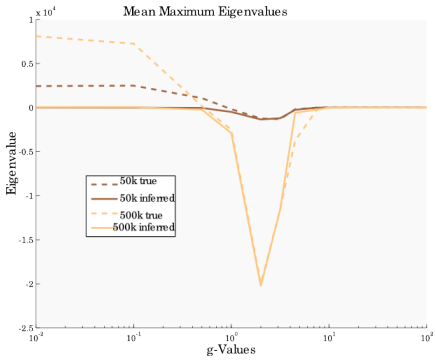


Figure 3.10: *Averaged maximum eigenvalues as a function of the scaling used to generate the model parameters. Plotted for two data set sizes, for both the true model (dashed) and the inferred (solid) parameters, in a network with ten observed and two hidden nodes. The weights were drawn from* $\mathcal{N}(0, g/\sqrt{No * Nh})$.

However, there is a region that behaves differently yet again, and it is the very same range of values which generated models we could infer better. We see a sudden drop in the maximum eigenvalues, which means that the likelihood surface is relatively more peaked for these strengths of connections. Interestingly, the maximum eigenvalues are also far below zero when calculated in the point of true parameters.

[2] As mentioned earlier, the positive eigenvalues found for the inferred parameters are likely to be artifacts of our, user-defined thresholds for when the algorithm has converged, and was not an issue when we let the algorithm learn for large numbers of iterations with no threshold for stopping.

We should be careful when interpreting, but such a result indicates that true parameters are within the convex region of the likelihood surface. If this is in fact the case, then in any data generating network with a $g \approx 2$, we can use our algorithm not only to find an optimal set of parameters, but also to set boundaries within which we will find the true model parameters. Without going overboard with interpretations of anecdotal results, however, we can say that using $1 < g < 10$ seems to be accompanied by interesting properties of the likelihood surface, which lead to preferable results in the inference of model parameters.

Another property to note is that the maximum eigenvalues found for strong connections were very near zero, for both true and inferred parameters. This means that there is likely to be at least one dimension in which the likelihood surface is more or less flat for such values of g, which could be interpreted to indicate that there are areas of a near-flat likelihood landscape away from the optimum too. This corresponds well with our earlier observations, and provides more evidence for why the algorithm would not converge completely on the optimal solution for large noise initial conditions with our threshold rules, as seen in figure 3.2.

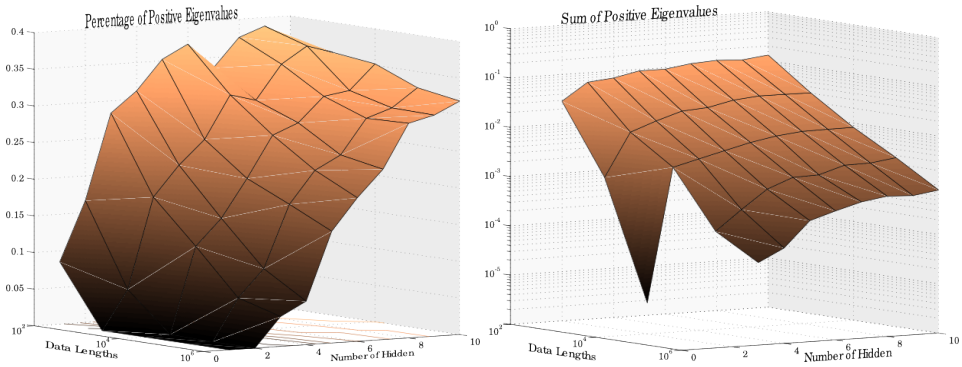
This assessment of the effect g has on the Hessian matrix eigenvalues gave us at least two interesting observations. First, models generated with the Goldilocks values of g seem to yield significantly different eigenvalues in both inferred and true parameters, than those generated with both weaker and strong connectivity. Secondly, for stronger connectivities, there seems to be more or less flat dimensions. This reverberates well with our observations about the maximum likelihood values presented in figure 3.3.

Effect of the Number of Parameters on Hessian Eigenvalues

We saw earlier that the precision in inference depends strongly on the number of hidden nodes in the network. We investigate how the positive eigenvalues, calculated from the true model parameters, picks up this dependence, and whether we can observe any trends of interest.

Holding the number of observed nodes constant, $No = 8$, and with $g = 1$, we generate a large data set ($\mathcal{O}(10^7)$) for an each number of hidden nodes. From each data set we subsample three smaller data sets which we use to calculate the Hessian matrix for the true model parameters. We record the Hessian eigenvalues, and in figure 3.11 we take a closer look at the positive ones. Here, the eigenvalues have been normalized by the number of data points, and their sum by the number of parameters, to make their values comparable.

In the figure we can see that a remarkably large proportion of all eigenvalues get positive as the network size increases, up to 40 percent when there are 10 hidden nodes present, even though the true weight values are still relatively close to the inferred solution in parameter space. Interestingly, the right plot shows that the sum of these eigenvalues are almost completely independent of the number of unobserved nodes in the network, and decreases in a scale-free fashion with in-



(A) (B)

Figure 3.11: *A visualization of positive eigenvalues of the Hessian matrix in the Likelihood function for the models true parameters, using different size networks and data lengths. The points are averaged over several ($\mathcal{O}(10)$) different realizations of the network. They all contained eight observed nodes, and for each realization a large data set was drawn from which smaller subsamples were used to calculate the Hessian matrix. In (a) we show the proportion of positive eigenvalues for different size networks and datasets, while (b) shows the sum of the positive eigenvalues for different size networks and datasets*

creasing data lengths. In other words, the values of the positive eigenvalues of $\mathscr{H}$ seems to get smaller on average as the data sets increase, and stay in a similar range as the number of nodes go up.

Comparing the proportion of the observed that is positive with the apparent eigenvalue distributions presented in supplementary figure C.9, we can hypothesize about the network sizes with $\approx 40\%$ positive eigenvalues sample points in parameter space that are a distance $\mathcal{O}(1/\sqrt{NoNh})$ away from what would be the inferred parameters of the data. This is, of course, based on qualitative investigation of anecdotal figures, and it can not be taken as conclusive evidence of anything.

It is, in fact, hard to say anything conclusive from these two figures, other than that it seems like the magnitude of the positive curvature falls as the complexity of the network increases. We get a general idea of the magnitude of the eigenvalues, however, since we have not added in what the gradient of the likelihood is in the points measured, we cannot really draw many conclusions.

To summarize our findings from this section, we can say that the multidimensional surface of the likelihood has peaks, surrounded by a small region of convexity, at points in parameter space relatively close to the true model parameters, and the corresponding sets of permuted parameters. The general distribution of eigenvalues changes from a unimodal, strictly negative, to a symmetric, bimodal distribution as the distance from the inferred parameters increases. And the eigenvalues of single points in parameter space sample these distributions quite consistently on a trial-to-trial basis.

Additionally, there are indications that the magnitude of the curvature decays,

making the surface more linear, when the guessed parameters are significantly larger than the true values used to generate the observed data. Finally, if the true model parameters are sampled from a wide distribution ($g > 10$), there is good chance that there is at least one dimension on the likelihood surface is flat, both in the true and inferred points of parameter space. On the other hand, if the true model was generated with $g \approx 2$, there is a good chance that the true parameters of the model will be contained within the inferred peak's region of convexity.

3.3 Permutations of Hidden Nodes and Local Convexity

So far in this chapter we have been talking about the inferred solution, almost as if it is a single point in parameter space. This is not the case, of course, because when we are working with mutually independent hidden nodes, no external fields, or both, there will be more than one equivalent solution. In other words, there are several equivalent peaks on the likelihood surface of the RBM we have been working on. More precisely, there should be at least $2*Nh!$ different local maxima, where the $Nh!$ stems from the possible permutations of the hidden nodes while the factor 2 is because of the sign invariance of the parameters when there is no external field. In the simulations done in previous sections we have reported that the learning in RBM's tend to converge on solutions with a low RMS error when compared to the true model parameters. We have not mentioned, however, that in the calculation of this root mean square, we have always controlled for the permutations of hidden nodes, and consequently chosen the permutation which yielded the lowest final RMS.

We briefly discussed the permutations, and why they must give equivalent probability density functions over the state space, in the introduction (take a look at figure 1.4 for a refresher), but since then we have left the issue untouched. Here we look into the local properties of the likelihood surface and the learning algorithm, in an attempt to learn how the parameters evolve to end up in a given permutation of the solution, and whether there are other points in parameter space that can yield the maximum likelihood value.

3.3.1 Learning with Clamped Parameters

To assess this, we decided to use a two-step process of learning the parameters in the network. In the first step we clamped one parameter, $J_{1,1}$ to a fixed value and let the rest of the weights converge to the maximum likelihood solution, given the clamped parameter value. Our goal here was to see if the final value of likelihood it ended up at was consistent across trials, and whether its likelihood value was equal to the global maxima. After recording the values of interest from the first stage, we un-clamp $J_{1,1}$ as well, and start the second stage of the inference. In this stage, all parameters are free to change, and we let them learn until they stop

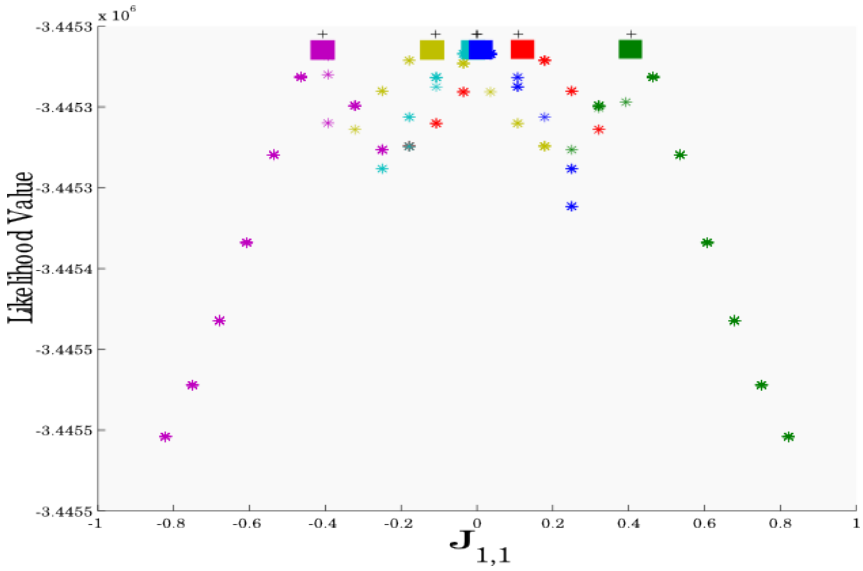


Figure 3.12: *Clamping a single parameter, J_{11}, and learning the remaining parameters, gives the likelihood values marked by stars. Releasing J_{11} and using the previously learned parameters as initial conditions for a new stage of learning yields the likelihood values, with updated J_{11}, marked by squares. The stars are colored according to which 'square they ended up in' after free learning. Black crosses are just references to the values of all connections from observed node 1 to hidden nodes, and their negatives.*
To generate the above plot we used a 10-by-3 RBM, with typical weight distributions ($g = 1$) and 50k samples in the data set. We clamped each value of $J_{1,1}$ 50 times and let all other parameters be drawn randomly from the same distribution as the true parameters. In the appendix supplementary figure C.10 shows a similar plot generated from a network with six observed and one hidden node, which may be simpler to decipher.

changing, just like in the normal algorithm.

We did this for different size networks, and we present two of the resulting figures in this report; in figure 3.12 the resulting figure is shown for a network with ten observed and three hidden nodes, and in supplementary figure C.10 for a simpler network with six observed and only one hidden node. Using $g = 1$ as usual, we draw parameters and generate a set of $500,000$ data points from the equilibrium distribution. We set 25 linearly spaced clamping values ranging over $\pm max(|\mathbf{J}|)$, and start the two-stage procedure.

In the figures we have marked the end-points of both learning stages by plotting the obtained value of $\mathscr{L}$ as a function of $J_{1,1}$'s value. The stars mark the final values of the first learning stage, while the middle of the squares mark the ultimate values of the second stage. We have color coded the stars according to which square they converged to during the un-clamped learning stage, and it is interesting to note that the final value of $J_{1,1}$ is not completely decided by its initial value. For example, when clamping $J_{1,1}$ to values around -0.2 it can end up in the purple, yellow, turquoise, and blue squares. Which square it ultimately

converges on is probably strongly related to the values of the remaining, randomly initialized, parameters.

A second thing to take note of, is the trial-to-trial consistency for both stages of learning. The clamped learning yields virtually identical values of likelihood every time, depending only on which square they will end up in after the second stage. The un-clamped learning is even more consistent - not only does it consistently yield one of six values for the, previously clamped, parameter $J_{1,1}$, but they all have the exact same likelihood.

Incidentally, the x-position of the six squares correspond very well with the black crosses at the top of the figure. These crosses mark the values of $\pm J_{1,1}$, $\pm J_{1,2}$ and $\pm J_{1,3}$ taken from the true model, which constitutes the complete list of connections to observed node $nr.1$ and their negatives, and as such exhausts the possible values $J_{1,1}$ can realize when taking into account the possible, equivalent permutations of the hidden nodes.

Taking all the information in figure 3.12 and C.10 together, it seems highly likely that the only peaks on the likelihood surface are $2 * Nh!$ equivalent permutations of the model parameters. At least, when considering values of the same order of magnitude as the true model parameters.

This sentiment rings true for what we have seen for the entire chapter; as long as we initialize the learning algorithm with weights of relatively similar magnitude as the true parameters, we can reconstruct the network connectivity arbitrarily well, only restricted by the length of the data, given that we control for permutations of the hidden nodes and sign invariance of the parameters. Of course, there are properties of the underlying model which helps in our quest to further improve the inference, such as having proper strength of connection, and containing more observed than hidden nodes, but in the end it seems the precision of RBM learning is only limited by the amount of data.

Chapter 4

Concluding Remarks

The work presented here should be considered a preliminary assessment of the strength and precision of RBM learning, and the convexity of the likelihood function associated with it. By investigating small networks with Boltzmann machine connectivity, we have been able to extract how certain properties of the learning algorithms developed depend on types, values and numbers of model parameters. This is by no means an exhaustive investigation of the efficiency of all Boltzmann machine methods, but merely a first step towards building up the currently lacking literature about the simplest versions of this stochastic neural network model - a model which is widely used with huge numbers of nodes, and approximate methods, due to its apparent functionality and simple implementation .

In this final chapter, we briefly discuss how to improve upon the algorithms applied and where move on from here, before trying to pull results of our investigation together into a conclusion.

