Deterministic Boltzmann Machines: Learning Instabilities and Hardware Implications

by

Roland Schneider

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

© 1993

National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-81812-3

Canada

Name ROLAND SCHNEIDER

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ENGINEERING — ELECTRONICS AND ELECTRICAL | 0 5 4 4

SUBJECT TERM | SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

DETERMINISTIC BOLTZMANN MACHINES: LEARNING INSTABILITIES AND HARDWARE IMPLICATIONS

BY

ROLAND SCHNEIDER

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

© 1993

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.
The author reserves other publications rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's permission.

Abstract

The deterministic Boltzmann machine (DBM) neural network architecture was originally derived from the stochastic Boltzmann machine (BM) by substituting the expected values of unit activations for the stochastic activations of the BM. Our simulations show that the DBM, unlike the BM, exhibits unstable behavior such as oscillation during learning and hypersensitivity to small perturbations of the weights or other network parameters. While other researchers have encountered similar oscillatory behavior, it has never been satisfactorily analyzed.

It is shown that this unstable behavior is the result of over-parameterization (excessive freedom in the weights), which leads to continuous instead of isolated optimal weight solution sets. Because the optimal weight solution sets are continuous, the weights are free to drift without correction from the learning algorithm until two minima in the network energy function are of equal depth and a gross output error occurs. The subsequent correction and later recurrence of these gross errors appears as a series of narrow spikes in the output error of the network. The DBM learning algorithm is incapable of preventing this oscillation because it uses only the final output error of the network to adjust the weights, and the output error is zero for an optimal weight set until a gross error occurs.

The existence of multiple minima in the DBM energy function, and the resulting behavior, is shown to be analogous to prematurely terminating the statistics gathering period in a BM. Since the required period increases with the size of the BM, and the DBM is analogous to an infinite-sized BM, simply increasing the size of the DBM network will not prevent the oscillatory DBM behavior.

Various issues relating to the implementation of DBMs using non-ideal analog hardware, and their relationship to the weight drift problem, are also explored. It is found that only non-ideal behaviors that cause the weight values to drift, most notably weight decay, have a significant effect on network performance, and that there is no threshold below which these behaviors can be tolerated. Other non-ideal analog behaviors, such as component non-linearities, do not seriously degrade network performance.

Acknowledgements

I would like to thank my advisor, Dr. Howard Card, for his help and support during the course of this thesis. I would also like to thank my brother, Christian Schneider, for his work in hardware implementations and many useful discussions.

Financial support from Micronet and the Natural Sciences and Engineering Research Council, and equipment donations from the Canadian Microelectronics Corporation are gratefully acknowledged.

Contents

List of Figures

Glossary

activation	The level of activity of a *unit.* Corresponds to the firing rate of a biological neuron.
BM	A stochastic Boltzmann machine.
basin hopping	Basin hopping is a phenomenon that occurs when a *DBM* network has two energy minima of equal depth to choose from when it is settling, and randomly chooses one or the other. One minimum typically corresponds to a correct answer, and the other to a *gross error.*
bias unit	A *unit* whose *activation* is permanently *clamped* at 1.0 in order to provide a bias or threshold to the other units in the network.
clamped	When a unit is held to a fixed externally applied *activation* value.
continuous solution	A continuous range of *optimal weight sets*, any of which result in an arbitrarily small *output error.*
CHL	Contrastive Hebbian Learning is the learning procedure used in a deterministic Boltzmann machine. The CHL rule is the mathematical formula used to update the weights.
DBM	A deterministic Boltzmann machine.
diameter	(of *optimal weight set*), is the minimum weight change required to produce a *gross error.* The diameter of the weight set is denoted $d(\mathbf{W}^*)$.

epoch	A single pass through the entire *training set.* An epoch is the basic unit of time during training.
error spikes	Brief *gross errors* that appear as narrow spikes in a plot of the *mean squared error* during learning.
free unit	A *unit* that is not *clamped.* The *hidden* and *output units* are usually free, while the input units are clamped.
gross error	A large *output error* that results in a network output having the wrong sign, usually due to *basin hopping.*
hidden unit	A *unit* that is neither an input nor an output, and is never *clamped.* Hidden units are not visible outside the network.
input unit	A *unit* that is always *clamped* to an externally supplied activation level, serving as an input to the network.
isolated solution	An *optimal weight set* that is a single fixed point in weight space. See also *continuous solution.*
learning	The process of adjusting the *weights* to reduce the *output error* of the network to as low a value as possible.
learning multiplier	The electronic circuit responsible for calculating the product of two unit activations during learning.
learning problem	A combination of the set of vectors in the *training set*, the architecture of the network, and the learning success criterion.
mean squared error	The average of the sum of the squares of the difference between the network outputs and the vectors in the *training set.*
non-separable problem	A set of input/output vectors in which the outputs cannot be calculated as a linear combination of the inputs. The exclusive or function is the simplest non-separable problem.

optimal weight set	A set of *weights* that allows the network to recall all the vectors in the *training set* with an arbitrary degree of accuracy. (denoted $\mathbf{W}^*$)
output error	The difference between the output of the network and the vectors in the *training set.*
output unit	A *unit* that serves as an output for the network.
simulated annealing	A process of using gradually decreasing levels of random noise to avoid local minima when searching for a global minimum.
solution	An *optimal weight set.*
state	The state of a unit is its activation level. The state of the network is the set of activations of all the units in the network.
target value	The analog value that the network is supposed to reproduce.
thermal equilibrium	A condition where the average activations of the units of a stochastic Boltzmann machine are no longer changing with time.
training set	The set of input and output vectors used to train the network.
unclamped	When a unit that is sometimes *clamped* is released and allowed to settle with the other *free units.*
unit	A processing element, typically with a logistic activation function. A unit corresponds to a neuron in a biological neural network.
weight	The strength of the connection from the output of one *unit* to the input of another. A weight corresponds to a synapse in a biological neural network.
weight decay	When all the *weights* in the network decrease in magnitude over time.
weight multiplier	The multiplication circuit responsible for calculating the product of a unit activation and a weight value.

Chapter 1

Introduction

The human brain contains approximately one trillion neurons, interconnected through many trillions of synapses. When a neuron cell fires, chemicals are released into its synapses to either excite or inhibit the neurons to which it is connected. Intelligence does not reside in any single neuron, but in the collection as a whole.

It is the goal of artificial neural network research to create intelligent machines by assembling large numbers of simple elements, called *units*, that are modeled after biological neurons. The units are interconnected by *weights* that are the analogous to biological synapses. The *strengths*, or *values*, of the weights represent the efficiency of the synapses and determine how large an effect the activation level, or *state*, of each unit has on those to which it is connected.

Just as biological nervous systems have sensory input and motor output neurons, so artificial neural networks (hereafter referred to as *neural networks*) have input and output units. In both biological and artificial neural networks, the majority of the neurons (units) have no direct connection to the outside world. These are the *hidden* units, and form the most vital part of a neural computation system because they give the system the ability to represent non-separable problems. The process of training, or *learning*, in a neural network involves the adjustment of the weights based on the presentation of training patterns (examples) to the input and output units of the network.

Neural network research, in one form or another, has existed for over three decades. Its popularity has waxed and waned considerably during that time, with a low point in 1969 with the publication of *Perceptrons* [MP88]. In it, Minsky and Papert showed that simple networks (with no hidden units), which had been the main focus of research up to that point, were incapable of representing the entire class of non-separable problems. Although networks with hidden units *could* represent non-separable problems, there was no learning algorithm known that could be used to train them. When the back propagation algorithm [RM87a], which *can* use hidden units to learn problems like XOR, became widely known in 1985, large numbers of researchers once again became interested in neural computation. Many other neural network algorithms have been developed since, but the ability to use hidden units to represent XOR-like problems has remained as a basic requirement in order for a neural network algorithm to be considered useful.

1.1 Neurons in VLSI

An important difference between a serial computer simulation of a neural network model and the corresponding biological or VLSI hardware system is the cost of sending information over long distances. Since every location in the memory of a serial computer is directly accessible, the concept of physical distance between pieces of information does not exist. However, both VLSI circuits and the cerebral cortex are essentially two-dimensional structures, so there are substantial costs in both time and space involved in non-local communication.

Our goal is to build highly parallel neural networks in VLSI, so we must limit ourselves to structures that rely only on local information to accomplish their tasks or face the prospect of devoting the major portion of our integrated circuits to wires. The Boltzmann machine [RM87a] is one of a number of neural network models that meet the local communication requirement. The Boltzmann machine (BM) consists of units whose state switches stochastically, with a probability determined from the inputs received through the weights connecting to other units in the network. The BM has a simple learning rule that relies only on locally available information, making it a natural candidate for VLSI implementation.

One of the disadvantages of Boltzmann machines is that their units change state stochastically, and must be monitored for long periods of time in order to gather sufficiently reliable statistics to perform the weight updates needed to train the network. Peterson and Anderson [PA87] use mean field theory to develop a deterministic version of the Boltzmann machine that circumvents the long monitoring periods by replacing the stochastic unit activations with their expected values.

The deterministic Boltzmann machine (DBM) retains the simplicity of the Boltzmann machine learning rule and avoids the pitfalls associated with stochastic behavior. Because of its advantages, the DBM has been used as a replacement for the Boltzmann machine by a number of researchers [AZL92, BP91, GH90, Hin89, Mov90a, Mov90b].

The goal of our project is to implement the DBM algorithm in analog VLSI hardware in order to exploit the enormous potential for parallelism in the DBM. However, before the the hardware can be built, we have to thoroughly understand the behavior of the DBM so that hardware design efforts can be focused on those issues that seriously affect the performance of the network.

1.2 Learning Instability

In the course of determining which hardware issues are important, a serious tendency towards instability in the DBM learning process, not evident from previous theoretical treatments, was uncovered. While the effects of this instability are most serious in analog hardware implementations, there are also ramifications for both serial and parallel digital implementations of DBM systems.

The root of the instability is shown to lie in the freedom of the weights to drift through a continuous range of values without affecting the output error of the network. Especially when combined with unavoidable non-ideal analog hardware behavior like capacitor leakage, which causes all the weight values to decay towards zero over time, this freedom can cause the learning process to produce a regular series of narrow error spikes. Even in a digital implementation, there is a tendency for the DBM algorithm to produce only marginally stable results.

1.3 Overview

The following is a short synopsis of the contents of each chapter of this thesis. The main results of this work are found in chapters 4, 5, and 6.

Chapter 2: The Boltzmann machine is introduced and its theoretical background is derived. The DBM is then derived by applying the mean field approximation to the BM.

Chapter 3: The experimental process used to determine our results, and the structure of the simulation software, are described in detail. Simulation results are presented for various network configurations and learning tasks. The results, particularly the presence of oscillations during learning, are not consistent with the behavior expected from the theoretical treatment in chapter 2 and in the literature. The cause of this unexpected behavior is explained in chapter 4.

Chapter 4: A small DBM learning the XOR problem is investigated analytically to determine the cause of the unanticipated simulation results in chapter 3. For this particular network, the DBM learning algorithm does not find a unique, isolated, weight set, leaving the weights free to drift. This drifting causes oscillation during learning, especially in the presence of weight decay. While the analysis is restricted to a particular network, the behavior it predicts (instability, oscillation, and sensitivity to weight decay) are prevalent in every non-separable problem we have simulated.

Chapter 5: The relationship between the BM and DBM is explored further to determine if the results derived in chapter 4 are due to a failure of the mean field approximation in small (less than 10 unit) networks. The behaviors of the BM and DBM converge as the networks become large (over 100 units). The convergence is not due to an improvement in the behavior of the DBM, but instead to a deterioration in the behavior of the BM.

Chapter 6: Various design issues pertaining to the implementation of DBMs in analog VLSI are explored, along with their relationship to the findings of chapter 4. The network is tolerant of most non-ideal hardware behavior, but not at all tolerant of weight decay and related effects, which cause the weights to drift and result in oscillation.

Chapter 7: It is demonstrated that a DBM can learn to produce certain analog input/output mappings without the normal process of allowing the units to settle to stable activation values.

Chapter 8: Conclusions and suggestions for further work are presented.

Chapter 2

Theoretical Background

2.1 Introduction

The theoretical background for stochastic and deterministic Boltzmann machines is derived in this chapter. We first provide an overview of both the BM and DBM network architectures, followed by a theoretical derivation of the BM and its learning algorithm. The DBM is then derived from the BM by applying mean field theory.

2.2 Overview of the BM and the DBM

As the name implies, deterministic Boltzmann machines [PH89] are a nonstochastic relative of the Boltzmann machine [AHS85]. The BM represents analog values, whether interpreted as degrees of certainty, or given other meanings, as the expectation value of a probability distribution of stochastic binary valued ($0, 1$ or $-1, +1$) outputs over time. The DBM represents analog values directly as the activation levels of its units.

Figure 2.1 shows a fully connected BM or DBM. Each unit i assumes an activation value according to its input. The instantaneous activations are $S_i = \pm 1$ for the BM, $-1 \leq a_i \leq +1$ for the DBM. The activations of all the units together form the *state* of the network.

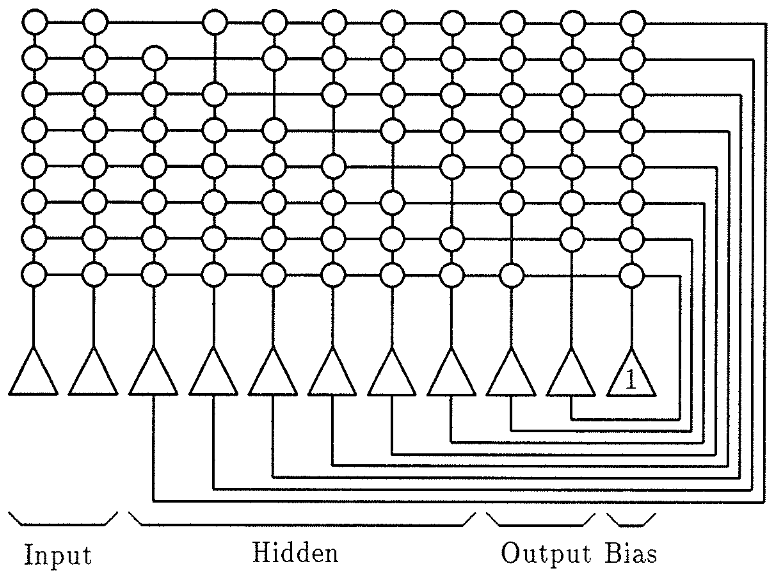


Figure 2.1: Example network configuration. The triangles are the units, the circles are synaptic weights. The vertical lines through the weights represent a distribution of the unit outputs to the weights. The horizontal lines represent a summation of the products calculated in the weights. Note the lack of self-connections.

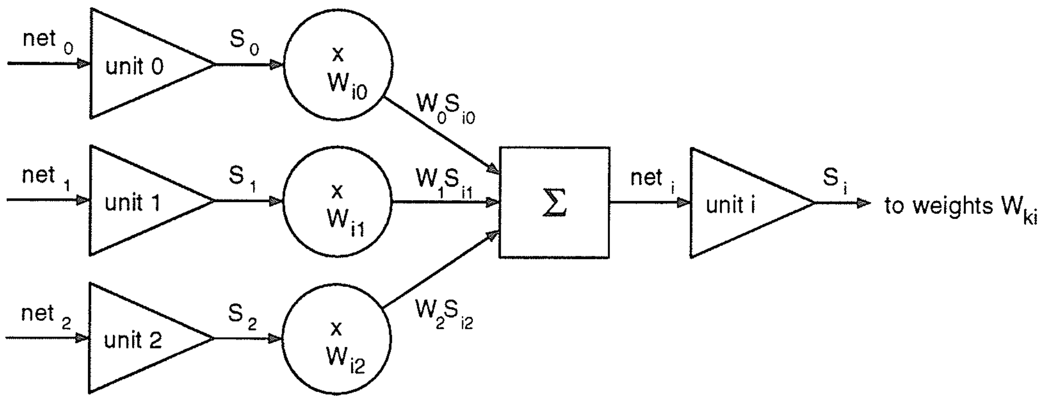


Figure 2.2: Schematic representation of unit input.

The *net input* to unit i is calculated as

$$net_i = \sum_j w_{ij} S_j, \tag{2.1}$$

where w_{ij} is the *weight* from unit j to unit i. The calculation of the unit input is shown schematically in figure 2.2.

The unit activations are updated according to various algorithms, discussed later, until they settle to an equilibrium condition (BM), or a stable state (DBM). This settling process can be viewed as gradient descent in an energy function [AHS85], also discussed later. Intuitively, one can imagine the settling process as an inertialess marble rolling across a hilly terrain and eventually coming to rest in a depression on that surface.

A pattern (or vector) is applied to the input or output units of the network by *clamping* them to fixed activation values instead of allowing them to settle freely. DBM units are clamped to a value between $+1$ and -1, while BM units are clamped to $+1$ or -1 with a probability determined by the value being represented.

The process of training the network to produce the desired mapping of input vectors to output vectors, is called *learning.* Learning involves adjusting the weights until the network responds to the application of each input pattern by producing the corresponding desired output pattern on the output units. Weight adjustment is accomplished by the BM or DBM learning algorithms derived in sections 2.3.2 and 2.4.

Both the BM and the DBM learning algorithms involve repeatedly clamping each input pattern on the input units and alternately clamping and unclamping the corresponding output pattern on the output units. The weights are incremented by an amount proportional to the difference between the products of the activations in the clamped and unclamped *phases* of the learning algorithm.

When evaluating how well a network has learned a task, it is important to keep the learning goals in mind. It is much easier to produce an output with the correct sign, and perhaps a magnitude greater than some fixed value (*digital criterion*), than to produce an exact analog output value (*analog criterion*). Most of the work in the literature employs the digital criterion. We use the analog criterion because it allows the network performance to be evaluated more precisely. Our general philosophy is that if the network is capable of producing the exact desired analog values, then it should do so. The distinction between the two criteria becomes less important when we show (in chapter 4) that a DBM exhibits unstable behavior that causes it to fail both criteria simultaneously.

2.3 Boltzmann Machine Theory

We begin by deriving[1] the process by which the BM units settle to an equilibrium value. We then derive the BM learning algorithm for both *pattern completion* applications, where the network is required to complete a partial pattern clamped on a subset of its input units, and for *pattern association* applications, where there are distinct input and output units and the network must map an input pattern to an output pattern.

[1]The derivations in this chapter closely follow [HKP91]. They are included here for reference and to establish the meanings of the symbols used in later chapters.

2.3.1 Boltzmann Machine Activation Dynamics

Consider a network of N units, each of which can assume a value $S_i = \pm 1$. (This is different from some of the literature where $S_i \in \{0, 1\}$.) The units are interconnected through weights, with w_{ij} representing a weight from the output of unit j to the input of unit i. If the weights are symmetric ($w_{ij} = w_{ji}$) then we can assign an energy function (a Lyapunov function) to the network [PA87]:

$$E = -\frac{1}{2}\sum_{i,j}^{N} w_{ij} S_i S_j - \sum_i \Theta_i S_i \tag{2.2}$$

where Θ_i is a bias or threshold term for unit i. The bias term can be absorbed into the first summation by adding one more unit, called the *bias unit*, with a fixed activation $S_b = 1$ and $w_{ib} = \Theta_i$. (The bias unit is always connected to all the other units.) We then have

$$E = -\frac{1}{2}\sum_{i,j}^{N} w_{ij} S_i S_j \tag{2.3}$$

with N one larger than before.

We want to find the set of activations $\mathbf{S}$ that minimizes E, because that set is maximally consistent with the constraints encoded in the weights w_{ij} and with the external constraints imposed by clamping some of the units to fixed values [AHS85]. We therefore require some sort of gradient descent on E.