4.1 Possible Future Directions

A common problem for numerical algorithms doing gradient ascent is that it is very hard to know when they have converged sufficiently close to a stationary solution. Of course, we would prefer to find the actual peak in the likelihood landscape, but this is not likely to happen, due to the nature of the algorithm used and computer precision. The gradient is unlikely to ever be calculated to *exactly* zero, and the algorithm will in most cases either start oscillating around some point in parameter space by overshooting the solution, or slow down to quickly to ever reach the solution.

These two problems are related, and depend strongly on the choice of learning rate. How we faced the issue here was finding a suitable value for the learning rate, η, by trial and error, keeping it constant and stopping the learning when the parameters were changing significantly slowly. This approach can lead the algorithm to suggest sub-optimal results, such as stopping the inference on near-flat areas of the likelihood. There are several ways to improve on our approach. For ex-

ample by using adaptive learning rates or more intelligent stopping criteria such as taking into account the Hessian eigenvalues. These are aspect to investigate which could potentially be applied to any method using gradient ascent or descent, and could therefore be beneficial for problem solving in a large range of fields.

On a more problem specific note, there are many things that can still be done to understand the most basic properties of Boltzmann machine learning on a more fundamental level. One thing to research closer would be the true number of peaks in the likelihood. We have already seen that there is likely to be at least two peaks for every permutation of hidden nodes when no external field is present, giving a total of $2 * Nh!$ equivalent maxima for the likelihood surface. However, we have not proven that these are the only maxima on the likelihood.

It is theoretically possible to analytically count the number of stationary points in any function, the log likelihood included, taking advantage of mathematical tools such as derivatives, delta functions and multivariate integrals. An expression for the number of peaks, N_P, could be derived from the general form $N_P = \int \delta\big(\sum_{ij} \frac{\partial \mathscr{L}}{\partial J_{ij}}\big)\partial\mathbf{J}$. This may look harmless in the form presented here, but we have no strong indications of this multi-dimensional integral leading to a simple closed form expression. Additionally, it would be necessary to find a way to separate the maxima from other stationary points, such as saddles and troughs, in which the sum of the derivatives are also zero. One way to get around this could be to multiply the integrand with the Heaviside step function of the Hessian eigenvalues. It could be arranged in a way so that the integral only gets a contribution when all the eigenvalues are negative, but is also likely to lead to an even messier expression...

Had it not been for the curse of time, we would have loved to look more in-depth into the Goldilocks-region of g-values. It is interesting both because it seemed to give more precise inference results, but also for the properties of the Hessian eigenvalues. Since the scaling factor we use here can be related to the inverse temperature in statistical mechanics, it can be tempting to look for evidence of critical points in the network. Even if there is no such point to be found, the mere fact that these values showed such peculiar properties justifies the resources needed to throughly investigate its effect on the network.

Another aspect that could be interesting to investigate more closely is the apparent saturation of inference error. We saw some anecdotal evidence for this when the number of hidden nodes got large, and the data set was comparatively small. What properties are responsible for such a saturation, and whether it even occurs, is not at all obvious, but if it does, one could possibly find an analytic expression for the number of data points needed to avoid saturation for a given network size.

We discussed the apparent saturation when comparing the error with the expected error when drawing random parameters for the model. In this comparison we have implied an expectation of the inferred parameters being drawn from the same dis-

tribution as the ground truth model parameters. This notion could be interesting to investigate as well; is the learning able to regenerate the distribution that the model parameters are drawn from? In our investigation we have only been using the normal distribution, and we have only been generating data from the correct underlying model framework. It could be interesting to study both of these; Can the inferred parameters tell us something about the underlying distribution of parameters, even if the actual values we find are incorrect? And what can a simplified network architecture tell us about the connectivity if the data is generated from more complex networks?

Another natural expansion of our investigation would be to study the impact of adding external fields in the analysis? From the form of the energy function, we can see that we will remove the sign invariance in the probability density. This would reduce the number of equivalent likelihood maxima, but it is not immediately obvious whether they are still local maxima or not. There may also be other, unforeseen aspects related to external fields, but we will not speculate further here.

Finally, it would be of interest to look more closely at how properties of the Boltzmann machine learning scales to larger networks. However, to avoid the enormous amount of computing power required for exact summation over, or time consumption of Monte Carlo sampling of, the exponentially growing number of states, other approximations are useful. Among the most popular ones are a class of methods stemming from the field of statistical physics, the mean field methods. In mean field approximations to Ising models, one seeks the best approximation to the probability distribution of states with the restriction that $P(\mathbf{h}|\mathbf{v})$ is separable in the hidden node variables [32]. Each of these are parametrized by a single variable, and a convenient choice turns out to be the magnetization μ_j, which is defined as the difference between the probability for the j'th hidden node to be $+1$ and -1. Three more advanced methods of approximations was discussed by Roudi *et al.*. [27] are 1) the independent-pair approximation, 2) the Sessak-Monasson approximation, a combination of the naive and independent-pair approximation, and 4) the inversion of TAP equations, in which the Onsager self-interaction terms are taken into account.

4.2 Conclusion

We have seen that the standard gradient ascent algorithm for finding the maximum likelihood parameters of a restricted Boltzmann machine is stable for small networks. It can infer the model parameters arbitrarily well, limited by the size of the data and the quality at which it has sampled the state space. To get parameters that are of relevant when assuming the hidden nodes are some unobserved, but existing entity, it is advantageous that the underlying model had its parameters drawn from a distribution with $g \approx 1$, and that the initial conditions of learning are not large compared to them.

From the investigation of the likelihood function surface, it seems like the magnitude, and sign of its curvature is quite consistent as a function of the distance to a peak. The same goes for the Hessian eigenvalues: they have quite a distinct distribution, and its variability as a function of the distance to a peak is quite stable. These properties seem to hold, not only on a trial-to-trial basis and for different realizations of a given size network, but also across networks sizes. Finally, the Goldilocks strengths of connectivity, give rise to a more strongly peaked likelihood surface and large deviations from these values for g open doors for maximum likelihood solutions with flat dimensions.

Bibliography

[1] Kandel, E. R. and Schwartz, J. H. and Jessel, T. M and Siegelbaum, S. A and Hudspeth, A. J. *Principles of Neural Science, Fith Edition*. McGraw Hill, 5 edition, 2012. ISBN 978-0-07-139011-8.

[2] Bear, M. F and Connors, B. W. and Paradisio, M. A. *Neuroscience: exploring the brain, Third Edition*. Lippincott Williams and Wilkins, a Wolters Kluwer business, 3 edition, 2006. ISBN 978-0-7817-6003-4.

[3] N. Brunel and M. C. van Rossum. Lapicque's 1907 paper: from frogs to integrate-and-fire. *Biol Cybern*, 97(5-6):337–339, 2007.

[4] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bull Math Biol*, 52(1-2):99–115, 1990.

[5] A. L. HODGKIN and A. F. HUXLEY. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol (Lond)*, 117(4):500–544, 1952.

[6] W. RALL. Branching dendritic trees and motoneuron membrane resistivity. *Exp Neurol*, 1:491–527, 1959.

[7] W. Rall. Theory of physiological properties of dendrites. *Ann N Y Acad Sci*, 96:1071–1092, 1962.

[8] E. De Schutter. Why are computational neuroscience and systems biology so separate? *PLoS Comput. Biol.*, 4(5):e1000078, 2008.

[9] Dayan, P. and Abbott, L. F. *Theoretical Neuroscience: computational and Mathematical Modeling of Neural Systems*. The MIT Press, 1 edition, 2001. ISBN 978-0-262-54185-5.

[10] Rolls, E. T. and Treves, A. *Neural Networks And Brain Function*. The Oxford University Press, 1 edition, 1998. ISBN 0-19-852432-3.

[11] M. Mezard and T. Mora. Constraint satisfaction problems and neural networks: A statistical physics perspective. *J Physiol Paris*, 103(1-2):107–113, 2009.

[12] Y. Roudi, S. Nirenberg, and P. E. Latham. Pairwise maximum entropy models for studying large biological systems: when they can work and when they can't. *PLoS Comput Biol*, 5(5):e1000380, 2009.

[13] B.A. CIPRA. An introduction to the Ising-model. *Am Math Mon*, 94(10): 937–959, 1987.

[14] M Opper and O Winther. Tractable approximations for probabilistic models: The adaptive TAP mean field approach. *Phys Rev Lett*, 86:3695–3699, 2001.

[15] Yasser Roudi and John A. Hertz. Mean field theory for non-equilibrium network reconstruction. *Phys Rev Lett*, 106,:048702, 2010.

[16] H. C. Nguyen and J Berg. Mean-field theory for the inverse Ising problem at low temperatures. *Phys. Rev. Lett.*, 109,:050602, 2012.

[17] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J Chem Phys*, 21 (6):1087–1092, 1953.

[18] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci USA*, 79(8):2554–2558, 1982.

[19] E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34(2):166–176, 2003.

[20] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Mol Syst Biol*, 3:88, 2007.

[21] C. Xianggao, H. Su, and L. Xiaola. Feature extraction using restricted Boltzmann machine for stock price prediction. In *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference*, volume 3, pages 80–83, 2012. doi: 10.1109/CSAE.2012.6272913.

[22] A. Pasini. *Neural network modeling in climate change studies.* Springer Netherlands, 2009. ISBN 978-1-4020-9117-9. doi: 10.1007/978-1-4020-9119-3_12.

[23] J. Hertz, Y. Roudi, and J. Tyrcha. Ising models for inferring network structure from spike data. 2011.

[24] M. Weigt, R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa. Identification of direct residue contacts in protein-protein interaction by message passing. *Proc Natl Acad Sci USA*, 106(1):67–72, 2009.

[25] T. R. Lezon, J. R. Banavar, M. Cieplak, A. Maritan, and N. V. Fedoroff. Using the principle of entropy maximization to infer genetic interaction networks from gene expression patterns. *Proc Natl Acad Sci USA*, 103(50):19033–19038, 2006.

[26] S. Cocco, S. Leibler, and R. Monasson. Neuronal couplings between retinal ganglion cells inferred by efficient inverse statistical physics methods. *Proc Natl Acad Sci USA*, 106(33):14058–14062, 2009.

[27] Y. Roudi, J. Tyrcha, and J. Hertz. Ising model for neural data: model quality and approximate methods for extracting functional connectivity. *Phys Rev E Stat Nonlin Soft Matter Phys*, 79(5 Pt 1):051915, 2009.

[28] E. Schneidman, M. J. Berry, R. Segev, and W. Bialek. Weak pairwise correlations imply strongly correlated network states in a neural population. *Nature*, 440(7087):1007–1012, 2006.

[29] E. Schneidman, S. Still, M. J. Berry, and W. Bialek. Network information and connected correlations. *Phys Rev Lett*, 91(23):238701, 2003.

[30] Gasper Tkacik, Elad Schneidman, Michael J. Berry, and William Bialek. Ising models for networks of real neurons. 2006.

[31] P. Smolensky. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter Information processing in dynamical systems: foundations of harmony theory, pages 194–281. MIT Press, 1986.

[32] J. Tyrcha and J Hertz. Network inference with hidden units.

[33] C vanVreeswijk and H Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274:1724–1726, 1996.

[34] G. E. Hinton. Machine learning for neuroscience. *Neural Syst Circuits*, 1(1): 12, 2011.

[35] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Sci*, 9:147–169, 1985.

[36] G.E. Hinton. Boltzmann Machines. http://www.cs.toronto.edu/ hinton/csc321/readings/boltz321.pdf, March 2007.

[37] A. Barra, A. Bernacchia, E. Santucci, and P. Contucci. On the equivalence of Hopfield networks and Boltzmann Machines. *Neural Netw*, 34:1–9, 2012.

[38] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput*, 18(7):1527–1554, 2006.

[39] Bengio, Y. and Lamblin, P. and Popovici, D. and Larochelle, H. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[40] Hinton, G. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14:2002, 2000.

[41] T. Tanaka. A theory of mean field approximation.

[42] M. Welling and G.E. Hinton. A new learning algorithm for mean field Boltzmann machines. In JosÃ©R. Dorronsoro, editor, *Artificial Neural Networks â ICANN 2002*, volume 2415 of *Lecture Notes in Computer Science*, pages 351–357. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44074-1. doi: 10.1007/3-540-46084-5_57.

[43] S. Kullback and R.A. LEIBLER. On information and sufficiency. *Ann Math Stat*, 22(1):79–86, 1951.

[44] N Le Roux and Y Bengio. Representational power of restricted boltzmann machines and deep belief networks. Technical report, 2007.

[45] B. Dunn and Y Roudi. Learning and inference in a nonequilibrium Ising model with hidden nodes. *Phys Rev E*, page 022127, 2013.

Appendix A

A.1 Derivations

A.1.1 Maximum Entropy Distribution

We are seeking the least structured, or equivalently, the maximum entropy, distribution available that agrees with the observed means and correlations in our data. This can be achieved by maximizing the entropy, $H(P)$, under the constraints that the probability distribution, P, is normalized, and that it yields means and correlations agreeing with those calculated from the data

By way of Lagrangian multipliers, this amounts to:

Function to be maximized:	$H(P) = -\sum_s P_s \log(P_s)$
Under constraints:	
Normalized distribution	$\sum_s P_s = 1$
Observed correlations	$E[v_i v_k] = \langle v_i v_k \rangle$
Between-layer correlations	$E[v_i h_j] = \langle v_i h_j \rangle$
Observed expected values	$E[v_i] = \langle v_i \rangle$

where the notation $\langle \dots \rangle$ denotes averages calculated from the data.

When attempting to maximize a function, $f(\mathbf{x})$, under a set of constraints, $g_i(x_i) = C_i$, by using Lagrange multipliers, λ_i, the standard approach is to define a Lagrangian

$$L = f(\mathbf{x}) + \sum_i \lambda_i (g_i(x_i) - C_i)$$

and proceed to differentiate L with respect to all the function variables, $\mathbf{x}$, as well as every λ_i. Demanding that every derivative equals zero, gives the parameters for a stationary point in $f(x)$, and we can explore the solutions to find the points of interest.

In our case, the Lagrangian turns out to be

$$\begin{aligned} L = &- \sum_s P_s \log(P_s) + \sum_{ik} W_{ik} \Big(E[v_i v_k] - \langle v_i v_k \rangle \Big) + \sum_{ij} J_{ij} \Big(E[v_i h_j] - \langle v_i h_j \rangle \Big) \\ &+ \sum_i f_i \Big(E[v_i] - \langle v_i \rangle \Big) + \lambda \Big(\sum_s P_s - 1 \Big) \end{aligned}$$

$$= -\sum_s P_s \log(P_s) + \sum_{ik} W_{ik}\Big(\sum_s v_i v_k P_s - \langle v_i v_k\rangle\Big) + \sum_{ij} J_{ij}\Big(\sum_s v_i h_j P_s - \langle v_i h_j\rangle\Big)$$
$$+ \sum_i f_i\Big(\sum_s v_i P_s - \langle v_i\rangle\Big) + \lambda\Big(\sum_s P_s - 1\Big)$$

where I have called the multipliers constraining the observed correlations, the between-layer correlations and the means W_{ik}, J_{ij} and f_i respectively to be consistent with the rest of the thesis. The multiplier for the normalization constraint is still called λ.

Now, the next step is taking the derivative with respect to the variables of the Lagrangian (P, all W_{ik}, all J_{ij}, all f_i and λ), and setting them equal to zero. In our case, that gives

$$\frac{\partial}{\partial P_s} L = -\Big(\log(P_s) + 1\Big) + \sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i + \lambda = 0 \tag{A.1}$$

$$\frac{\partial}{\partial \lambda} L = \sum_s P_s - 1 = 0 \tag{A.2}$$

$$\frac{\partial}{\partial W_{ik}} L = \Big(\sum_s v_i v_k P_s - \langle v_i v_k\rangle\Big)\Big(\sum_s v_i v_k \frac{\partial}{\partial W_{ik}} P_s\Big) = 0$$

$$\frac{\partial}{\partial J_{ij}} L = \Big(\sum_s v_i h_j P_s - \langle v_i h_j\rangle\Big)\Big(\sum_s v_i h_j \frac{\partial}{\partial J_{ij}} P_s\Big) = 0$$

$$\frac{\partial}{\partial f_i} L = \Big(\sum_s v_i P_s - \langle v_i\rangle\Big)\Big(\sum_s v_i \frac{\partial}{\partial f_i} P_s\Big) = 0$$

Now, the variable that we are interested in here is the probability distribution, P. To get an expression for it, we rearrange (A.1)

$$\Big(\log(P_s) + 1\Big) = \sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i + \lambda$$
$$\log(P_s) = \sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i + \lambda - 1$$
$$P_s = \exp(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i) \exp(\lambda - 1) \tag{A.3}$$

Using this expression with the condition from (A.2), gives

$$\sum_s P_s = \sum_s \exp\Big(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i\Big) \exp(\lambda) \exp(-1) = 1$$
$$\exp(-\lambda) = \exp(-1) \sum_s \exp\Big(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i\Big)$$
$$\lambda = 1 - \log\Big[\sum_s \exp\Big(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i\Big)\Big]$$
$$\lambda = 1 - \log(\mathscr{Z}) \tag{A.4}$$

where

$$\mathscr{Z} = \sum_s \exp\Big(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i\Big) \tag{A.5}$$

is called the partition function.

By inserting the expression (A.4) into (A.3), to eliminate λ from the expression for P, we get

$$\begin{aligned} P_s &= \exp(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i) \exp(1 - \log(\mathscr{Z}) - 1) \\ P_s &= \exp(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i) \mathscr{Z}^{-1} \end{aligned} \tag{A.6}$$

Recognizing the exponent here as the negative energy function, $\mathscr{E}$, of a semi restricted Boltzmann machine, we get the following familiar expression for the probability distribution

$$P_s = \frac{\exp(\mathscr{E})}{\mathscr{Z}} \tag{A.7}$$

Now, notice that we only used one of the four constraints in this derivations. This is to keep it general, as the observed means and correlations, obviously, depend on the data set used. In main part of this thesis, we will use the remaining constraints iteratively in our simulation algorithms to find the optimal parameters J, W and f for our model.

A.1.2 Learning Rules from Log-Likelihood

Here we define the log likelihood function, find expressions for its gradient, and derive the learning rules used for our inference algorithm doing gradient ascent on the log likelihood surface. For this derivation I am assuming statistical independence between the points of the data sample, in accordance with the assumption that all observations are drawn from an equilibrium distribution. In this case, the log likelihood, $\mathscr{L}$, is defined as

$$\begin{aligned}\mathscr{L}(\mathbf{J}|\mathbf{data}) &= \log(\prod_{d}^{D} P(data_d|\mathbf{J})) \\ &= \sum_{d}^{D} \log(P(data_d|\mathbf{J})) \\ &= \sum_{d}^{D} \log(\sum_{\{\mathbf{h}\}} P(data_d|\mathbf{J},\mathbf{h}) \end{aligned} \tag{A.8}$$

Taking the derivative of $\mathscr{L}$ with respect to some parameter, X, and setting it equal to zero, gives us an expression for the stationary points in the log-likelihood function given some value for the other parameters. Using the Boltzmann distribution, (A.5) derived above, gives us

$$\begin{aligned}\frac{\partial}{\partial X}\mathscr{L} &= 0 \\ \frac{\partial}{\partial X}\sum_{d}^{D} \log(\frac{\exp(\mathscr{E}_d)}{\mathscr{Z}}) &= \sum_{d}^{D} \frac{\partial}{\partial X} \log(\frac{\exp(\mathscr{E}_d)}{\mathscr{Z}}) \\ &= \sum_{d}^{D} \frac{\partial}{\partial X}(\log(\exp(\mathscr{E}_d)) - \log \mathscr{Z}) \end{aligned} \tag{A.9}$$

As an example, taking the derivative with respect to one of the between-layer weights gives

$$\begin{aligned}\frac{\partial}{\partial J_{ij}}\mathscr{L} &= 0 \\ &= \sum_{d}^{D} v_i^d h_j^d - D\sum_{s}^{S} v_i^s h_j^s \frac{\exp(\mathscr{E}_s)}{\mathscr{Z}} \\ &= D\langle v_i h_j\rangle_{data} - D\langle v_i h_j\rangle_{model} \end{aligned}$$

$$\langle v_i h_j\rangle_{data} = \langle v_i h_j\rangle_{model}$$

From this we see that, for the function to be in a stationary point, it is required that all the correlations between observed and hidden nodes are the same whether they are calculated from the data, or exactly from the modeling equilibrium Boltzmann distribution. This result holds as long as the energy function, $\mathscr{E}$, is separable in the weights.

Depending on which parameter the likelihood function is being differentiated with respect to, and the form of the energy function, the end result will look somewhat different. But as we will see in the following sections, they all have the same general form.

Restricted Boltzmann Machine

In the case of the restricted Boltzmann machine (RBM), there are only between-layer weights present, which leads to energy and partition functions of the form

$$\mathscr{E} = \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i$$
$$\mathscr{Z} = \sum_s \exp(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s).$$

Using these, with a data set, we can construct an expression for the log-likelihood, or use expressions of the form (A.9) directly to find the learning rules associated with this system. The parameters one needs to consider when finding learning rules for the RBM are the weights, $\mathbf{J}$, and the external fields, $\mathbf{f}$. Taking the derivatives necessary to fill in (A.9) gives the following expressions

$$\frac{\partial}{\partial J_{ij}} \mathscr{E} = v_i h_j$$
$$\frac{\partial}{\partial J_{ij}} \mathscr{Z} = \sum_s v_i^s h_j^s \exp(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s)$$

$$\frac{\partial}{\partial f_i} \mathscr{E} = v_i$$
$$\frac{\partial}{\partial f_i} \mathscr{Z} = \sum_s v_i^s \exp(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s)$$

Inserting these expressions back into (A.9), where X is substituted for the corresponding parameter, gives

$$\begin{aligned}\frac{\partial}{\partial J_{ij}} \mathscr{L} &= \sum_d^D v_i^d h_j^d - D \sum_s v_i^s h_j^s \frac{\exp(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s)}{\mathscr{Z}} \\ &= D\langle v_i h_j \rangle_{data} - D\langle v_i h_j \rangle_{model}\end{aligned}$$

for the between-layer weigths, and similarly

$$\begin{aligned}\frac{\partial}{\partial f_i} \mathscr{L} &= \sum_d^D v_i^d - D \sum_s v_i^s \frac{\exp(\sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s)}{\mathscr{Z}} \\ &= D\langle v_i \rangle_{data} - D\langle v_i \rangle_{model}\end{aligned}$$

for the external fields.

Since the $\langle \ldots \rangle_{model}$ depend on all parameters in the model, the above expressions will only yield a true stationary point when all the formulas equal zero simultaneously. One way of finding this set of parameters is, after making an initial guess at their values, updating each one by adding a term proportional to their derivative to them. This, an

algorithm known as gradient ascent, should move the model to a coordinate in parameter space with a higher likelihood-value, and culminate in a stationary point of the likelihood surface.

Since each variables component of the gradient, a vector pointing in the direction of highest increase, is proportional to the difference between the correlations, gradient ascent leads to the following rules for updating the model parameters

$$\Delta J_{ij} = \eta(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \tag{A.10}$$

$$\Delta f_i = \eta(\langle v_i \rangle_{data} - \langle v_i \rangle_{model}) \tag{A.11}$$

Here, η is known as the learning rate, which is responsible for how large the steps taken should be. This parameter needs to be set, and adjusted, depending on the system being analyzed. It should be as big as possible to speed up the simulation, but small enough to avoid over-shooting the peak in the likelihood, leading to oscillations and slow, or no, convergence.

Semi-Restricted Boltzmann Machine

A semi-restricted Boltzmann machine (sRBM) is similar to the RBM, but has additional weights, $\mathbf{W}$, between the observed nodes. This changes the energy function, and subsequently the partition function, to have the following forms

$$\begin{aligned}\mathscr{E} &= \sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i \\ \mathscr{Z} &= \sum_s \exp(\sum_{ik} W_{ik} v_i v_k + \sum_{ij} J_{ij} v_i h_j + \sum_i f_i v_i).\end{aligned}$$

Since the only thing distinguishing these from the corresponding expressions for the RBM, is the addition of the term $\sum_{ik} W_{ik} v_i v_k$ in the energy function, and the analogous term in the partition, the derivatives with respect to $\mathbf{J}$ and $\mathbf{f}$ remain unchanged. Again, using (A.9) to find the gradient of $\mathscr{L}$ with respect to the intra-layer weights, $\mathbf{W}$, we get the following

$$\begin{aligned}\frac{\partial}{\partial W_{ik}} \mathscr{E} &= v_i v_k \\ \frac{\partial}{\partial W_{ik}} \mathscr{Z} &= \sum_s v_i^s v_k^s \exp\big(\sum_{ik} W_{ik} v_i^s v_k^s + \sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s\big)\end{aligned}$$

When this is inserted back into (A.9) we get

$$\begin{aligned}\frac{\partial}{\partial W_{ik}} \mathscr{L} &= \sum_d^D v_i^d v_k^d - D \sum_s v_i^s v_k^s \frac{\exp(\sum_{ik} W_{ik} v_i^s v_k^s + \sum_{ij} J_{ij} v_i^s h_j^s + \sum_i f_i v_i^s)}{\mathscr{Z}} \\ &= D\langle v_i v_k \rangle_{data} - D\langle v_i v_k \rangle_{model}\end{aligned}$$

Using similar arguments as in the RBM case, the update rules for connections between the observed nodes can be written as

$$\Delta W_{ik} = \eta(\langle v_i v_k \rangle_{data} - \langle v_i v_k \rangle_{model}) \tag{A.12}$$

The Issue with the Values of h in the Correlations

Since we do not actually know the values of the hidden node activity when observing a state, it may seem hard to find correlations between them and other nodes. We do, however, know some properties of the hidden nodes, and their interactions with their environment. which can help us in the task of calculating the correlations. For example, the nodes only take the values ± 1, and we have some parameters, $\mathbf{J}$, that connect their values to that of the observed nodes.

The following derives an expression for ΔJ_{ij}, equivalent to (A.10), which does not depend on the specific values of the hidden nodes. Again using the energy function of the sRBM for generality, and starting from (A.8), we derive an expression for the log-likelihood, and do the differentiations necessary to find a valid expression for its gradient with respect to $\mathbf{J}$.

$$\begin{aligned}
\mathscr{L} &= \sum_d^D \log(\sum_{\{\mathbf{h}\}} P(data_d|\mathbf{J},\mathbf{h})) \\
&= \sum_d^D \log(\sum_{\{\mathbf{h}\}} \frac{\exp(\mathscr{E})}{\mathscr{Z}}) \\
&= \sum_d^D \log\left[\exp(\sum_{ik} W_{ik} v_i^d v_k^d + \sum_i f_i v_i) \sum_{\{\mathbf{h}\}} \exp(\sum_{ij} J_{ij} v_i^d h_j^d)\right] - \sum_d^D \log(\mathscr{Z}) \\
&= \sum_d^D \left(\sum_{ik} W_{ik} v_i^d v_k^d + \sum_i f_i v_i\right) + \sum_d^D \log\left[\sum_{\{\mathbf{h}\}} \exp(\sum_{ij} J_{ij} v_i^d h_j^d)\right] + D\log(\mathscr{Z})
\end{aligned} \tag{A.13}$$

Now, this may look messy, but considering we only need the expression for differentiating to find learning rules, it is not too bad. Since we want to get rid of the dependence on $\mathbf{h}$, we need to rewrite the middle term and partition function of (A.13). The middle term can be rewritten in the following way

$$\begin{aligned}
&\sum_d^D \log[\sum_{\{\mathbf{h}\}} \exp(\sum_{ij} J_{ij} v_i^d h_j^d)] \\
=&\sum_d^D \log\left[\sum_{h_1=\pm 1} \sum_{h_2=\pm 1} \cdots \sum_{h_m=\pm 1} \prod_j \exp(h_j^d \sum_i J_{ij} v_i^d)\right] \\
=&\sum_d^D \log\left[\sum_{h_1=\pm 1} \exp(h_1^d \sum_i J_{i1} v_i^d) \sum_{h_2=\pm 1} \exp(h_2^d \sum_i J_{i2} v_i^d) \cdots \sum_{h_m=\pm 1} \exp(h_m^d \sum_i J_{im} v_i^d)\right] \\
=&\sum_d^D \log\left[(e^{\sum_i J_{i1} v_i^d} + e^{-\sum_i J_{i1} v_i^d})(e^{\sum_i J_{i2} v_i^d} + e^{-\sum_i J_{i2} v_i^d}) \cdots (e^{\sum_i J_{im} v_i^d} + e^{-\sum_i J_{im} v_i^d})\right] \\
=&\sum_d^D \log\left[\prod_j 2\cosh(\sum_{ij} J_{ij} v_i^d)\right] \\
=&\sum_d^D \sum_j \log\left[2\cosh(\sum_{ij} J_{ij} v_i)\right]
\end{aligned} \tag{A.14}$$