Consider a unit k. The difference in E due to the choice of $S_k = -1$ as opposed to $S_k = +1$ is

$$\Delta E_k = E|_{S_k=-1} - E|_{S_k=+1} = 2\sum_i w_{ki} S_i,$$

which is just twice the net input to unit k (see equation (2.1)). Therefore, gradient descent in E is achieved by setting $S_k = -1$ if $\Delta E_k < 0$ and setting $S_k = +1$ if $\Delta E_k > 0$, or

$$S_k = \mathrm{sgn}(\Delta E_k). \tag{2.4}$$

The network activation settles to a minimum by repeatedly choosing k at random and applying (2.4) until no more activation changes take place. Stability is guaranteed because

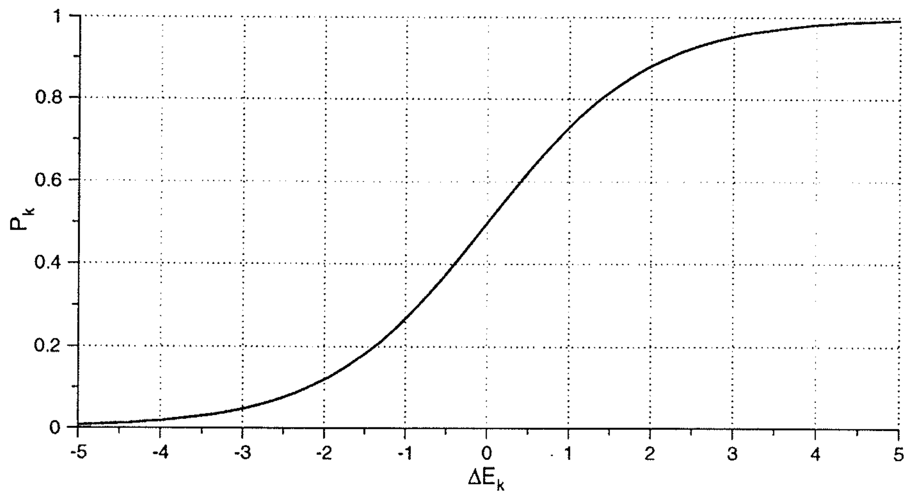


Figure 2.3: Plot of equation (2.5) at $T = 1$.

we are always descending in E. However, the fixed point reached is usually a local, and not a global energy minimum.

One way to escape from local minima, and to find the global energy minimum, is to use *simulated annealing* [AK89]. Simulated annealing employs the addition of a gradually decreasing level of random noise to allow the system to make occasional uphill energy moves to escape from the local minima. Consider setting S_k to $+1$ with probability P_k regardless of its current state, where P_k is defined as

$$P_k = \frac{1}{1 + e^{-\Delta E_k / T}}. \tag{2.5}$$

T is the *temperature*[2] parameter that controls the noise level and therefore the likelihood of uphill energy moves. At $T = \infty$, $P_k = 1/2$ and S_k is equally likely to be $+1$ or -1, regardless of ΔE_k, so all network states are equally likely.

[2]Of course, T is unrelated to the physical temperature of the network. The term "temperature" is used because of the analogy to thermal noise in physical systems.

As T is reduced, the network states with lower energy begin to dominate, although uphill moves are still possible at any temperature above $T = 0$. In the limit,

$$\lim_{T\to 0^+} P_k = \frac{1}{1 + e^{-\mathrm{sgn}(\Delta E_k)\cdot\infty}} = \begin{cases} 0 & \text{if } \Delta E_k < 0 \\ 1 & \text{if } \Delta E_k > 0 \end{cases} .$$

Since P_k is the probability of setting $S_k = +1$ and $(1 - P_k)$ is the probability of setting $S_k = -1$, we have

$$S_k = \begin{cases} -1 & \text{if } \Delta E_k < 0 \\ +1 & \text{if } \Delta E_k > 0 \end{cases} = \mathrm{sgn}(\Delta E_k),$$

which is the same as equation (2.4), as we would expect.

A system updated according to (2.5) at non-zero temperatures eventually reaches thermal equilibrium, and the probability of the various global network states, labeled α, obey the Boltzmann-Gibbs distribution [HKP91]

$$P^\alpha = \frac{e^{-E^\alpha/T}}{Z} \tag{2.6}$$

where Z is a normalizing factor, called the *partition function*, defined as

$$Z = \sum_\alpha e^{-E^\alpha/T}.$$

In physics, T is multiplied by Boltzmann's constant, k_B, but we set $k_B = 1$ which changes the scale of T. Since T does not represent a physical temperature, its scale is arbitrary.

Thermal equilibrium is eventually reached at any non-zero temperature, but this may take a very long time at low temperatures.[3] With simulated annealing, T is gradually reduced while the units are updated using equation (2.5), resulting in the equilibrium state after a much shorter time.

2.3.2 Boltzmann Machine Learning

The purpose of a Boltzmann machine is to perform either pattern completion or pattern association. It is well known that a network requires hidden units to represent a pattern set

[3]Time is normally defined in terms of the number of activation updating passes.

that is not linearly separable [MP88]. The learning algorithm, which we will now derive, must therefore be able to adjust weights connected to the hidden units.

We start by separating the units into visible, with global state α, and hidden, with global state β.[4] The complete state of the network is now labeled $\alpha\beta$. The value of unit i in global state $\alpha\beta$ is $S_i^{\alpha\beta}$. $P^{\alpha\beta}$ is the probability of finding the network in state $\alpha\beta$, and is given by

$$P^{\alpha\beta} = \frac{e^{-E^{\alpha\beta}/T}}{Z} \tag{2.7}$$

where

$$Z = \sum_{\alpha\beta} e^{-E^{\alpha\beta}/T}. \tag{2.8}$$

Since we cannot actually observe the values of the hidden units, their state is not relevant to the visible behavior of the network. We are interested only in P^{α}:

$$P^{\alpha} = \sum_{\beta} P^{\alpha\beta} = \sum_{\beta} \frac{e^{-E^{\alpha\beta}/T}}{Z} \tag{2.9}$$

with

$$E^{\alpha\beta} = -\frac{1}{2}\sum_{i,j} w_{ij} S_i^{\alpha\beta} S_j^{\alpha\beta} \tag{2.10}$$

and Z as in (2.8).

We want the network to learn to reproduce the probability distribution of patterns in the environment on the visible units. Let R^{α} be the probability of pattern α in the environment. P^{α} is the probability of the network producing that same pattern (state) on its visible units. When $P^{\alpha} = R^{\alpha}$ for all α, the network has learned to reproduce the environment exactly. When a subset of the visible units is then clamped with a partial pattern, the remaining units continue to reproduce the now-restricted subset of environmental patterns, and the network functions as a pattern completion network.

[4] Later, we will further separate the visible units into input and output for pattern association.

In order to teach the network the probability distribution of patterns in the environment, we first need a measure of the distance[5] between the probability distribution $\{R^\alpha\}$ in the environment and $\{P^\alpha\}$, the probability distribution produced by the network. Consider

$$\epsilon = \sum_\alpha R^\alpha \log \frac{R^\alpha}{P^\alpha}. \tag{2.11}$$

This equation defines the *relative entropy* [HKP91], a measure of the distance between the two probability distributions. $\epsilon \geq 0$ for any $\{P^\alpha\}$ and $\{R^\alpha\}$, and $\epsilon = 0$ if and only if $P^\alpha = R^\alpha$ for all α. (A proof of this fact is given in the appendix at the end of this chapter.)

We want to minimize ϵ by adjusting the weight set $\{w_{ij}\}$. To perform gradient descent on ϵ, the weights must be updated according to

$$\Delta w_{ij} = -\eta \frac{\partial \epsilon}{\partial w_{ij}} = \eta \sum_\alpha \frac{R^\alpha}{P^\alpha} \frac{\partial P^\alpha}{\partial w_{ij}}, \tag{2.12}$$

where η is a parameter that controls the learning rate. Note that R^α comes from the environment and does not depend on the weights. Expanding P^α according to (2.9) and (2.10), and differentiating with respect to w_{ij} gives

$$\begin{aligned} \frac{\partial P^\alpha}{\partial w_{ij}} &= \sum_\beta \left[\frac{1}{Z} \left(\frac{S_i^{\alpha\beta} S_j^{\alpha\beta}}{T} e^{-E^{\alpha\beta}/T} \right) - \frac{e^{-E^{\alpha\beta}/T}}{Z^2} \sum_{\lambda\mu} \frac{S_i^{\lambda\mu} S_j^{\lambda\mu}}{T} e^{-E^{\lambda\mu}/T} \right] \\ &= \frac{\sum_\beta S_i^{\alpha\beta} S_j^{\alpha\beta} e^{-E^{\alpha\beta}/T}}{TZ} - \frac{\left(\sum_\beta e^{-E^{\alpha\beta}/T}\right) \left(\sum_{\lambda\mu} S_i^{\lambda\mu} S_j^{\lambda\mu} e^{-E^{\lambda\mu}/T}\right)}{TZ^2} \end{aligned} \tag{2.13}$$

We now have to simplify equation (2.13). In the first term, we recognize (from equation (2.7)) that

$$\frac{e^{-E^{\alpha\beta}/T}}{Z} = P^{\alpha\beta},$$

so the first term simplifies to

$$\frac{1}{T} \left(\sum_\beta S_i^{\alpha\beta} S_j^{\alpha\beta} P^{\alpha\beta} \right).$$

The second term can be factored as

$$\left(\frac{1}{T}\right) \left(\frac{\sum_\beta e^{-E^{\alpha\beta}/T}}{Z} \right) \left(\frac{\sum_{\lambda\mu} S_i^{\lambda\mu} S_j^{\lambda\mu} e^{-E^{\lambda\mu}/T}}{Z} \right).$$

[5]The distance between two probability distributions is used to quantify how similar they are to each other.