Getting back to the task of finding learning rules for $\mathbf{J}$'s, we insert (A.14) into (A.13) and differentiate with respect to J_{IJ}, the connection strength between observed node I

and hidden node J, much like in the equation (A.9) above. Since the first term in (A.13) is independent of $\mathbf{J}$ it disappears during the differentiation, leaving us with having to differentiate (A.14) and $D\log(\mathscr{Z})$. In the following, we will be differentiating with respect to a specific coupling, between the I'th observed and J'th hidden node.

$$\begin{aligned}
\frac{\partial}{\partial J_{IJ}}\mathscr{L} &= \frac{\partial}{\partial J_{IJ}}\sum_{d}^{D}\sum_{j}\log[2\cosh(\sum_{ij}J_{ij}v_i)] - \frac{\partial}{\partial J_{IJ}}D\log(\mathscr{Z}) \\
&= \sum_{d}^{D}\frac{\partial}{\partial J_{IJ}}\log[2\cosh(\sum_{ij}J_{ij}v_i)] - D\frac{\partial}{\partial J_{IJ}}\log(\mathscr{Z}) \\
&= \sum_{d}^{D}\frac{v_I\sinh(\sum_{i}J_{iJ}v_i)}{\cosh(\sum_{j}J_{iJ}v_i)} - D\frac{\frac{\partial}{\partial J_{IJ}}\mathscr{Z}}{\mathscr{Z}} \\
&= \sum_{d}^{D}v_I\tanh(\sum_{i}J_{iJ}v_i) - D\frac{\frac{\partial}{\partial J_{IJ}}\mathscr{Z}}{\mathscr{Z}} \qquad \text{(A.15)}
\end{aligned}$$

Separating each term in the partition function in a similar fashion as in the derivation of (A.14), we get

$$\begin{aligned}
\frac{\partial}{\partial J_{IJ}}\mathscr{Z} &= \frac{\partial}{\partial J_{IJ}}\sum_{\{\mathbf{v}\}}\left(\exp(\sum_{ij}W_{ik}v_i^s v_k^s + \sum_i f_i v_i)\sum_{\{\mathbf{h}\}}\exp(\sum_{ij}J_{ij}v_i^s h_j^s)\right) \\
&= \frac{\partial}{\partial J_{IJ}}\sum_{\{\mathbf{v}\}}\left(\exp(\sum_{ij}W_{ik}v_i^s v_k^s + \sum_i f_i v_i)\prod_j 2\cosh(\sum_i J_{ij}v_i)\right) \\
&= \sum_{\{\mathbf{v}\}}\left(\exp(\sum_{ij}W_{ik}v_i^s v_k^s + \sum_i f_i v_i) * \frac{\partial}{\partial J_{IJ}}\prod_j 2\cosh(\sum_i J_{ij}v_i)\right) \\
&= \sum_{\{\mathbf{v}\}}\left(\exp(\sum_{ij}W_{ik}v_i^s v_k^s + \sum_i f_i v_i) * 2v_I\sinh(\sum_i J_{iJ}v_i)\prod_{j\setminus J} 2\cosh(\sum_i J_{ij}v_i)\right)
\end{aligned}$$

Multiplying through each of the terms in this expression by $1 = \frac{\cosh(\sum_i J_{iJ}v_i)}{\cosh(\sum_i J_{iJ}v_i)}$, we get

$$\begin{aligned}
\frac{\partial}{\partial J_{IJ}}\mathscr{Z} &= \sum_{\{\mathbf{v}\}}\left(\exp(\sum_{ij}W_{ik}v_i^s v_k^s + \sum_i f_i v_i) * v_I\tanh(\sum_i J_{iJ}v_i)\prod_j 2\cosh(\sum_i J_{ij}v_i)\right) \\
&= \sum_{\{\mathbf{v}\}}v_I\tanh(\sum_i J_{iJ}v_i)\left(\exp(\sum_{ij}W_{ik}v_i^s v_k^s + \sum_i f_i v_i)\prod_j 2\cosh(\sum_i J_{ij}v_i)\right) \\
&= \sum_{\{\mathbf{v}\}}v_I\tanh(\sum_i J_{iJ}v_i)\left(\mathscr{Z} * P(\mathbf{v})\right)
\end{aligned}$$

In the transition to the last expression above, we have recognized the terms contained by the parenthesis as the numerator of the models probability of observing state $\mathbf{v}$. This can be seen by the separability of the energy function, and a derivation follows a very similar framework as the one seen leading up to A.13.

Inserting this equation into (A.15), we get

$$\begin{aligned}\frac{\partial}{\partial J_{IJ}}\mathscr{L} =& \sum_{d}^{D} v_I \tanh(\sum_i J_{iJ}v_i) - D\frac{\mathscr{Z} * \sum_{\{\mathbf{v}\}} v_I \tanh(\sum_i J_{iJ}v_i)P(\mathbf{v})}{\mathscr{Z}} \\ =& \sum_{d}^{D} v_I \tanh(\sum_i J_{iJ}v_i) - D\sum_{\{\mathbf{v}\}} v_I \tanh(\sum_i J_{iJ}v_i)P(\mathbf{v}) \\ =& D\langle v_I \tanh(\sum_i J_{iJ}v_i)\rangle_{data} - D\langle v_I \tanh(\sum_i J_{iJ}v_i)\rangle_{model}\end{aligned}$$

This result generalizes for any connection J_{ij}, and can be written as

$$\frac{\partial}{\partial J_{ij}}\mathscr{L} = D\langle v_i \tanh(\sum_i J_{ij}v_i)\rangle_{data} - D\langle v_i \tanh(\sum_i J_{ij}v_i)\rangle_{model}$$

The expression for the gradient is now on a form very similar to that of (A.10), only the $\mathbf{h}$ has been exchanged with the expression $\tanh(\sum J_{ij}v_i)$. We shall see that this substitution makes sense in certain systems, or analyses. Briefly explained, it is because it can be shown that the expected value of the j'th hidden node given some state of the observed nodes, $\mathbf{v}$, is $\tanh(\sum J_{ij}v_i^s)$, where v_i^sÂ denotes the activity of the i'th node in the s'th observed state.

This can be shown using some algebra, and it holds as long as the hidden nodes are all independent in the model.

$$\begin{aligned}E(h_J|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J}) =& 1 * P(h_J = 1|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J}) + (-1) * P(h_J = -1|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J}) \\ =& \frac{\exp(\mathscr{E}(h_J = 1|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J})) - \exp(\mathscr{E}(h_J = -1|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J}))}{\exp(\mathscr{E}(h_J = 1|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J})) + \exp(\mathscr{E}(h_j = -J|\mathbf{J},\mathbf{v},\mathbf{h}_{\setminus J}))} \\ =& \frac{\exp(\sum_{ik} W_{ik}v_iv_k)\exp(\sum_i f_iv_i)\exp(\sum_{ij_{\setminus J}} J_{ij}v_ih_j)}{\exp(\sum_{ik} W_{ik}v_iv_k)\exp(\sum_i f_iv_i)\exp(\sum_{ij_{\setminus J}} J_{ij}v_ih_j)} \\ & \cdots \frac{\left(\exp(\sum_i J_{iJ}v_i) - \exp(-\sum_i J_{iJ}v_i)\right)}{\left(\exp(\sum_i J_{iJ}v_i) + \exp(-\sum_i J_{iJ}v_i)\right)} \\ =& \frac{\exp(\sum_i J_{iJ}v_i) - \exp(-\sum_i J_{iJ}v_i)}{\exp(\sum_i J_{iJ}v_i) + \exp(-\sum_i J_{iJ}v_i)} \\ =& \tanh(\sum_i J_{iJ}v_i)\end{aligned}$$

These calculations show that we, in the learning rules for between-layer connections, $\mathbf{J}$, can exchange the the unknown, hidden, binary activity, h_j, with its fully observable, continuously valued, expected value, $\tanh(\sum_i J_{ij}v_i)$. It is analytically equivalent in cases where the hidden nodes are mutually independent, and the substitution $h_j \rightarrow \langle h_j\rangle$ can be used as a naive approximation in systems where the hidden nodes are also interconnected.

A.1.3 The Hessian Matrix

When surveying the shape of the likelihood surface, the Hessian matrix can be very helpful. It is basically a matrix containing all different second derivatives of a multi-variate function, so for a general function $f(\mathbf{X})$ of n different variables, it can be written in the following form

$$\mathscr{H} = \begin{bmatrix} \frac{\partial^2}{\partial X_1 \partial X_1} f(\mathbf{X}) & \dots & \frac{\partial^2}{\partial X_i \partial X_1} f(\mathbf{X}) & \dots & \frac{\partial^2}{\partial X_n \partial X_1} f(\mathbf{X}) \\ \frac{\partial^2}{\partial X_1 \partial X_2} f(\mathbf{X}) & \ddots & & & \frac{\partial^2}{\partial X_n \partial X_2} f(\mathbf{X}) \\ \vdots & & \frac{\partial^2}{\partial X_i \partial X_i} f(\mathbf{X}) & & \vdots \\ \frac{\partial^2}{\partial X_1 \partial X_{n-1}} f(\mathbf{X}) & & & \ddots & \frac{\partial^2}{\partial X_n \partial X_{n-1}} f(\mathbf{X}) \\ \frac{\partial^2}{\partial X_1 \partial X_n} f(\mathbf{X}) & \dots & \frac{\partial^2}{\partial X_i \partial X_n} f(\mathbf{X}) & \dots & \frac{\partial^2}{\partial X_n \partial X_n} f(\mathbf{X}) \end{bmatrix}$$

Like the normal second derivatives, this measure can tell us something about the curvature of the function being studied in any given point. Transforming it to a diagonal form, we also get an orthogonal set of vectors (its eigenvectors) informing us of the principle directions of the functions' curvature, with corresponding eigenvalues reflecting the magnitude of the curvature.

Restricted Boltzmann Machine

When working with a network with RBM architecture and no external field, there are only between-layer connections, meaning the only parameters of the model are the $No * Nh$ between-layer connections, $\mathbf{J}$. This gives a Hessian matrix containing elements of the form

$$\mathscr{H}_{xy} = \frac{\partial}{\partial J_{ab}} \frac{\partial}{\partial J_{ij}} \mathscr{L}$$

such that $a + b = x$ and $i + j = y$. Since we already know, from (A.10), that

$$\frac{\partial}{\partial J_{ij}} \mathscr{L} = D\big[\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}\big]$$

we can rewrite the equation of the Hessian as,

$$\begin{aligned} \mathscr{H}_{xy} &= \frac{\partial}{\partial J_{ab}} D[\langle v_i h_j \rangle_d - \langle v_j h_i \rangle_m] \\ &= D\big[\frac{\partial}{\partial J_{ab}} \langle v_i \tanh(\sum_i J_{ij} v_i) \rangle_d - \frac{\partial}{\partial J_{ab}} \langle v_i \tanh(\sum_i J_{ij} v_i) \rangle_m\big] \\ &= D\left[\langle v_i \frac{\partial}{\partial J_{ab}} \tanh(\sum_i J_{ij} v_i) \rangle_d - \frac{\partial}{\partial J_{ab}} \langle v_j \tanh(\sum_i J_{ij} v_i) \rangle_m\right] \\ &= \sum_d \left[v_i \frac{\partial}{\partial J_{ab}} \tanh(\sum_i J_{ij} v_i)\right] ... \\ &\quad - D \frac{\partial}{\partial J_{ab}} \sum_{states} \left[v_i \tanh(\sum_i J_{ij} v_i) \prod_j 2 \cosh(\sum_i J_{ij} v_i)\right] \mathscr{Z}^{-1} \end{aligned} \tag{A.16}$$

Where $\mathscr{Z} = \sum_v \prod_j 2\cosh(\sum_i J_{ij}v_i)$ like earlier, and we have used the functional similarity of $\mathbf{h}$ and $\tanh(\sum_i J_{ij}v_i)$.

Now, working with one part of this expression at a time, we can simplify it step by step as follows. Starting with the first term of (A.16), we get

$$\begin{aligned}
&\sum_d \left[v_i \frac{\partial}{\partial J_{ab}} \tanh(\sum_i J_{ij}v_i) \right] \\
=&\sum_d \left[v_i v_a \Big(1 - \tanh^2(\sum_i J_{ib}v_i)\Big) \delta(b-j) \right] \\
=&D\Big[\langle v_i v_a \rangle_d - \langle v_i v_a h_j h_b \rangle_d \Big] \delta(b-j)
\end{aligned}$$

where $\delta(b-j) = 1$ when $b = j$, and zero otherwise. It should be noted that we use h_j here for notational convenience only, and it is always substituted for $\tanh(\sum_i J_{ij}v_i)$ in this section.

For the second part of (A.16) we can write

$$\begin{aligned}
&D\frac{\partial}{\partial J_{ab}} \sum_{states} \left[v_i \tanh(\sum_i J_{ij}v_i) \prod_j 2\cosh(\sum_i J_{ij}v_i) \right] \frac{1}{\mathscr{Z}} \\
=&D \sum_{states} \Bigg(v_i \tanh(\sum_i J_{ij}v_i) \prod_j 2\cosh(\sum_i J_{ij}v_i) \frac{\partial}{\partial J_{ab}} \frac{1}{\mathscr{Z}} \ldots \\
&+ \frac{1}{\mathscr{Z}} \frac{\partial}{\partial J_{ab}} \left[v_i \tanh(\sum_i J_{ij}v_i) \prod_j 2\cosh(\sum_i J_{ij}v_i) \right] \Bigg)
\end{aligned}$$

where
$$\frac{\partial}{\partial J_{ab}} \frac{1}{\mathscr{Z}} = -\langle v_a h_b \rangle_m * \mathscr{Z}^{-1}$$

$\Longrightarrow$
$$\begin{aligned}
&\sum_{states} v_i \tanh(\sum_i J_{ij}v_i) \prod_j 2\cosh(\sum_i J_{ij}v_i) \frac{\partial}{\partial J_{ab}} \frac{1}{\mathscr{Z}} \\
=&-\langle v_a h_b \rangle_m \sum_{states} v_i \tanh(\sum_i J_{ij}v_i) \prod_j 2\cosh(\sum_i J_{ij}v_i) * \mathscr{Z}^{-1} \\
=&\langle v_a h_b \rangle_m \langle v_i h_j \rangle_m
\end{aligned}$$

and
$$\begin{aligned}
&\frac{\partial}{\partial J_{ab}} \left[v_i \tanh(\sum_i J_{ij}v_i) \prod_j 2\cosh(\sum_i J_{ij}v_i) \right] \frac{1}{\mathscr{Z}} \\
&= D\Big[\langle v_i v_a \rangle_m - \langle v_i v_a h_j h_b \rangle_m \Big] \delta(b-j) + \langle v_i v_a h_j h_b \rangle_m
\end{aligned}$$

Putting all of this together in the order of (A.16), we get an expression for each of the elements of the Hessian matrix.

$$\mathscr{H}_{xy} = D\Big[\Big(\langle v_i v_a \rangle_d - \langle v_i v_a h_j h_b \rangle_d \Big) \delta(b-j) - \langle v_a h_b \rangle_m \langle v_i h_j \rangle_m \ldots$$

$$
\begin{aligned}
&\quad - \Big(\langle v_i v_a\rangle_m - \langle v_i v_a h_j h_b\rangle_m\Big)\delta(b-j) + \langle v_i v_a h_j h_b\rangle_m\Big] \\
\implies \mathscr{H}_{xy} &= D\Big[\Big(\langle v_i v_a\rangle_d - \langle v_i v_a h_j h_b\rangle_d - \langle v_i v_a\rangle_m + \langle v_i v_a h_j h_b\rangle_m\Big)\delta(b-j)... \\
&\quad + \langle v_a h_b\rangle_m \langle v_i h_j\rangle_m - \langle v_i v_a h_j h_b\rangle_m\Big]
\end{aligned}
$$

Written on another form, this means

$$
\mathscr{H}_{xy} = D * \begin{cases} \langle v_i v_a\rangle_d - \langle v_i v_a\rangle_m - \langle v_i v_a h_j h_b\rangle_d + \langle v_a h_b\rangle_m \langle v_i h_j\rangle_m & \text{if } j = b \\ \langle v_a h_b\rangle_m \langle v_i h_j\rangle_m - \langle v_i v_a h_j h_b\rangle_m & \text{if } j \neq b \end{cases}
$$

These expressions are possible to calculate analytically for any system in which we can calculate the partition function.

Semi-restricted Boltzmann Machine

When we add connections between the observed nodes into the network, the amount of parameters in the model increases by No^2. This is reflected in the dimensionality of the Hessian, as we now have to deal with an $No * Nh + No^2$-dimensional square matrix, and we will see that it takes the form of a symmetric block matrix of the following type

$$
\mathscr{H} = \left[\begin{array}{c|c} \frac{\partial^2}{\partial \mathbf{W}\partial \mathbf{W}}\mathscr{L} & \frac{\partial^2}{\partial \mathbf{W}\partial \mathbf{J}}\mathscr{L} \\ \hline \frac{\partial^2}{\partial \mathbf{J}\partial \mathbf{W}}\mathscr{L} & \frac{\partial^2}{\partial \mathbf{J}\partial \mathbf{J}}\mathscr{L} \end{array}\right] \tag{A.17}
$$

Now, the bottom right block of this matrix should remind you of Hessian of the RBM, with the exception that the form of the likelihood function itself is different. But since the probability function for the semi-restricted Boltzmann machine (sRBM) is separable in the weights, we have $\frac{\partial}{\partial \mathbf{J}}\mathscr{L}_{sRBM} = \frac{\partial}{\partial \mathbf{J}}\mathscr{L}_{RBM}$, and we see that the bottom right corner is, in fact, identical to the RBM Hessian. Since all the second partial derivatives of $\mathscr{L}$ are continuous we can use Schwartz's theorem, saying that the order of taking derivatives is irrelevant, so it will suffice to find the derive the left half of the Hessian to fill in the blanks.

Again, starting from the derivations of learning rules of the sRBM, we already know that $\frac{\partial}{\partial W_{ik}}\mathscr{L} = D\big(\langle v_i v_k\rangle_d - \langle v_i v_k\rangle_m\big)$. Now, differentiating this expression with respect to $\mathbf{W}$ will yield the top left corner, while differentiating with respect to $\mathbf{J}$ gives the bottom left quadrant:

$$
\begin{aligned}
\frac{\partial^2}{\partial J_{ab}\partial W_{ik}}\mathscr{L} &= \frac{\partial}{\partial J_{ab}} D\big(\langle v_i v_k\rangle_d - \langle v_i v_k\rangle_m\big) \\
&= -D\sum_s v_i^s v_k^s \frac{\partial}{\partial J_{ab}} P(s) \\
&= -D\sum_s v_i^s v_k^s \Big[v_a h_b P(s) - P(s)\sum_s v_a h_b P(s)\Big] \\
&= -D\langle v_i v_k v_a h_b\rangle_m - \langle v_i v_k\rangle_m \langle v_a h_b\rangle_m
\end{aligned}
$$

where it is implicit that the data correlation between visible units is independent of $\mathbf{J}$, and that $P(s) = \exp(\mathscr{E}(s)) * \mathscr{Z}^{-1}$, and $\mathscr{Z} = \sum_s \exp(\mathscr{E}(s))$. Using the same expressions

for probabilities, and the same sort of arguments, we get that

$$\begin{aligned}\frac{\partial^2}{\partial W_{ab}\partial W_{ik}}\mathscr{L} &= \frac{\partial}{\partial J_{ab}} D\big(\langle v_i v_k\rangle_d - \langle v_i v_k\rangle_m\big) \\ &= -D\sum_s v_i^s v_k^s \frac{\partial}{\partial W_{ab}} P(s) \\ &= -D\sum_s v_i^s v_k^s \Big[v_a v_b P(s) - P(s)\sum_s v_a h_b P(s)\Big] \\ &= -D\langle v_i v_k v_a v_b\rangle_m - \langle v_i v_k\rangle_m \langle v_a v_b\rangle_m\end{aligned}$$

Putting all these expressions into their rightful place in a matrix corresponding to (A.17), we get

$$\mathscr{H} = D\left[\begin{array}{c|c} \langle v_i v_j\rangle_m\langle v_a v_b\rangle_m - \langle v_i v_j v_a v_b\rangle_m & \langle v_i h_j\rangle_m\langle v_a h_b\rangle_m - \langle v_i h_j v_a h_b\rangle_m \\ \hline \langle h_i v_j\rangle_m\langle h_a v_b\rangle_m - \langle h_i v_j h_a v_b\rangle_m & \Big(\langle v_i v_a\rangle_d - \langle v_i v_a\rangle_m + \langle v_i v_a h_j h_b\rangle_m\Big)\delta(j-b) \\ & -\langle v_i v_a h_j h_b\rangle_m + \langle v_i h_j\rangle_m\langle v_a h_b\rangle_m \end{array}\right]$$

It is notable that only terms in the lower right quadrant, with elements calculated solely from the between-layer connections, depend on statistics from the data. In other words, the curvature of the likelyhodd function is completely described by the model parameters in a fully observed Boltzmann machine.

A.1.4 Correlations Without Connections

This section of the appendix is somewhat beside the scope of the thesis, but the author found its results to be interesting, and we therefore present some findings about connections between restricted and semi-restricted Boltzmann machines.

Even though restricted Boltzmann Machines do not have direct connections between the observed nodes, the correlations due to indirect connections through the hidden layer can be quantified. In the following, an expression for the value of these correlations will be derived.

The correlation between the I'th and the K'th observed node is defined as

$$\begin{aligned}\langle v_I v_K \rangle &= \sum_s v_I^s v_K^s P(s) \\ &= \sum_s v_I^s v_K^s \exp(\sum_{ij} v_i^s h_j^s J_{ij}) \mathscr{Z}^{-1}\end{aligned} \tag{A.18}$$

where we, as always, have that

$$\mathscr{Z} = \sum_s \exp(\sum_{ij} v_i^s h_j^s J_{ij})$$

When looking at a restricted architecture, the partition function can be rewritten in the following way

$$\begin{aligned}\mathscr{Z} &= \sum_s \exp\Big(\sum_{ij} v_i^s h_j^s J_{ij}\Big) \\ &= \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp\Big(\sum_{ij} v_i^s h_j^s J_{ij}\Big) \\ &= \sum_{\mathbf{v}} \sum_{h_1=\pm 1} \sum_{h_2=\pm 1} \cdots \sum_{h_N h=\pm 1} \exp\Big(\sum_{ij} v_i^s h_j^s J_{ij}\Big) \\ &= \sum_{\mathbf{v}} \sum_{h_1=\pm 1} \sum_{h_2=\pm 1} \cdots \sum_{h_{Nh}=\pm 1} \prod_j \exp\Big(h_j^s \sum_i v_i^s J_{ij}\Big)\end{aligned}$$

Since each term in this product only depends on one hidden node value, we can separate the expression, placing each factor into its respective sum.

$$\begin{aligned}\mathscr{Z} &= \sum_{\mathbf{v}} \sum_{h_1=\pm 1} \exp\Big(h_1^s \sum_i v_i^s J_{i1}\Big) \cdots \sum_{h_{Nh}=\pm 1} \exp\Big(h_{Nh}^s \sum_i v_i^s J_{iNh}\Big) \\ &= \sum_{\mathbf{v}} \Big[\exp\Big(\sum_i v_i^s J_{i1}\Big) + \exp\Big(-\sum_i v_i^s J_{i1}\Big)\Big] \cdots \Big[\exp\Big(\sum_i v_i^s J_{iNh}\Big) + \exp\Big(-\sum_i v_i^s J_{iNh}\Big)\Big] \\ &= \sum_{\mathbf{v}} \prod_j 2\cosh\Big(\sum_i v_i^s J_{ij}\Big)\end{aligned} \tag{A.19}$$

This result is general for any RBM, and is generally the simplest form of the partition function. For very small networks, though, this can be rewritten to a form which we can use to do further calculations. For example, in the special case with two observed and one hidden node, there is only four possible configurations of the observed node activity, so we can write out the expression explicitly, leading to the following form for the partition function

$$\begin{aligned}\mathscr{Z} &= 2\cosh\big(J_{11}+J_{21}\big)+2\cosh\big(-J_{11}+J_{21}\big)+2\cosh\big(J_{11}-J_{21}\big)+2\cosh\big(-J_{11}-J_{21}\big)\\ &= 4\Big[\cosh\big(J_{11}+J_{21}\big)+\cosh\big(-J_{11}+J_{21}\big)\Big]\end{aligned}$$

Here we have used the symmetric property cosine hyperbolic function, $\cosh(A)=\cosh(-A)$. Now, using $\cosh(A\pm B)=\cosh(A)\cosh(B)\pm\sinh(A)\sinh(B)$, we can rewrite the expression for the partition further.

$$\begin{aligned}\mathscr{Z} =&4\Big[\cosh(J_{11})\cosh(J_{21})+\sinh(J_{11})\sinh(J_{21})+\cosh(-J_{11})\cosh(J_{21})+\sinh(-J_{11})\sinh(J_{21})\Big]\\ =&4\Big[\cosh(J_{11})\cosh(J_{21})+\sinh(J_{11})\sinh(J_{21})+\cosh(J_{11})\cosh(J_{21})-\sinh(J_{11})\sinh(J_{21})\Big]\\ =&8\Big[\cosh(J_{11})\cosh(J_{21})\Big] \qquad\qquad (A.20)\end{aligned}$$

where we again used the symmetry of cosine hyperbolic together with the corresponding anti-symmetric property of the sine hyperbolic function, $\sinh(A)=-\sinh(-A)$.

Going back to the original problem of finding an expression for the correlation between the observed nodes, we start with rearranging (A.18) for correlations between the I'th and K'th observed nodes, using similar arguments as in the rewriting of $\mathscr{Z}$

$$\begin{aligned}\mathscr{Z}\langle v_I v_K\rangle &= \sum_s v_I^s v_K^s \exp(\sum_{ij} v_i^s h_j^s J_{ij})\\ &= \sum_{\mathbf{v}} v_I^s v_K^s \sum_{h_1=\pm1}\sum_{h_2=\pm1}\cdots\sum_{h_{Nh}=\pm1}\prod_j \exp\Big(h_j^s\sum_i v_i^s J_{ij}\Big)\\ &= \sum_{\mathbf{v}} v_I^s v_K^s \Big[\sum_{h_1=\pm1}\exp\Big(h_1^s\sum_i v_i^s J_{i1}\Big)\\ &\qquad\qquad \sum_{h_2=\pm1}\exp\Big(h_2^s\sum_i v_i^s J_{i2}\Big)\cdots\sum_{h_N h=\pm1}\exp\Big(h_{Nh}^s\sum_i v_i^s J_{iNh}\Big)\Big]\\ &= \sum_{\mathbf{v}} v_I^s v_K^s \prod_j 2\cosh\Big(\sum_i v_i^s J_{ij}\Big)\end{aligned}$$

For the simplest case of two observed and one hidden node, using the same arrangement of observed states as in the derivation of the partition, this can be written out as

$$\mathscr{Z}\langle v_1 v_2\rangle = 2\cosh\big(J_{11}+J_{21}\big)-2\cosh\big(-J_{11}+J_{21}\big)$$

$$
\begin{aligned}
&\quad - 2\cosh\big(J_{11} - J_{21}\big) + 2\cosh\big(-J_{11} - J_{21}\big) \\
&= 4\Big[\cosh\big(J_{11} + J_{21}\big) - \cosh\big(-J_{11} + J_{21}\big)\Big] \\
&= 4\Big[\cosh(J_{11})\cosh(J_{21}) + \sinh(J_{11})\sinh(J_{21}) \\
&\qquad - \cosh(-J_{11})\cosh(J_{21}) - \sinh(-J_{11})\sinh(J_{21})\Big] \\
&= 4\Big[\cosh(J_{11})\cosh(J_{21}) + \sinh(J_{11})\sinh(J_{21}) \\
&\qquad - \cosh(J_{11})\cosh(J_{21}) + \sinh(J_{11})\sinh(J_{21})\Big] \\
&= 8\Big[\sinh(J_{11})\sinh(J_{21})\Big]
\end{aligned}
$$

Combining this expression with (A.20), we get an expression for the correlation between v_1 and v_2

$$
\langle v_1 v_2 \rangle = \frac{8\Big[\sinh(J_{11})\sinh(J_{21})\Big]}{8\Big[\cosh(J_{11})\cosh(J_{21})\Big]}
$$

$$
\implies \langle v_1 v_2 \rangle = \tanh(J_{11})\tanh(J_{21}) \tag{A.21}
$$

Additionally, it can be shown that for any RBM with a single hidden node, there is a general expression for the correlation between the I'th and K'th observed node, $\langle v_I v_K \rangle = \tanh(J_{I1})\tanh(J_{K1})$. Unfortunately this result does not seem to easily generalize to networks with more hiddennodes, due to the symmetric, and antisymmetric, properties of $\sinh(\dots)$ and $\cosh(\dots)$ are harder to take advantage of for more than two arguments. This is not to say it is impossible to find analytical expressions (which it is not, as we have found formulas for several examples), but finding a general formula for the observed correlation given the number of hidden nodes is non-trivial.