From (2.9) we have

$$\frac{\sum_{\beta} e^{-E^{\alpha\beta}/T}}{Z} = P^{\alpha}.$$

The expected value of any quantity X that depends on α, with α occurring with probability P^{α} is

$$\langle X \rangle = \sum_{\alpha} P^{\alpha} X^{\alpha},$$

so

$$\frac{\sum_{\lambda\mu} S_i^{\lambda\mu} S_j^{\lambda\mu} e^{-E^{\lambda\mu}/T}}{Z} = \sum_{\lambda\mu} S_i^{\lambda\mu} S_j^{\lambda\mu} P^{\lambda\mu} = \langle S_i S_j \rangle.$$

Putting it all back together and factoring out $1/T$ gives

$$\frac{\partial P^{\alpha}}{\partial w_{ij}} = \frac{1}{T}\left(\sum_{\beta} S_i^{\alpha\beta} S_j^{\alpha\beta} P^{\alpha\beta} - P^{\alpha}\langle S_i S_j \rangle\right). \tag{2.14}$$

Substituting (2.14) into (2.12) gives

$$\begin{aligned} \Delta w_{ij} &= \frac{\eta}{T}\sum_{\alpha}\frac{R^{\alpha}}{P^{\alpha}}\left(\sum_{\beta} S_i^{\alpha\beta} S_j^{\alpha\beta} P^{\alpha\beta} - P^{\alpha}\langle S_i S_j \rangle\right) \\ &= \frac{\eta}{T}\left[\sum_{\alpha}\frac{R^{\alpha}}{P^{\alpha}}\sum_{\beta} S_i^{\alpha\beta} S_j^{\alpha\beta} P^{\alpha\beta} - \left(\sum_{\alpha} R_{\alpha}\right)\langle S_i S_j \rangle\right]. \end{aligned} \tag{2.15}$$

Recognizing that $\sum_{\alpha} R^{\alpha} = 1$ and distributing $1/P^{\alpha}$ in the first term of (2.15) gives

$$\Delta w_{ij} = \frac{\eta}{T}\left[\sum_{\alpha} R^{\alpha}\sum_{\beta} S_i^{\alpha\beta} S_j^{\alpha\beta}\frac{P^{\alpha\beta}}{P^{\alpha}} - \langle S_i S_j \rangle\right].$$

Using the identity $P(\alpha\beta) = P(\alpha)\cdot P(\beta|\alpha)$, we can replace $P^{\alpha\beta}/P^{\alpha}$ with $P^{\beta|\alpha}$, giving

$$\Delta w_{ij} = \frac{\eta}{T}\left(\sum_{\alpha} R^{\alpha}\sum_{\beta} P^{\beta|\alpha} S_i^{\alpha\beta} S_j^{\alpha\beta} - \langle S_i S_j \rangle\right).$$

$P^{\beta|\alpha}$ is the probability that the hidden units are in state β given that the visible units are in state α. The state of the visible units is "given" when they are clamped to a fixed value. The sum over the states of the hidden units gives us the expected value of the products of the activations for each clamped pattern on the visible units, so

$$\Delta w_{ij} = \frac{\eta}{T}\left(\sum_{\alpha} R^{\alpha}\langle S_i S_j\rangle^{\alpha}_{\text{clamped}} - \langle S_i S_j\rangle_{\text{unclamped}}\right).$$

Finally, the R^{α} weighted average over the environmental patterns clamped on the visible units gives $\overline{\langle S_i S_j\rangle}_{\text{clamped}}$, the averaged expected value of $S_i S_j$. The weight updating rule is therefore

$$\Delta w_{ij} = \frac{\eta}{T}\left(\overline{\langle S_i S_j\rangle}_{\text{clamped}} - \langle S_i S_j\rangle_{\text{unclamped}}\right). \tag{2.16}$$

The Boltzmann machine learning algorithm is:

1. Clamp the visible units to a pattern taken at random from the environment being learned. Each pattern α should be picked with probability R^{α}.

2. Start with a large value for T ($T = T_{\text{initial}}$) and allow it to decrease slowly while updating the activations of the units at random using equation (2.5).

3. At $T = T_{\text{final}}$, collect statistics on $S_i S_j$ over a large number of updates.

4. Repeat steps 1 – 3 a large number of times to calculate the average of the expected values of $S_i S_j$ over the probability distribution in the environment. This gives $\overline{\langle S_i S_j\rangle}_{\text{clamped}}$.

5. Perform steps 1 – 3 again, but do not clamp the visible units. This time only a single pass through steps 1 – 3 is required because there is no environmental probability distribution over which to average. This gives $\langle S_i S_j\rangle_{\text{unclamped}}$.

6. Update the weights according to (2.16).

7. Repeat steps 1 – 6 until the network is performing satisfactorily.

Note that in the clamped case we are sampling two probability distributions: the vectors in the environment with probability distribution $\{R^{\alpha}\}$, and the states of the hidden units with probability distribution $\{P^{\beta|\alpha}\}$. The sampling must be over a sufficiently long period of time to ensure that we have reliable estimates of $\langle S_i S_j \rangle$ and can make accurate updates to the weights.

When the environmental vectors have been perfectly learned, $R^\alpha = P^\alpha$ for all α, $R^\alpha P^{\beta|\alpha} = P^{\alpha\beta}$, and $\overline{\langle S_i S_j \rangle}_{\text{clamped}} = \langle S_i S_j \rangle_{\text{unclamped}}$ so $\Delta w_{ij} = 0$ for all i, j and the weight updating stops.

A variation on the weight updating algorithm described above can be derived as follows. Start by rewriting (2.16) as

$$\Delta w_{ij} = \sum_\alpha R^\alpha \Delta w_{ij}^\alpha$$

where

$$\Delta w_{ij}^\alpha = \frac{\eta}{T} \left(\langle S_i S_j \rangle_{\text{clamped}}^\alpha - \langle S_i S_j \rangle_{\text{unclamped}} \right) . \tag{2.17}$$

Using (2.17), the weights can be updated after statistics have been gathered for each clamped pattern α instead of accumulating an average over all the patterns to estimate $\overline{\langle S_i S_j \rangle}_{\text{clamped}}$. Using this method, averaging over the patterns is performed in the weights themselves, which can be a substantial advantage for hardware implementations. The only disadvantage of this method is that the result of each weight update affects the next update, and Δw_{ij}^α becomes time-dependent. We are no longer guaranteed steepest descent in ϵ, but a low learning rate (small η) minimizes this effect.

Looking at equation (2.16) or (2.17), we see that weight w_{ij} is updated from statistics gathered solely from units i and j. These are the two units to which w_{ij} is connected, making the information required for learning *local*. This leads to a very natural parallel hardware implementation of the BM, with the learning logic associated with each weight.[6]

2.3.3 Boltzmann Machine Pattern Associators

A pattern associator maps an input pattern to an output pattern. The visible units must therefore be divided into an input set, with states γ, and an output set, with states α. The

[6]By contrast, back propagation learning requires error derivatives to be propagated through various layers before the weights can be updated.

goal is to train the network to reproduce each output pattern with a probability dependent on the pattern clamped on the input units. In other words, we want $P^{\alpha|\gamma} = R^{\alpha|\gamma}$. In this case, the distance between the probability distributions can be calculated as

$$\epsilon = \sum_\gamma R^\gamma \sum_\alpha R^{\alpha|\gamma} \log \frac{R^{\alpha|\gamma}}{P^{\alpha|\gamma}}.$$

Going through a derivation similar to the one for the pattern completer in section 2.3.2:

$$\Delta w_{ij} = -\eta \frac{\partial \epsilon}{\partial w_{ij}} = \eta \sum_\gamma R^\gamma \sum_\alpha \frac{R^{\alpha|\gamma}}{P^{\alpha|\gamma}} \frac{\partial P^{\alpha|\gamma}}{\partial w_{ij}},$$

$$P^{\alpha|\gamma} = \sum_\beta \frac{e^{-E^{\alpha\beta\gamma}/T}}{Z},$$

$$\begin{aligned} \frac{\partial P^{\alpha|\gamma}}{\partial w_{ij}} &= \frac{\sum_\beta S_i^{\alpha\beta\gamma} S_j^{\alpha\beta\gamma} e^{-E^{\alpha\beta\gamma}/T}}{TZ} - \frac{\left(\sum_\beta e^{-E^{\alpha\beta\gamma}/T}\right) \sum_{\lambda\mu} S_i^{\lambda\mu\gamma} S_j^{\lambda\mu\gamma} e^{-E^{\lambda\mu\gamma}/T}}{TZ^2}, \\ &= \frac{1}{T}\left(\sum_\beta S_i^{\alpha\beta\gamma} S_j^{\alpha\beta\gamma} P^{\alpha\beta|\gamma} - P^{\alpha|\gamma} \langle S_i S_j \rangle^\gamma_{\text{clamped}} \right), \end{aligned}$$

and

$$\begin{aligned} \Delta w_{ij} &= \frac{\eta}{T} \sum_\gamma R^\gamma \left(\sum_\alpha R^{\alpha|\gamma} \sum_\beta S_i^{\alpha\beta\gamma} S_j^{\alpha\beta\gamma} \frac{P^{\alpha\beta|\gamma}}{P^{\alpha|\gamma}} - \left(\sum_\alpha P^{\alpha|\gamma} \right) \langle S_i S_j \rangle^\gamma_{\text{clamped}} \right) \\ &= \frac{\eta}{T} \sum_\gamma R^\gamma \left(\sum_\alpha R^{\alpha|\gamma} \langle S_i S_j \rangle^{\alpha\gamma}_{\text{clamped}} - \langle S_i S_j \rangle^\gamma_{\text{clamped}} \right) \\ &= \frac{\eta}{T} \sum_\gamma R^\gamma \left(\overline{\langle S_i S_j \rangle}^\gamma_{\text{clamped}} - \langle S_i S_j \rangle^\gamma_{\text{clamped}} \right) \\ &= \frac{\eta}{T} \left(\overline{\overline{\langle S_i S_j \rangle}}_{\text{clamped}} - \overline{\langle S_i S_j \rangle}_{\text{clamped}} \right) \end{aligned}$$

where $\overline{\langle S_i S_j \rangle}_{\text{clamped}}$ is averaged with only the input units clamped and $\overline{\overline{\langle S_i S_j \rangle}}_{\text{clamped}}$ is averaged with both the input and output units clamped.

Alternatively, as with the pattern completer, weight updates can be performed for each input pattern, or for each input and each output pattern instead of averaging over all input and output patterns.

2.3.4 Local minima in weight space

Minimizing ϵ by gradient descent makes the implied assumption that ϵ has a unique minimum. If there are local minima, it is possible for $\Delta w_{ij} = 0$ for all i, j while ϵ is still non-zero, and the network has not learned to generate the environmental probability distributions.

Here we are concerned about a local minimum in weight space, a problem that is not solved by the simulated annealing process used to reach thermal equilibrium in activation space. One possible method of avoiding local minima is to add a decreasing level of noise to the weight updates. Both our simulations and those of other researchers show that this is rarely necessary.

Updating the weights after each environmental pattern instead of after averaging over the environment may help avoid local minima in weight space because Δw_{ij}^{α} from equation (2.17) is zero for all α only if the network has learned all the patterns correctly ($\epsilon = 0$), while Δw_{ij} (from equation (2.16)) may be zero if a local minimum in ϵ is reached.

While a non-zero Δw_{ij}^{α} does not guarantee escape from the local minimum, there is a better chance of escape than when $\Delta w_{ij} = 0$. Escape is not guaranteed because the Δw_{ij}^{α} are approximately averaged over time in the weights, but the time-dependence of Δw_{ij}^{α} at least creates an opportunity for escape. This opportunity may be enhanced by applying heuristics to momentarily increase the learning rate η.

2.4 Deterministic Boltzmann Machine Theory

One of the most serious shortcomings of Boltzmann machines is that the values of $\langle S_i S_j \rangle$ during learning, and $\langle S_i \rangle$ during recall, must be estimated by averaging over a period of time. Recall that "time" is measured in terms of the number of activation update cycles of the units in the network. These updates consume *real* time in both simulations and hardware implementations. Another problem is that the estimates of $\langle S_i \rangle$ and $\langle S_i S_j \rangle$ are necessarily noisy, unless averaging is performed over an infinitely long period of time.

Since the network inputs and outputs are all expressed as expected values, it makes intuitive sense to replace the stochastic binary valued (± 1) units with continuous valued units having an activation level $a_k = \langle S_k \rangle$. Mean field theory tells us that this approximation is valid if the number of units in the network is large.

From equation (2.5) we have

$$\begin{aligned} a_k &= \sum_{S_k=\pm 1} S_k P_k \\ &= \frac{2}{1+e^{-\Delta E_k/T}} - 1 \\ &= \tanh\left(\frac{\Delta E_k}{2T}\right) \\ &= \tanh\left(\frac{1}{T} net_k\right) \end{aligned}$$

where

$$\begin{aligned} net_k &= \sum_i w_{ki} \langle S_i \rangle \\ &= \sum_i w_{ki} a_i . \end{aligned}$$

Similarly, $\langle S_i S_j \rangle$ is expressed as $a_i a_j$. Now there is no longer anything random in the network, and the temperature parameter controls the steepness, or "gain" of the $\tanh(\cdot)$ function instead of a noise level (see figure 2.4). The *free energy* [HKP91] in this network is given by

$$F = -\sum_{i<j} a_i a_j w_{ij} + T \sum_i \left[\frac{1+a_i}{2} \cdot \log \frac{1+a_i}{2} + \frac{1-a_i}{2} \cdot \log \frac{1-a_i}{2} \right].$$

The DBM learning rule, also referred to as the contrastive Hebbian learning rule (CHL), is carried over directly from the BM (except that no averaging is required):

$$\Delta w_{ij} = \eta(\breve{a}_i^+ \breve{a}_j^+ - \breve{a}_i^- \breve{a}_j^-),$$

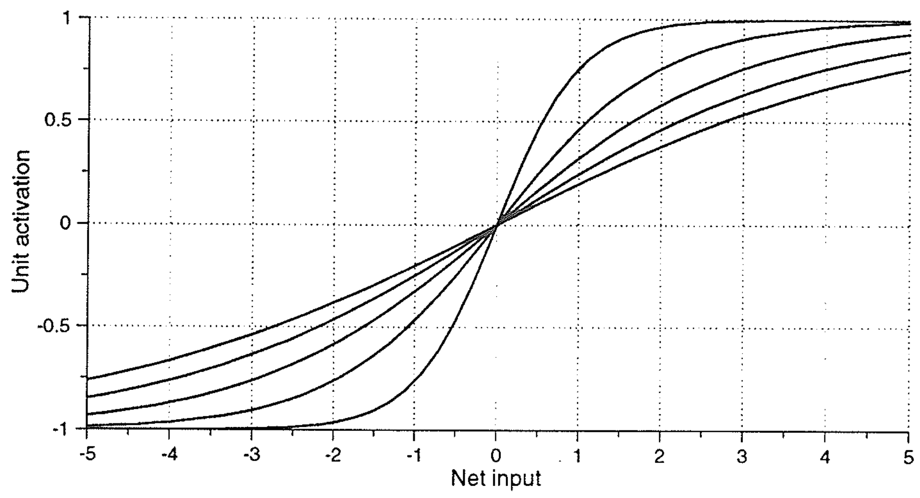


Figure 2.4: tanh(·) gain function for various values of the temperature parameter T. Higher temperatures result in flatter gain curves.

where $\breve{a}_i^+$ is the activation of unit i at equilibrium[7] when the output units are clamped, and $\breve{a}_i^-$ is the activation of unit i at equilibrium when the output units are unclamped ($\breve{}$ indicates equilibrium). The reader is referred to [PA87] for a more formal derivation.

It is important to realize that replacing the stochastic BM activations S_i with their expected values, a_i, is strictly valid only when the number of units in the network approaches infinity. If there are enough units, the contribution of each individual unit to the inputs of the other units becomes negligible, and the instantaneous activation values can be ignored. However, the DBM learning algorithm has also been derived without reference to the BM at all [Mov90b] by minimizing the difference between the free energy F when the network output units are clamped and when they are unclamped. Other researchers [PH89, Mov90b] have reported good results with small ($N \approx 10$) networks.

[7]Equilibrium has a new meaning in the DBM. It is now the final, stable deterministic state to which the network settles at the final annealing temperature, T_{final}.

2.5 Appendix: Relative Entropy

To show that ϵ as defined in equation (2.11) is a valid measure of the distance between the environmental and network probability distributions, we have to prove that $\epsilon \geq 0$ for all probability distributions, and that $\epsilon = 0$ only when the distributions $\{P^\alpha\}$ and $\{R^\alpha\}$ are identical.

First, we prove that

$$\log x \geq 1 - \frac{1}{x}$$

or

$$f(x) = \log x - 1 + \frac{1}{x} \geq 0.$$

The extrema of $f(x)$ occur when $\frac{df(x)}{dx} = 0$. Performing the differentiation, we get

$$\frac{d}{dx} f(x) = \frac{1}{x} - \frac{1}{x^2} = 0 \Longrightarrow x = 1.$$

Taking the second derivative of $f(x)$ and evaluating at $x = 1$, we obtain

$$\frac{d^2}{dx^2} f(x)|_{x=1} = -x^{-2} + 2x^{-3}|_{x=1} = 1 > 0,$$

therefore $x = 1$ is a minimum of f and $f(1) = 0$ so $f(x) \geq 0$.

Substituting $x = R^\alpha / P^\alpha$ and multiplying by R^α gives

$$R^\alpha \log \frac{P^\alpha}{R^\alpha} \geq R^\alpha (1 - \frac{R^\alpha}{P^\alpha}) \tag{2.18}$$

and

$$\begin{aligned} \epsilon = \sum_\alpha R^\alpha \log \frac{R^\alpha}{P^\alpha} &\geq \sum_\alpha R^\alpha (1 - \frac{P^\alpha}{R^\alpha}) \\ &= \sum_\alpha (R^\alpha - P^\alpha) \\ &= \sum_\alpha R^\alpha - \sum_\alpha P^\alpha \\ &= 1 - 1 \\ &= 0 \end{aligned}$$

so $\epsilon \geq 0$. The left and right hand sides of (2.18) are equal only for $x = R^\alpha / P^\alpha = 1$, so $\epsilon = 0$ if and only if $P^\alpha = R^\alpha$ for all α.

Chapter 3

Simulation

3.1 Introduction

The behavior of a DBM is difficult to characterize analytically because it is a multi-layer, non-linear, recurrent network. Therefore, simulation of a DBM on a digital computer is the preferred method of determining network behavior. A simulator, consisting of about 11 500 lines of C code, was developed to test various network configurations, learning tasks, and the effects of non-ideal behavior in hardware implementations.

There are many variations of the simulation algorithm and the network parameters, any of which can change the observed behavior of the network. Although we explored many of these variations, the guiding principle was to restrict the simulations to features that can be readily implemented in a highly parallel manner in hardware. Additionally, the simulations were limited to pattern associators, where an input pattern is mapped to an output pattern.

3.2 Details of the DBM Simulation Algorithm

The following is an overview of the DBM simulation algorithm. The many options and variables that model non-ideal behavior, used to explore the effects of implementing a

DBM in analog hardware, are not described here. The relevant parameters are mentioned later (chapter 6) when the effects of non-ideal components are discussed.

The following is a high-level pseudo-code representation of the simulation algorithm and is sufficient to allow the pertinent details of the simulation to be extracted and the results in this work to be duplicated.

Learning

The learning algorithm and its variations are described below. Underlined steps are described in more detail in following sections. Note that the symbol $\breve{a}_i$ represents the activation of unit i after annealing, when the network is in thermal equilibrium. (Equilibrium is the stable resting state in a DBM)

```
repeat a fixed number of times or until $\overline{error} < MinError$
    for each training pattern
        Unclamp output units
        Clamp input pattern on input units
        Anneal network giving activation vector $\breve{\mathbf{a}}^-$
        Compare output to desired output and compute error
        Clamp desired output pattern on output units
        Anneal network giving activation vector $\breve{\mathbf{a}}^+$
        Compute weight change for all weights in the network using
            $\Delta w_{ij} = \eta \left( \breve{a}_i^+ \breve{a}_j^+ - \breve{a}_i^- \breve{a}_j^- \right)$
        Update the weights
```

There are a number of common variations possible in this learning algorithm:

1. Either the clamped or unclamped phase may be performed first. The choice of which phase is performed first does not have much effect unless either (a) a low initial annealing temperature is used (or no annealing is performed at all); or (b) the weights are updated after each phase instead of after each pattern or after each pass through the entire set of training data.