Correlation with Connections

It seems plausible, for the simple restricted architecture, that there exists some semi-restricted architecture that generates data with equivalent correlations between the observed nodes. This idea can be argumented for by thinking of an RBM as atype of sRBM in which the observed-observed connections, $\mathbf{W}$, just happen to be zero. It is not beyond our imagination to picture a change in these $\mathbf{W}$'s being followed by a corresponding change in the $\mathbf{J}$'s which tweaks the observed correlations back to what they originally were with no observed-observed connections.

In the following we will derive an expression for the observed correlations in a two observed one hidden (2x1) semi-restricted Boltzmann machine (sRBM). We will then compare it with (A.21) and find an analytical solution showing that any observed-observed connection, W, can generate equivalent correlations given a suitable set of between layer connections, J.

Now, we start by finding an expression for the observed node correlations in a sRBM.

$$\begin{aligned}\langle v_I v_K \rangle &= \sum_s v_I^s v_K^s P(s) \\ &= \sum_s v_I^s v_K^s \exp(\sum_{ik} v_i^s v_k^s W_{ik} + \sum_{ij} v_i^s h_j^s J_{ij}) \mathscr{Z}^{-1} \end{aligned} \tag{A.22}$$

where the partition function now is

$$\mathscr{Z} = \sum_s \exp(\sum_{ik} v_i^s v_k^s W_{ik} + \sum_{ij} v_i^s h_j^s J_{ij})$$

Using much the same arguments as in the derivation of the correlations in the RBM, we get

$$\langle v_I v_K \rangle = \frac{\sum_s v_I^s v_K^s \exp(\sum_{ik} v_i^s v_k^s W_{ik} + \sum_{ij} v_i^s h_j^s J_{ij})}{\sum_s \exp(\sum_{ik} v_i^s v_k^s W_{ik} + \sum_{ij} v_i^s h_j^s J_{ij})}$$

$$= \frac{\sum_{\mathbf{v}} v_I^s v_K^s \exp(\sum_{ik} v_i^s v_k^s W_{ik}) \sum_{\mathbf{h}} \exp(\sum_{ij} v_i^s h_j^s J_{ij})}{\sum_{\mathbf{v}} \exp(\sum_{ik} v_i^s v_k^s W_{ik}) \sum_{\mathbf{h}} \exp(\sum_{ij} v_i^s h_j^s J_{ij})}$$

$$= \frac{\sum_{\mathbf{v}} v_I^s v_K^s \exp(\sum_{ik} v_i^s v_k^s W_{ik}) \prod_j 2\cosh(\sum_i v_i^s J_{ij})}{\sum_{\mathbf{v}} \exp(\sum_{ik} v_i^s v_k^s W_{ik}) \prod_j 2\cosh(\sum_i v_i^s J_{ij})}$$

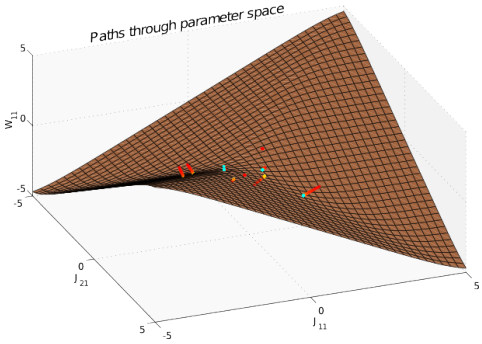
Again, since there are only four possible observed states, and the v_i can only take the values ± 1, we can write out the full expression explicitly.

$$\langle v_1 v_2 \rangle = \frac{2 * \exp(W_{12}) \cosh(J_{11} + J_{21}) - 2 * \exp(-W_{12}) \cosh(-J_{11} + J_{21})}{2 * \exp(W_{12}) \cosh(J_{11} + J_{21}) + 2 * \exp(-W_{12}) \cosh(-J_{11} + J_{21})}$$

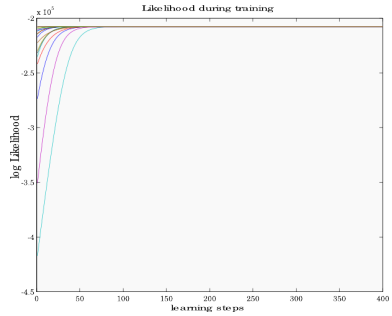
where we have, once again, used the symmetry of the hyperbolic cosine. Using the relation $\cosh(A \pm B) = \cosh(A)\cosh(B) \pm \sinh(A)\sinh(B)$ and the antisymmetry of sine hyperbolic, we can rewrite this equation quite drastically, to

$$\langle v_1 v_2 \rangle = \frac{4\sinh(W_{12})\cosh(J_{11})\cosh(J_{12}) + 4\cosh(W_{12})\sinh(J_{11})\sinh(J_{12})}{4\cosh(W_{12})\cosh(J_{11})\cosh(J_{12}) + 4\sinh(J_{12})\sinh(J_{11})\sinh(J_{12})}$$

$$= \frac{\cosh(J_{11})\cosh(J_{12})\cosh(W_{12})\big[\tanh(W_{12}) + \tanh(J_{11})\tanh(J_{12})\big]}{\cosh(J_{11})\cosh(J_{12})\cosh(W_{12})\big[1 + \tanh(W_{12})\tanh(J_{11})\tanh(J_{12})\big]}$$



(A) Paths of parameters during learning, and the analytical surface of predicted sets of parameters recreating the data statistics.



(B) Likelihood evolution for the learning.

FIGURE A.1: *Here we try to visualize the continuum of solutions in a semi restricted Boltzmann machine predicted by equation A.23. The plane shows the possible solutions, the stars show the paths of several learning simulations based on data generated from a restricted Boltzmann machine.*

$$= \frac{\tanh(W_{12}) + \tanh(J_{11})\tanh(J_{12})}{1 + \tanh(W_{12})\tanh(J_{11})\tanh(J_{12})}$$

Equating this with (A.18) could give us an expression for the W's and J's needed to regenerate the correlations obtained from an RBM with J^*_{11} and J^*_{21}.

$$\tanh(J^*_{11})\tanh(J^*_{21}) = \frac{\tanh(W_{12}) + \tanh(J_{11})\tanh(J_{12})}{1 + \tanh(W_{12})\tanh(J_{11})\tanh(J_{12})}$$

$$\tanh(J^*_{11})\tanh(J^*_{21})\big(1 + \tanh(W_{12})\tanh(J_{11})\tanh(J_{12})\big) = \tanh(W_{12}) + \tanh(J_{11})\tanh(J_{12})$$

$$\tanh(W_{12}) = \frac{\tanh(J_{11})\tanh(J_{12}) - \tanh(J^*_{11})\tanh(J^*_{21})}{\tanh(J_{11})\tanh(J_{12})\tanh(J^*_{11})\tanh(J^*_{21}) - 1} \tag{A.23}$$

This equation has solutions as long as the absolute value of the right hand side is smaller than one. In other words, we check which values for J's give a solution, given a pair of J^*.

$$\begin{aligned}
\left|\frac{\tanh(J_{11})\tanh(J_{12}) - \tanh(J^*_{11})\tanh(J^*_{21})}{\tanh(J_{11})\tanh(J_{12})\tanh(J^*_{11})\tanh(J^*_{21}) - 1}\right| &\leq 1 \\
\frac{|t(J_{11})t(J_{12}) - t(J^*_{11})t(J^*_{21})|}{|t(J_{11})t(J_{12})t(J^*_{11})t(J^*_{21}) - 1|} &\leq 1 \\
|t(J_{11})t(J_{12}) - t(J^*_{11})t(J^*_{21})| &\leq |t(J_{11})t(J_{12})t(J^*_{11})t(J^*_{21}) - 1| \\
(t(J_{11})t(J_{12}) - t(J^*_{11})t(J^*_{21}))^2 &\leq (t(J_{11})t(J_{12})t(J^*_{11})t(J^*_{21}) - 1)^2 \\
(t(J_{11})t(J_{12}))^2 + (t(J^*_{11})t(J^*_{21}))^2 &\leq (t(J_{11})t(J_{12})t(J^*_{11})t(J^*_{21}))^2 + 1 \\
(t(J^*_{11})t(J^*_{21}))^2 - 1 &\leq (t(J_{11})t(J_{12})t(J^*_{11})t(J^*_{21}))^2 - (t(J_{11})t(J_{12}))^2
\end{aligned}$$

$$\begin{aligned}(t(J_{11}^*)t(J_{21}^*))^2 - 1 &\leq (t(J_{11})t(J_{12}))^2\left[(t(J_{11}^*)t(J_{21}^*))^2 - 1\right] \\ 1 &\geq \tanh(J_{11})^2 \tanh(J_{12})^2\end{aligned}$$

where we have used the name t in place of tanh to get past spacial limitations. The result tells us that (A.23) is valid for any set of J's, since since $\tanh(J)^2 \leq 1$ for any J, and that one can predict the value of W that gives equivalent correlations as an RBM with J^*'s, given any pair of J. Additionally, this leads us to the conclusion that the 2x1 case of the semi restricted Boltzmann machine is not a strictly convex problem, since the statistics of any dataset can be generated from from a RBM with the correct set of J^*'s, can be recreated by an sRBM with an arbitrary set of $J*$.

This result is not very surprising, however, as the 2-by-1 RBM is not convex either, and adding another degree of freedom does not generally constrain systems more. Also, generalizing this result to larger systems is not trivial, but we did find closed form expressions of the observed correlations for 3-by-1, 3-by-2 and 4-by-1 systems, which gave similar, but more complex, formulas only consisting of hyperbolic tangents of single parameters. Such expressions may lead to closed form expressions connecting RBM and sRBM parameters, and could provide strong insight into the convexity of the sRBM.

Appendix B

Sample Code

Here we include example code containing some of the functions written and used in the project. It is supposed to be a more or less minimal functioning code, containing data generation from a restricted Boltzmann machine network, learning of parameters from different initial conditions, and visualization in form of plots.

Template Code for Boltzmann Learning

```
% Number of nodes in the network
No = 8;         Nh = 2;         N = No+ Nh;

% Generative parameters of the model
W = zeros(No);                  % Observed-Observed  connections
W = (W+W'-2*diag(diag(W)))/2;   % Making W symetric with 0 diagonals
J = randn(No,Nh)/sqrt(No*Nh);   % Observed-Hidden connections
K = zeros(Nh);                  % Hidden-Hidden connections
K = (K+K'-2*diag(diag(K)))/2;   % Making K symmetric with 0 diagonals
f = zeros(No,1);                % Observed node external fields

D = 100000;   % Length of data
L = 10000;    % Maximum learning steps

% Genrating matrix of all possible observed (Os) and hidden (Hs) states
[Os,Hs] = StateGeneration(No,Nh);

% Generating data set from the specified network
data     = MetropolisDataGeneration(No,Nh,W,J,K,f,D,1);

% Inferring the maximum likelihood parameters from the data
% From correct
Jinf     = BMinference('RBM',data,J,W,K,L);
% and random initial conditions
JinfRan  = BMinference('RBM',data,rand(No,Nh)/sqrt(No*Nh),W,K,L);

% Visualizing the learning evolution
SubplotEvolutionRBM(J,data,Jinf,JinfRan);
```

State Generation

A function for generating truth tables; a matrix containing all possible states of activation in the network. It returns separate matrices for for observed (Os), hidden (Hs) and all states, given that the number of nodes is not to big to kill the memory.

```
function [ObsS,HidS,AllS] = StateGeneration(No,Nh)

N = No + Nh;

if N<20
    AllS  = 2*bitget(repmat((0:2^N-1)',1,N),repmat(-(-N:-1),2^N,1)) - 1;
end

ObsS = 2*bitget(repmat((0:2^No-1)',1,No),repmat(-(-No:-1),2^No,1)) - 1;
HidS = 2*bitget(repmat((0:2^Nh-1)',1,Nh),repmat(-(-Nh:-1),2^Nh,1)) - 1;
```

Metropolis Data Generation

The function for generating data uses a Metropolis algorithm, with a burn-in period to control for bias in the initial conditions. If not explicitly stated, most input arguments have defined standard values, but we advice defining everything to control the simulation.

```
function [vdata hdata data Sfinal] = ...
    MetropolisDataGeneration(No,Nh,W,J,K,f,steps,flips,Sinit)

N = No + Nh;

% Controlling for options in input variables
Train = 1;
if nargin==6
    steps  = 20000*No*Nh;
    flips  = ceil(N/10);
    Sinit  = 2*round(rand(N,1))-1;
elseif nargin==7
    flips  = ceil(N/10);
    Sinit  = 2*round(rand(N,1))-1;
elseif nargin == 8
    Sinit  = 2*round(rand(N,1))-1;
elseif nargin == 9
    Train =0;
elseif nargin == 3
    J = W;
    W = zeros(No);
    K = zeros(Nh);
    f = zeros(No,1);
    steps  = 50000*No*Nh;
    flips  = ceil(N/10);
    Sinit  = 2*round(rand(N,1))-1;
end

if steps<100*round(sqrt(Nh))
    train = steps;
else
    train = 100*round(sqrt(Nh));
end


% Data generation
data   = zeros(N,steps);
S      = Sinit;
flip   = ceil(N*rand(flips,steps));
accept = log(rand(1,steps));
oldE   = -(0.5*S(1:No)'*W*S(1:No) + S(1:No)'*J*S(No+1:N)...
    + 0.5*S(No+1:N)'*K*S(No+1:N)+ S(1:No)'*f);

% Generating data to get to equilibrium (burn-in period)
if Train
```

```
    for i=1:train

        % flip one spin
        for j=1:flips
            S(flip(j,i)) = S(flip(j,i))*(-1);
        end

        data(:,i)  = S;

        % calculate and compare energy to accept/reject flip
        newE = -(0.5*(S(1:No)'*W*S(1:No)+AddSelfW) + ...
            S(1:No)'*J*S(No+1:N) + 0.5*S(No+1:N)'*K*S(No+1:N)...
            + S(1:No)'*f);
        changeE = oldE - newE;

        if changeE<accept(i)
            for j=1:flips
                % rejecting
                S(flip(j,i)) = S(flip(j,i))*(-1);
            end
            data(:,i)  = S;

        else
            % accepting
            oldE = newE;
        end

    end
end
% Assuming equilibrium, generating data
for i=1:steps

    % flip one
    for j=1:flips
        S(flip(j,i)) = S(flip(j,i))*(-1);
    end

    data(:,i)  = S;

    % calculate and compare energy to accept/reject flip
    newE = -(0.5*(S(1:No)'*W*S(1:No)+AddSelfW) ...
        + S(1:No)'*J*S(No+1:N) + S(No+1:N)'*K*S(No+1:N) ...
        + S(1:No)'*f);
    changeE = oldE - newE;

    if changeE<accept(i)
        for j=1:flips
            % rejecting
            S(flip(j,i)) = S(flip(j,i))*(-1);
        end
        data(:,i)  = S;
    else
        % accepting
        oldE = newE;
    end

end
Sfinal = S;
hdata  = data(No+1:N,:);
vdata  = data(1:No,:);
```