In case (a), the order is important because the network will tend to remain "stuck" in the energy minimum it settled to in the previous clamped phase. In case (b), the order determines if the positive or negative half of the CHL updating rule is performed first.

2. The training patterns may be presented in a fixed or a random order. Random order is preferable from a theoretical viewpoint, but results in a noisy error. If a fixed order is used, it is best to arrange the order so as to avoid long runs of the same input or output value for a single unit. This decreases the potential of time dependent biasing effects.

3. Weights may be updated after each phase (clamped or unclamped), after each pattern, or after each epoch. Updating after each epoch is preferable for stability, but hard to implement in analog hardware because it requires storage of intermediate results.

4. Weight updates may be by Δw_{ij} or by $\eta\,\mathrm{sgn}(\Delta w_{ij})$ (Manhattan learning). Some researchers have reported good results with Manhattan learning [PH89] and it has some advantages in hardware implementations [Sch91].

5. Error can be computed during learning or as a separate test after each epoch. Testing after each epoch is more representative of a real applications environment, but can mask the learning oscillation described in chapter 4.

Anneal Network

Two different annealing schedules are available. The first is standard geometric annealing [PH89], the second is a slightly modified version of an improved schedule developed by Galland [Gal]. In the following description, $\log_{(T_{\mathrm{ratio}})}$ is a logarithm, base T_{ratio}, and $(T_{\mathrm{ratio}})^n$ is T_{ratio} raised to the power n.

```
**Geometric annealing** (parameters: $T_{\text{initial}}, T_{\text{final}}, nsteps$):
calculate temperature ratio: $T_{\text{ratio}} = \left(\frac{T_{\text{final}}}{T_{\text{initial}}}\right)^{\frac{1}{nsteps-1}}$
for $n = 0$ to $nsteps - 1$
    $T = T_{\text{initial}} \cdot T_{\text{ratio}}^{n}$
    update activations for new temperature $T$

**Galland annealing** (parameters: $T_{\text{initial}}, T_{\text{final}}, T_{\text{ratio}}, T_{\text{error}}$):
calculate number of annealing steps: $nsteps = \log_{(T_{\text{ratio}})}\left(\frac{T_{\text{error}}}{T_{\text{initial}}-T_{\text{final}}}\right)$
calculate residual temp: $T_{\text{resid}} = (T_{\text{initial}} - T_{\text{final}})(T_{\text{ratio}})^{nsteps-1}$
for $n = 0$ to $nsteps - 1$
    $T = (T_{\text{initial}} - T_{\text{final}})T_{\text{ratio}}^{n} + T_{\text{final}} - T_{\text{resid}}$
    update activations for new temperature $T$
```

The advantage of Galland's annealing schedule is that there are more temperature steps taken in the critical temperature range (phase transition) where the unit outputs converge to their final values (see figure 3.1).

A new annealing procedure is described in [AZL92]. It uses a *volatility* measure, computed as

$$q = \frac{1}{N}\sum a_i^2,$$

to measure the degree to which the unit activations have settled near ± 1. The annealing temperature steps are adjusted so that steps in q are not larger than a predetermined value. Alternatively, the temperature is adjusted to keep q at a constant value without annealing.

The volatility based annealing procedure is not easily implementable in analog hardware because it requires storage of activation values from the previous temperature step so that the current temperature step can be rerun with a smaller step size if the change in q is too large. The second approach, where q is kept constant, has the disadvantage that the value of q required for proper operation is highly problem dependent.

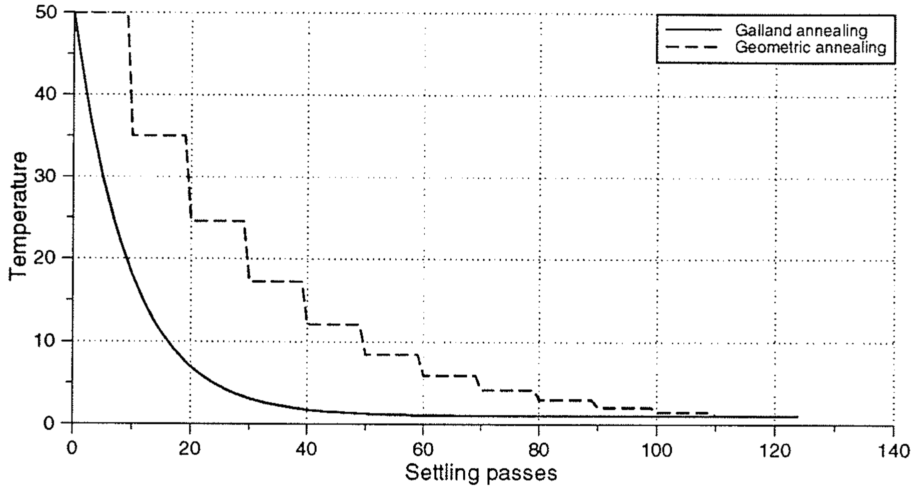


Figure 3.1: Comparison of Galland's annealing schedule with geometric annealing. Galland's technique has more temperature steps (smooth curve) in the critical range of temperatures where the activations are settling to their final values. (between $T = 1$ and $T = 5$)

Update Activations

First, calculate the net input to a unit:

$$net_i = \sum_j w_{ij} a_j,$$

and then update the unit activation asynchronously, using

$$a_i = \tanh\left(\frac{net_i}{T}\right).$$

The order of unit update may be deterministic or random.

Alternatively, we can use synchronous updating, where all the activations are pre-computed and then simultaneously updated using

$$a_i^{t+1} = (1-\tau)a_i^t + \tau \tanh\left(\frac{net_i^t}{T}\right),$$

where the superscript t indicates a time index and τ is the synchronous time step. τ acts as a damping factor to make the network settle more reliably and is picked as a number somewhat less than one (usually $\tau = 0.8$).

When using geometric annealing, there are typically many activation updating passes at each temperature. With Galland annealing, there are many more temperature steps, so only a single pass is normally used at each temperature.

3.3 Simulation Parameters

All simulations in this work were performed with the following choice of parameters unless otherwise indicated. An experiment was rerun with different parameters whenever there was any concern that a result might be an artifact of a particular choice of parameters.

Parameter	**Value**	**Reason for choice**
Annealing type	Galland	better results than geometric annealing
T_{initial}	50	high enough to give initial $a_i \approx 0$
T_{final}	1	arbitrary, simplifies calculations
T_{ratio}	0.9	produces moderate number of T steps
T_{error}	1×10^{-4}	a small error value
nsteps	124	calculated
passes per temperature	1	standard for Galland annealing
activation updating	synchronous	reduces randomness in results
τ	0.8	a number slightly less than 1.0
initial weights	random (-2.0, 2.0)	large enough values for good results
error calculation	during training	oscillation details observable
weight updates	after each pattern	easy hardware implementation
η	0.00001 to 0.1	highly variable depending on problem

3.4 Choice of Problems

The performance of a neural network depends on the learning task with which it is presented and the structure of the interconnections among the units. We define a *problem* as the set of vectors to be learned together with the pattern of connections between the units in the

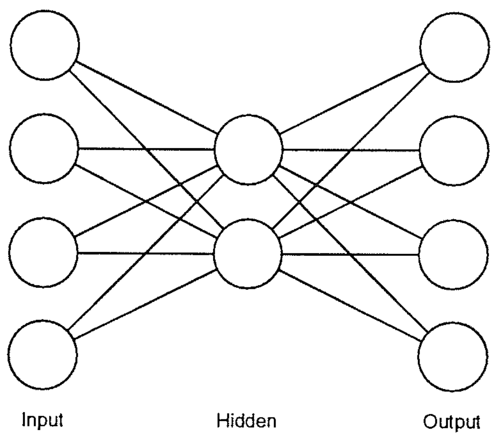


Figure 3.2: Network configuration for the 4-2-4 encoder problem. Notice that the four input and output units are connected only through the two hidden units.

network. For most simulations, a fully connected network, where every unit is connected to every other unit, was used. This is the most general case, and gives the network the most freedom in choosing the weights to solve the problem. The various unit configurations used here are described with the notation

$$\textit{input} \times \textit{hidden} \times \textit{output}$$

where *input*, *hidden*, and *output* are the number of each type of unit. In addition, all networks have a bias unit that is permanently set to an activation level of $+1$. All networks are fully connected unless otherwise noted.

One interesting problem, the 4-2-4 encoder, requires a special network with no input/output connections (see figure 3.2). The goal of the encoder is to map a one-of-four input to an identical output, and to force the network to communicate the one-of-four value through only two intermediate units. While the one-of-four to one-of-four input/output mapping is not in itself very interesting, the 4-2-4 encoder does demonstrate that a DBM network can learn to encode an internal representation of an external stimulus on the hidden units. The 4-2-4 encoder will be examined further in chapter 6.

The most important feature of neural network algorithms like CHL is that they are capable of discovering how to use the hidden units in the network to solve a problem. Because hidden units are only needed for problems that are not linearly separable, most of the simulations conducted are of non-separable problems. The simplest non-separable problem is exclusive or (XOR). When the XOR problem is extended to n bits, it becomes n-bit parity, which is computed as the XOR of each successive bit. Most simulations in this work use a two-input XOR because it represents the simplest non-trivial problem.

3.5 Testing Process

The initial weight set can have a significant effect on the outcome of a network simulation. In order to avoid a bias due to a particular set of initial weights, five separate simulation runs are performed for each experiment, each starting with a different set of random weights. The same five sets of initial weights are used for all experiments that use the same sized network so that results can be compared directly, without the complicating factor of additional randomness introduced by different sets of initial weights.