Boltzmann Learning Algorithm

The part of our code specific for the restricted Boltzmann machine learning is presented here, but the functionality to do learning for different architectures or clamped learning has been removed here to save space.

```
unction [Jinf Winf Kinf] = BMinference(type,data,Jinit,Winit,Kinit,maxSteps)

[No Nh] = size(Jinit);
etaJ     = 0.5;
etaW     = 0.5;
etaK     = 0.5;
L       = size(data,2);

 % Generating weight matrices
Jinf  = zeros(No,Nh,maxSteps); % Weight guess
Jinf(:,:,1) = Jinit;

Winf        = zeros(No,No,maxSteps);
Winf(:,:,1) = Winit;

Kinf        = zeros(Nh,Nh,maxSteps);
Kinf(:,:,1) = Kinit;

[ObsStates,HidStates] = StateGeneration(No,Nh);

i = 0;
j = 0;
lim = 10^(-8);

% Updating loop for performing gradient ascent on the likelihood
while i<maxSteps && j<2
    i = i+1;

     % Calculate correlations from data
    meanHdata = tanh(Jinf(:,:,i)'*data);
    corrHdata = data*meanHdata'/L;

     % Calculate model correlations
    meanHmodel  = tanh(ObsStates*Jinf(:,:,i));
    eModel      = (ObsStates*Jinf(:,:,i))*HidStates';
    Z           = sum(sum(exp(eModel)));
    pModel      = sum(exp(eModel),2)/Z;
    corrOHmodel = meanHmodel'*(repmat(pModel,1,No).*ObsStates);

     % Update weights
    Jinf(:,:,i+1) = Jinf(:,:,i) + etaJ*(corrHdata-corrOHmodel');

     % Checking tresholds for stopping learning
    if mod(i,50)==0

        testJ  = sum(sum((Jinf(:,:,i)-Jinf(:,:,i-13)).^2));
        testJ2 = sum(sum((Jinf(:,:,i)-Jinf(:,:,i-13)).^2));

        if testJ<lim && testJ>0 && testJ2>0
            j = j+1;
            etaJ = etaJ/2;
        elseif testJ<0 || testJ2<0
            etaJ = etaJ/2;
        end
    end
end
```

Visualizing the Learning

One of the functions used for visualizing the parameter behavior during learning. Sending in the real RBM parameters, the data matrix and any number of matrices containing the parameters during learning, will result in a matrix of subplots showing certain variables of interest.

```
 function SubplotEvolutionRBM(J,data,varargin)

r = 0.97;
b = 0.97;
```

```
g = 0.97;

[No Nh] = size(J);
[Os Hs] = StateGeneration(No,Nh);

B = max(size(varargin)) ;
col = copper(B+1);
Jr = vararginB(:,:,end);

figure;
hold on;
subplot(2,2,3);hold on;scatter(J(:),Jr(:),25,col(1,:),'o','filled');

for i = 1:B
    Jif = varargini;
    s(i) = max(size(Jif));
end

K = max(s);

% Making plots for every
for i = 1:B
    Jinf = varargini;
    A = max(size(Jinf));

    RMS = zeros(1,A);
    Jperm = zeros(No,Nh,A);
    Likelihood = zeros(1,A);

    [~,Jperm] = PermutedRMS(J,Jinf);

    for j = 1:A
        RMS(j)        = sqrt(sum(sum((Jperm(:,:,j)-J).^2))/(No*Nh));
        Likelihood(j) = LikelihoodCalculation(No,Nh,W,Jperm(:,:,j),Kw,data,Os,Hs);
    end
    if A<K
        for k=A:K
            RMS(k) = RMS(A);
            Jperm(:,:,k) = squeeze(Jperm(:,:,A));
            Likelihood(k) = Likelihood(A);
        end
    end

    J1 = Jperm(:,:,1);
    Jf = Jperm(:,:,end);

    x = linspace(1,length(RMS),length(RMS));

    size(squeeze(Jinf(2,1,:)))
    size(squeeze(Jinf(1,1,:)))
    subplot(2,2,1);hold on;plot(x,Likelihood,'Color',col(i+1,:),'Linewidth',2);
    subplot(2,2,2);hold on;plot(x,RMS,'Color',col(i+1,:),'Linewidth',2);
    subplot(2,2,3);hold on;scatter(J(:),Jf(:),25,col(i+1,:),'*'); hold on;
    subplot(2,2,4);hold on;plot(squeeze(Jperm(1,1,:)),squeeze(Jperm(2,1,:)),':*','MarkerSize',3,'Color',col(i+1,:))
end

% Finalizing plots with labels, titles and background color
subplot(2,2,3);plot([-1 1],[-1 1],'k:');
subplot(2,2,1);xlabel('Learning iterations');ylabel('log Likelihood');
title('Likelihood evolution');set(gca,'Color',[r b g]);
subplot(2,2,2);xlabel('Learning iterations');ylabel('Root Mean Square Error');
title('RMS evolution');set(gca,'Color',[r b g]);
subplot(2,2,3);xlabel('Inferred weights');ylabel('Real weights');
title('Scatter of inferred Parameters');set(gca,'Color',[r b g]);
subplot(2,2,4);hold on;scatter(J(1,1),J(2,1),40,'g','*');
xlabel('No1 - Nh1');ylabel('No2 - Nh1');title('Path of Parameters');
set(gca,'Color',[r b g]);
subplot(2,2,1);xlabel('Learning iterations');ylabel('log Likelihood');title('Likelihood evolution');
suptitle('Evolution of measures/parameters during inference');
```

These just show a small sample of the codes written. There are several more functions and variations written to investigate the properties visited in the main text. And in case full functional code is of interest, please contact the author at *bjorneju@gmail.com*, for matlab files.

Appendix C

Additional Plots and figures

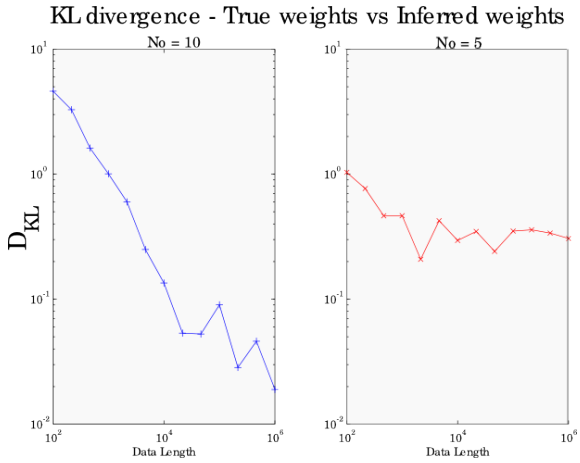


Figure C.1: *KL-divergence between the distribution used to draw the true weights, the distribution of weights inferred by the RBM algorithm ($\mathcal{O}(100)$ runs). The left (blue) figure shows the relation for the same network of 8 observed and 3 hidden nodes presented in figure 3.1, while the right (red) shows the same for a 5x1 network. In both cases we use the standard $g = 1$ for scaling weights.*

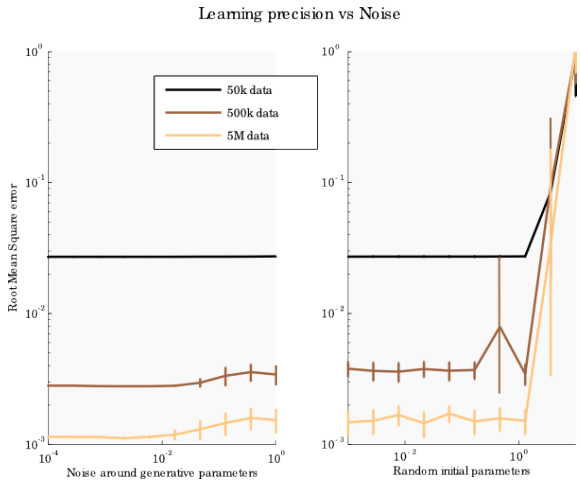


Figure C.2: *On the left: the root mean square error of inference as a function of the amount of noise in the initial conditions. On the right: root mean square error as a function of the standard deviation of random initial conditions. The standard deviation of the random variable used to add noise is varied along the x-axis, and the values marking the axis are the different standard deviations of the noisy σ used.*

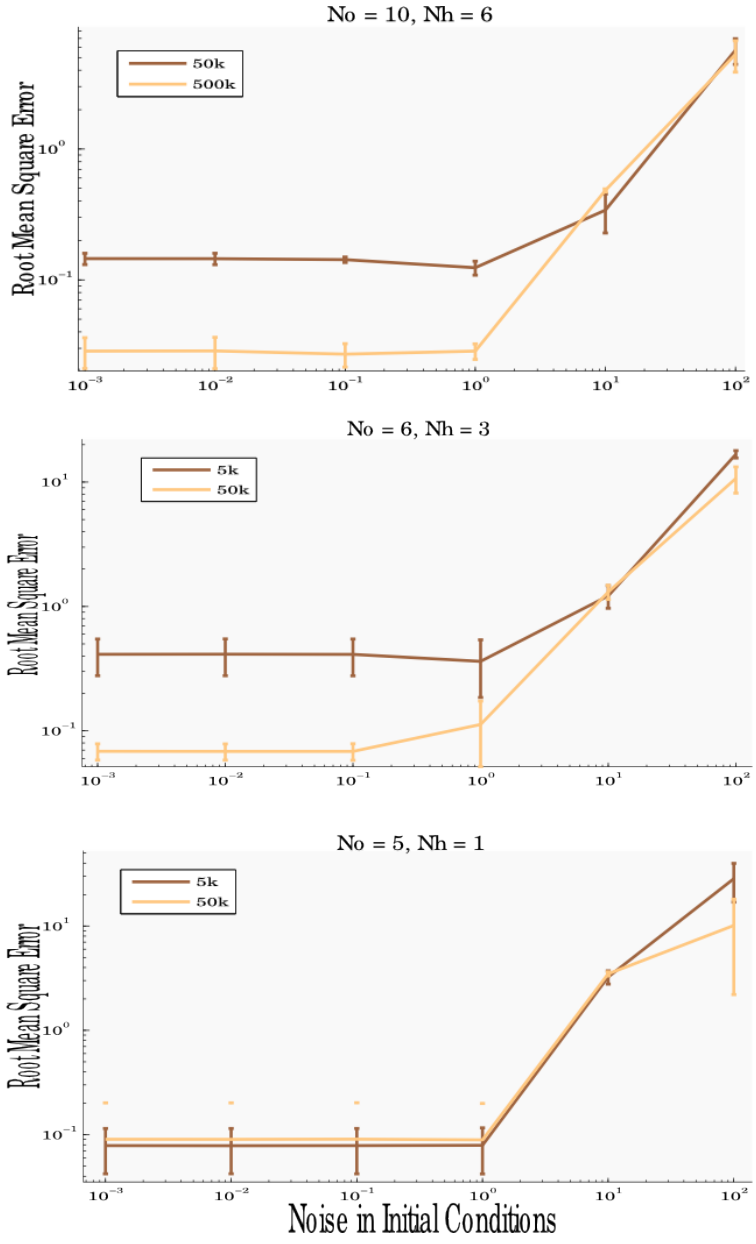


Figure C.3: *Expanding on the results from figure 3.2 we show the sensitivity to initial conditions for three different sized networks. In every case the $g = 1$ and the lines show averages of 5 simulations.*

(A) $No = 6$ and $Nh = 3$

(B) $No = 8$ and $Nh = 1$

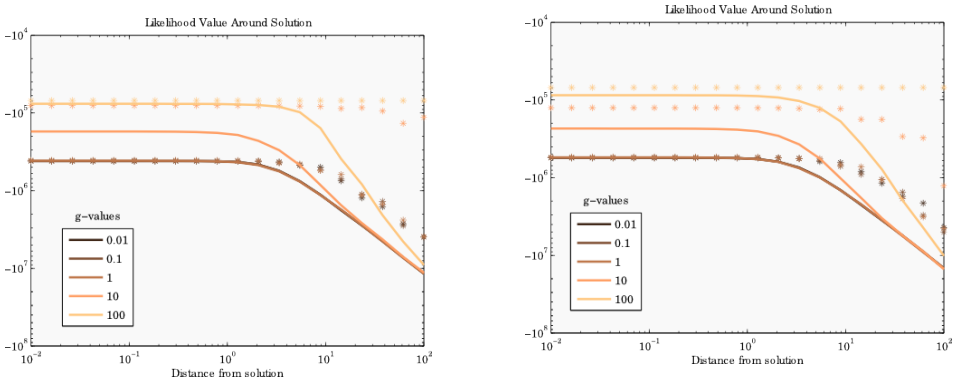


Figure C.4: *Two supplementary figures showing the effect on the average value of the likelihood surface as a function of the distance from the inferred point in parameter space. These figures agree well with the main observations inf figure 3.4. The peak sizes are similar, smal g-values are virtually indistinguishable and large g-values have near-max likelihood values far from the inferred solution.*