3.5.1 Learning Plots

The mean squared error is recorded during training as a measure of the performance of the network. Mean squared error is a measure of the difference between the actual and desired output of the network. It is defined as

$$\mathrm{MSE} = \frac{\sum_{r=1}^{n} \sum_{j=1}^{m} (\breve{a}_j^r - t_j^r)^2}{n \cdot m},$$

where n is the number of vectors, m is the number of output units, $\breve{a}_j^r$ is the equilibrium value of output unit j when with vector r clamped on the inputs, and t_j^r is the desired (target) value of output j for input vector r.

For "binary" problems, where the target values consist of only a single positive and a single negative value, such as ± 0.4, the "fraction correct" statistic is also recorded. In such cases, an output is considered "correct" as long as it has the correct sign, which often occurs

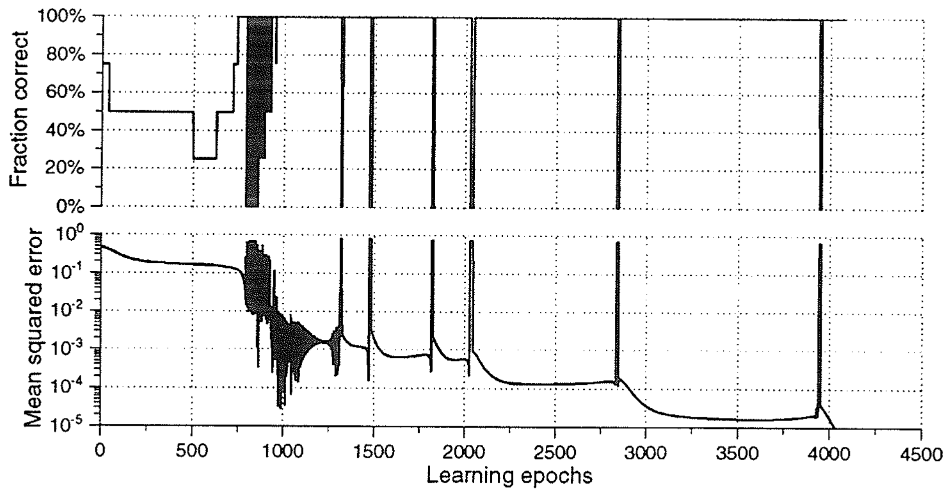


Figure 3.3: Sample learning performance plot.

long before the analog MSE measure becomes small. An output vector is considered correct only if the signs of *all* the output units match the signs of the training values. (Only a single output unit is used in most of our tests.)

Figure 3.3 shows a typical plot of learning performance. The horizontal axis represents the number of passes through the training set, or epochs, while the vertical axes indicate the mean squared error, plotted on a log scale, and the percentage of correctly recalled vectors. The cause of the narrow error spikes will be explained in chapter 4.

3.5.2 Energy Plots

If a network has only two free units (one hidden and one output) the free energy of the network can be plotted for all activations of the free units for a single input vector. The energy function

$$F = -\sum_{i<j} a_i a_j w_{ij} + T \sum_i \left[\frac{1+a_i}{2} \cdot \log \frac{1+a_i}{2} + \frac{1-a_i}{2} \cdot \log \frac{1-a_i}{2} \right]$$

can be represented as a contour plot as shown in figure 3.4. The energy contour plot has two shortcomings:

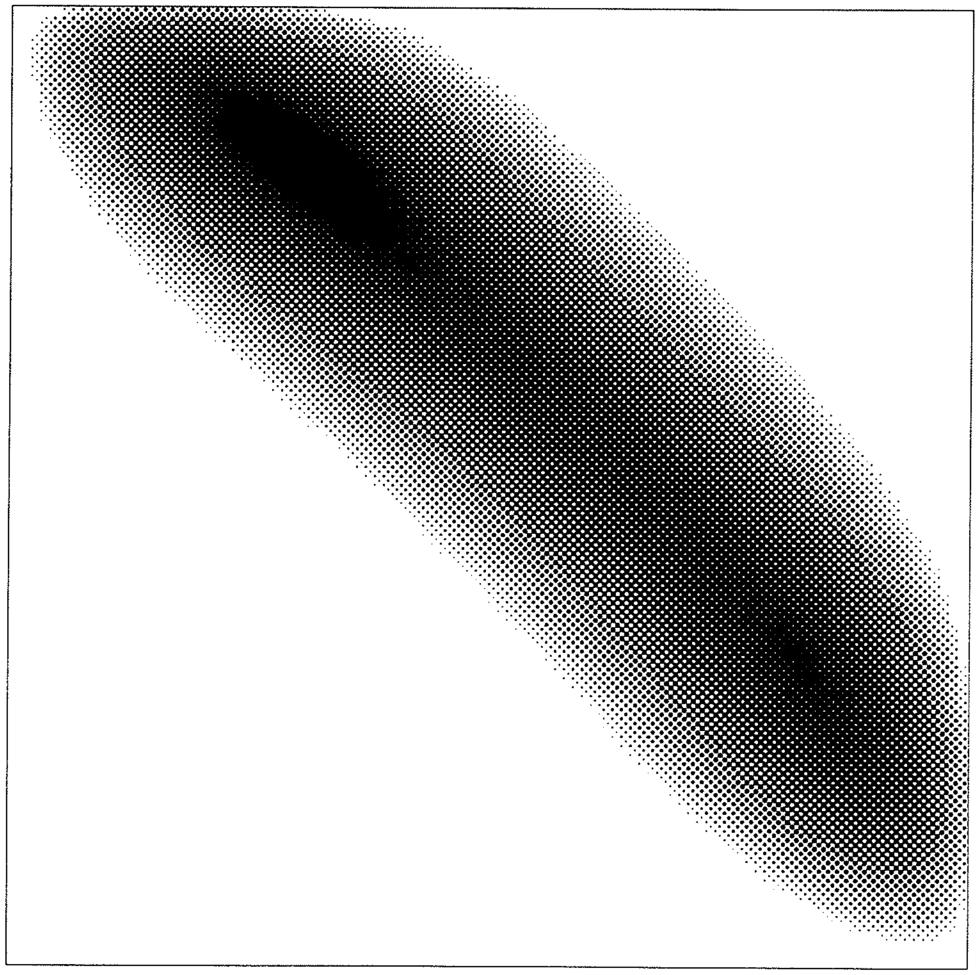


Figure 3.4: A contour plot of the energy surface for a one hidden unit, one output unit network. The horizontal axis represents the activation of the output unit, from -1 to $+1$. The vertical axis represents the activation of the hidden unit from -1 to $+1$. Depth is indicated by shading, with black being the deepest. (Note: the "depth" is clipped to 6% of its highest value in order to make the interesting regions around the minima visible.)

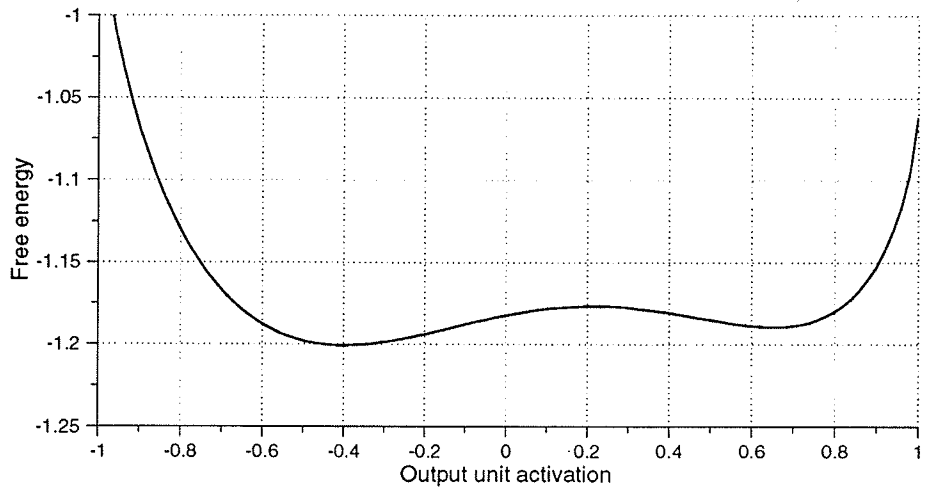


Figure 3.5: A cross section of the free energy contour plot in figure 3.4 along the deepest part of the valley between the two minima.

1. It is very difficult to determine precise values of F from the shading. A three-dimensional perspective plot is no better.

2. The contour plot can only represent two independent variables, so it is only usable for networks with two free units.

The obvious solution to the first problem is to view a "slice" through the energy surface as a standard cartesian plot. The difficulty with this approach is that the precise location at which to take the slice is hard to determine. In most cases, we are interested mainly in the depth of the minima and the profile of the valley connecting them. Therefore, we have developed an alternate way of representing the energy function — an output unit is clamped to successive values in its activation range from -1 to $+1$ and the remaining free units are allowed to settle using the standard annealing procedure. The result is shown in figure 3.5.

The same technique can be applied to networks with more than two free units, but the results must be interpreted with care because of the possible existence of other minima. The technique generally works because the clamped output unit severely restricts the freedom of the other units, so a good approximation of a two dimensional cross section of a multi-

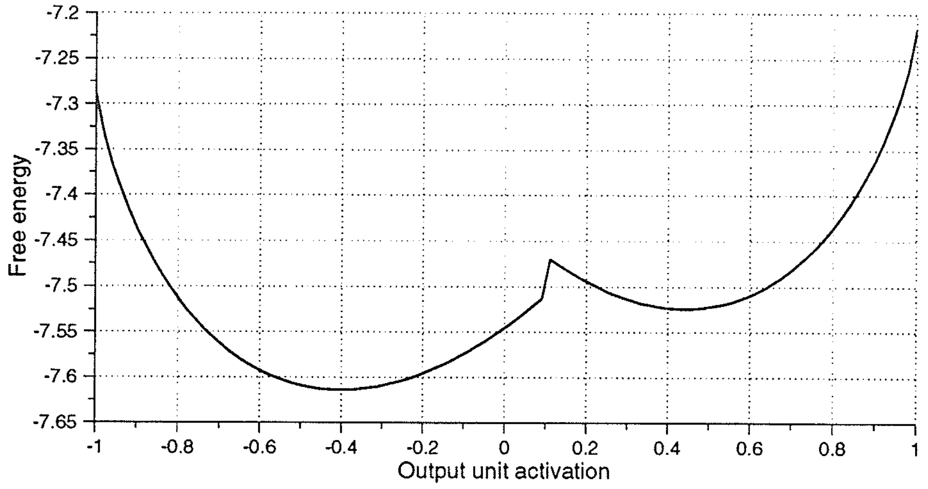


Figure 3.6: Two dimensional cross section of a 10 dimensional energy surface. Note the sharp corner between the two minima, indicating a switch between minima in the hidden units.

dimensional surface is produced near the minima. However, the valley between the minima often features a sharp corner, indicating a jump between minima internal to the hidden units[1] (see figure 3.6).