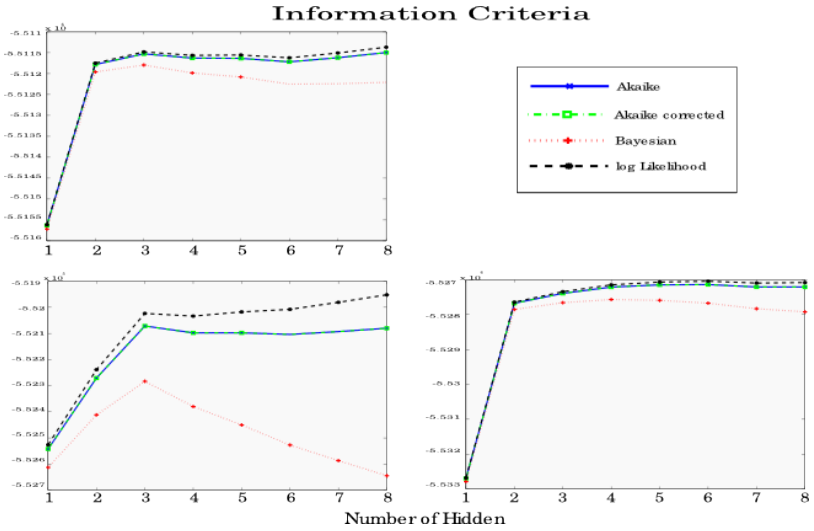


Figure C.5: *Plotting measures to find the models goodness-of-fit to the data observed. Comparing two versions of Akaike, and the Bayesian, information criterion with the a normalized log likelihood against the number of hidden nodes in the reconstruction. There were two, four and eight nodes in the generative models for the top-left, bottom-left and bottom-right figures respectively.*

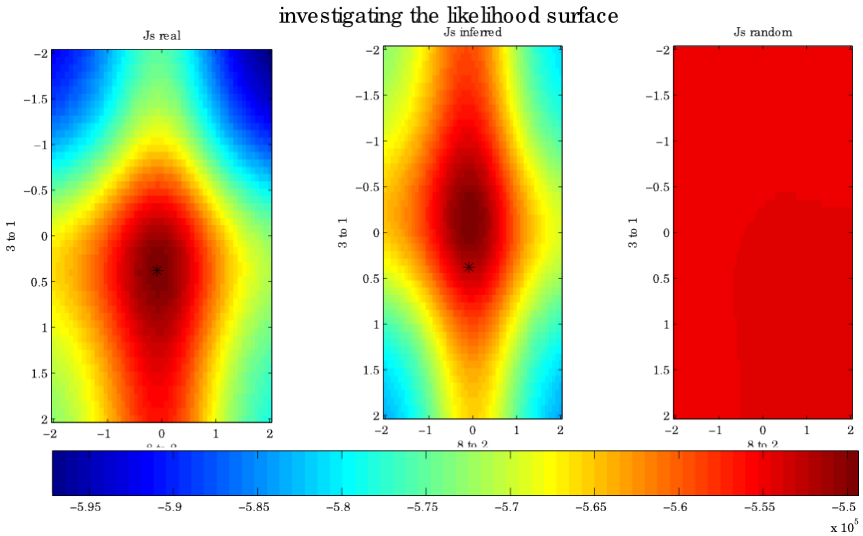


Figure C.6: *Surveying the Likelihood surface of an RBM with 8 observed and 3 hidden nodes in three different areas of the parameter space. In all figures we sweep over a range of values for the weights between observed number 1 and hidden node 3, and observed number 5 and hidden node 3. In the left figure, the remaining parameters are held at their correct values - the values which they had during data generation. In the central figure, the remaining weights are held at the inferred values. In the right figures, all parameters were given a random value drawn from the same distribution as the real parameters. The black stars mark the true and inferred values, of J_{53} and J_{13}, in the left and center plot respectively. The value in the bottom left corner of each plot denotes the maximum likelihood value encountered during each local sweep.*

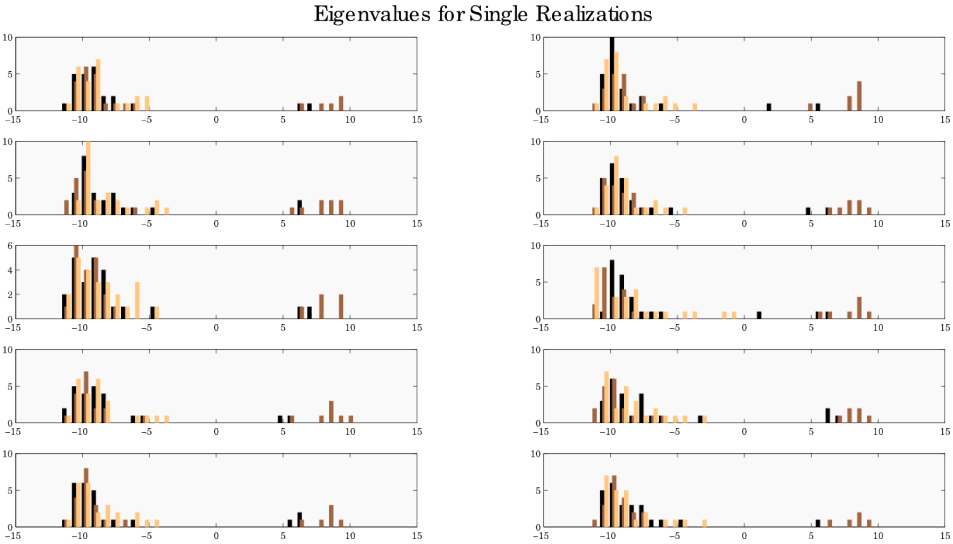
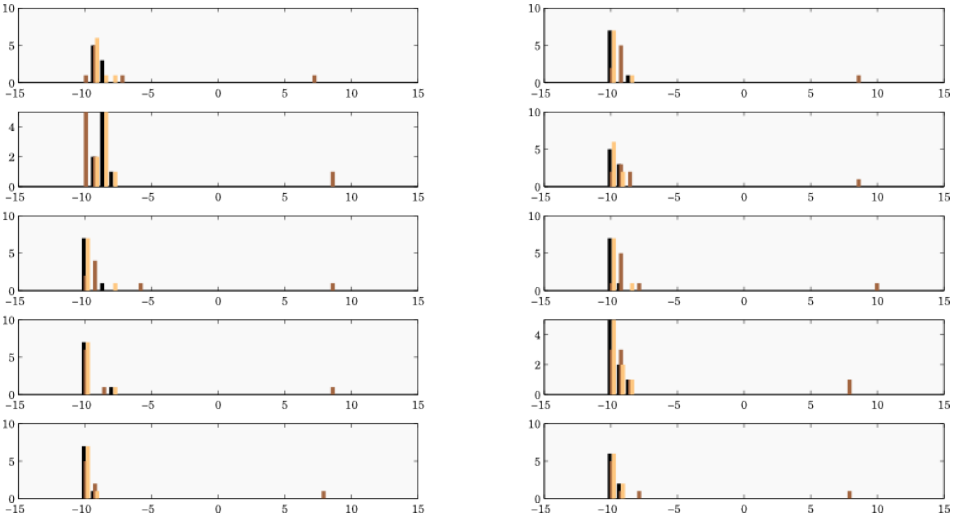


Figure C.7: *Showing histograms of the Hessian eigenvalues from ten different realizations of 8-by-3 restricted Boltzmann machines. It shows that the within trial variation in the eigenvalue distributions, for all points (true, random and inferred) is small, and very similar to the overall distributions depicted in figure 3.8.*
Below we show a similar plot from a 5-by-1 network, in which the distribution of eigenvalues still seems stable. In the inferred all eigenvalues are positive, but that is likely due to the fact that the parameters are inferred with a smaller error in simpler systems, and is therefor closer to the ground truth parameters.



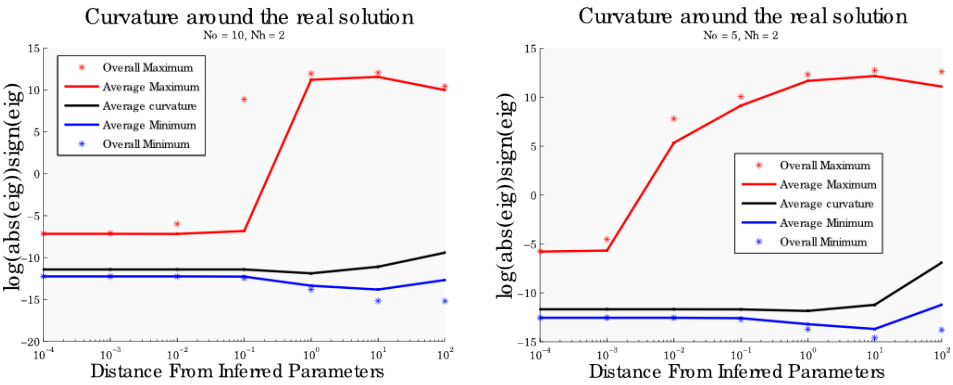


Figure C.8: *Following the procedure explained in the caption of 3.9, this figure shows some descriptive statistics of the Hessian eigenvalues as a function of the distance from the inferred peak parameters. Here we have used 10-by-2 and 5-by-2 RBM networks in the left and right figures respectively.*

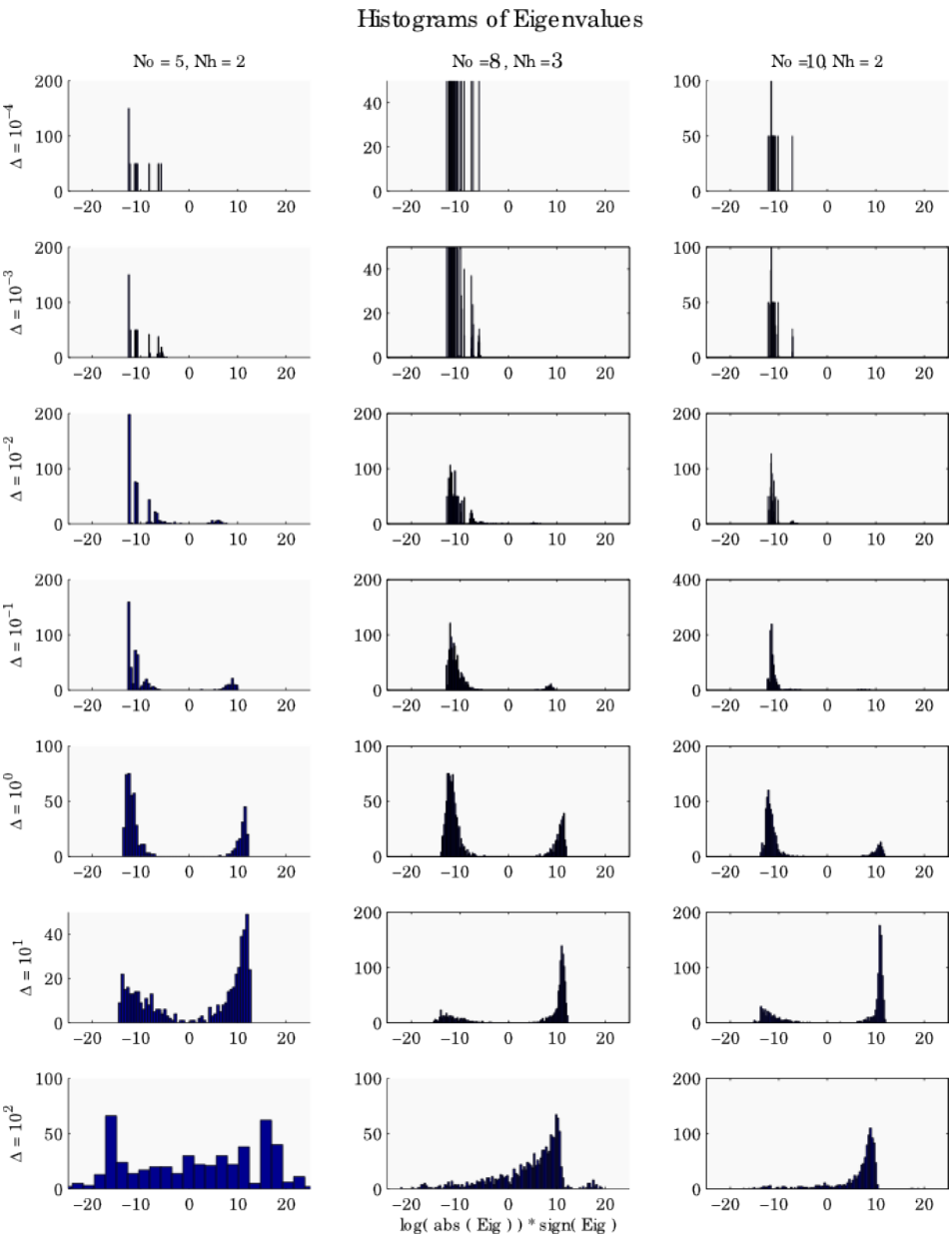


Figure C.9: *Histograms of all the eigenvalues used to generate the plots, both in the main text (figure 3.9) and here in the appendix (figure C.8). Each column contains the eigenvalues from one network, while the rows contain the values for one distance, each. The bottom left histogram looks wrong, but it is due to it having more than 50% of its eigenvalues valued between -1 and 1, making the logarithmic transformation unambiguous. The unambiguity is also visible in the bottom center figure, but is less apparent. This observation makes it clear that there are near zero eigenvalues for large distances away from the peak. However, since we are no longer (necessarily) in a stationary point, it is hard to draw more precise conclusions from this.*

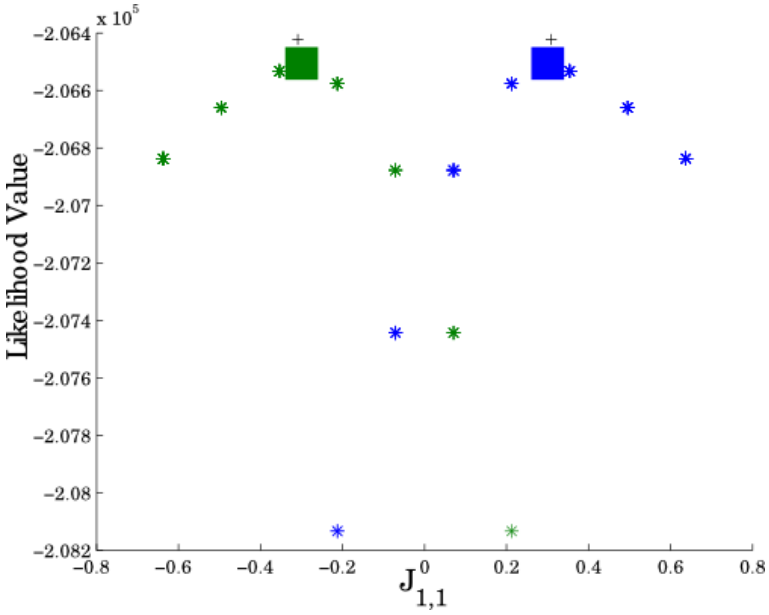


Figure C.10: *Following the procedure explained in the caption of 3.12, this figure shows the local convexity of a 6-by-1 RBM, with typical weight distributions and 50k samples in the data set.*