3.5.3 Weight Plots

A special program was used to allow weight values to be observed during simulation, with positive weight values drawn as green squares and negative ones as red squares. The intensity of the color indicated the strength of the weight. To avoid having to reproduce color here, we use a Hinton diagram [AHS85] (see figure 3.7). Each weight is represented by a square, and its position in the matrix indicates the units to which it is connected. The horizontal dimension is "from", the vertical is "to". The size of a square indicates the relative value of the weight — white is positive, black is negative. A solid gray square indicates a zero, or near-zero, weight.

[1]If there is more than one output, the unclamped output unit(s) may also be involved in the transfer to a new minimum.

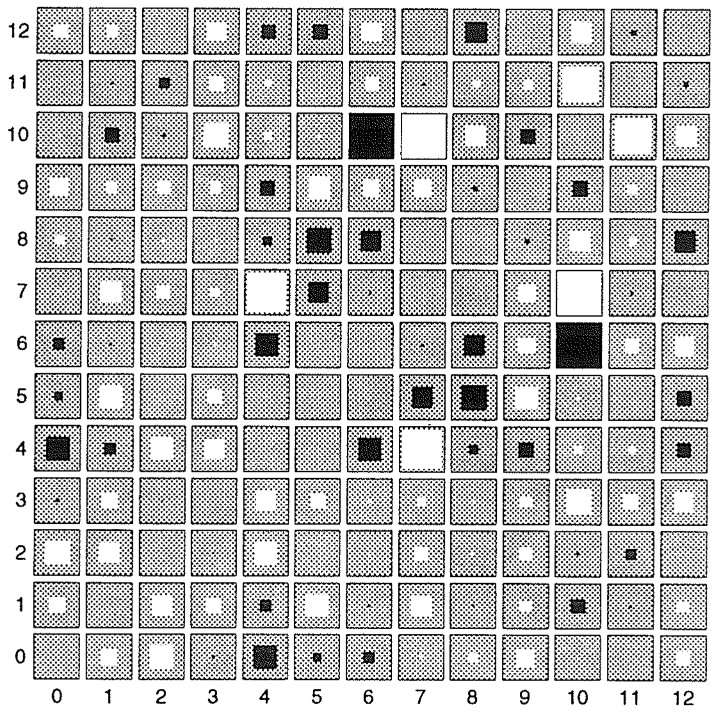


Figure 3.7: A Hinton diagram of a $2 \times 9 \times 1$ network. Horizontal axis is "from", vertical axis is "to". Units 0 and 1 are inputs, units 2 through 10 are hidden, unit 11 is the output, and unit 12 is the bias unit.

3.6 Simulation Results

The first simulations were performed on an arbitrarily chosen non-separable problem. The mean squared error and fraction correct were averaged over ten learning epochs and the results plotted. Performance appeared fairly good, but never quite reached perfection (results are not plotted here).

When the averaging was removed, it became apparent that what seemed like a small, slightly noisy background error was actually long periods of 100% correct performance punctuated by a regular pattern of narrow error spikes. Although the *average* error performance was good, the observed error spikes were inconsistent with proofs in the literature [Hin89] that showed that the CHL learning algorithm performed steepest descent in weight space and should therefore settle to a stable minimum error value. Other researchers [Mov90b, BP91] have also observed these error spikes under various conditions. A long series of experiments employing variations of the learning algorithm, as described in the previous section, generated varied results, but did not completely eliminate the periodic error spikes.

One of the peculiarities of this series of experiments is that a change to a simulation parameter rarely had a predictable effect. Every parameter change, and every different set of starting weights, produced a different result, but in a seemingly random way. For example, a small learning rate η is expected to make the learning process slow and stable, but the effect is often to make it slow and *unstable* (See figure 3.8).

3.6.1 Exclusive Or

Since success with the initial problem was elusive, the learning task was simplified to a two bit exclusive or. Separable problems like a logical "and" were also tried, but they were learned perfectly almost immediately, and, since they require no hidden units, are generally considered uninteresting. Moderate-sized networks are used in these experiments because it is well known that the mean field approximation on which the DBM is based only holds

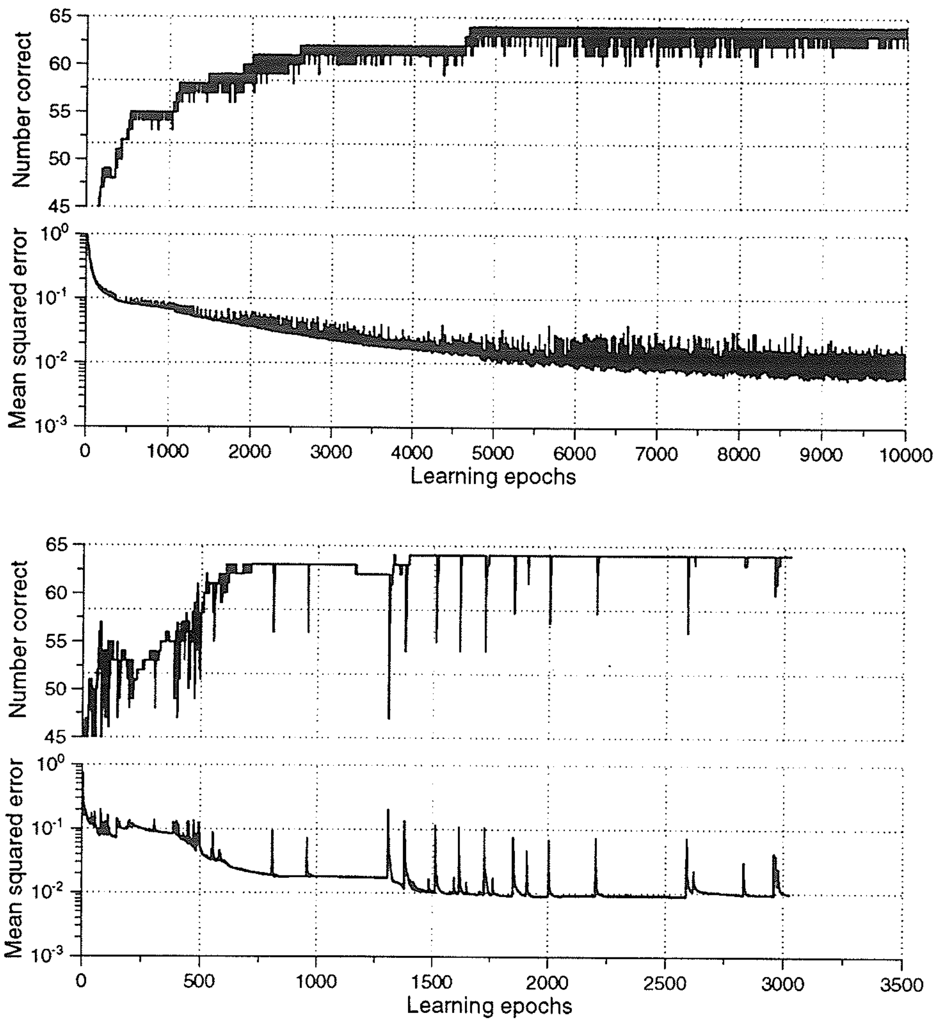


Figure 3.8: Learning test of a $6 \times 10 \times 1$ network mapping 64 6-bit input vectors to a random 1-bit output. The experiments are identical, except that the top one used a learning rate of $\eta = 0.001$ while the bottom one used $\eta = 0.05$. Although the error performance is not very good in either case, there is significantly less oscillation when the higher learning rate is used. Note: "number correct" refers to the number of output vectors with the correct sign (maximum 64).

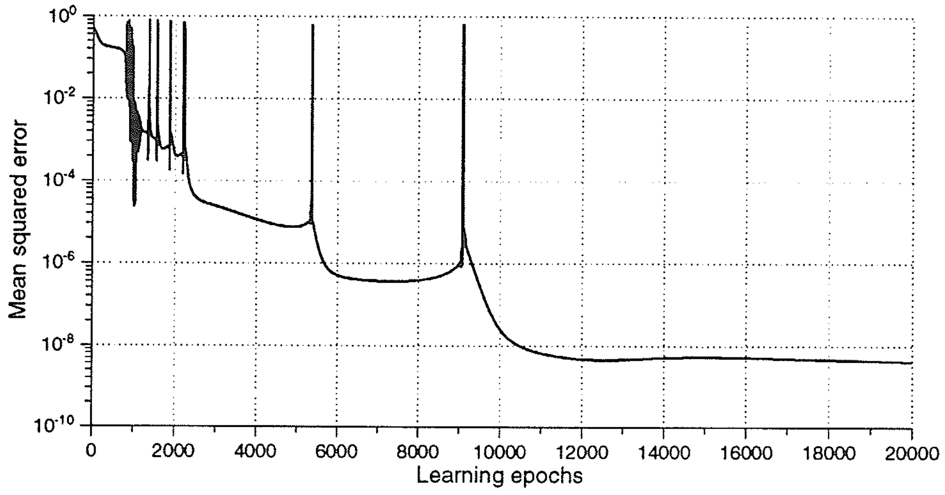


Figure 3.9: Performance of a $2 \times 9 \times 1$ network learning XOR.

if there are a large number of free units [PH89]. Generally, a 13 unit network (10 free units) is used.[2]

By choosing the learning rate carefully, and using initial weights in the correct range, it is possible to achieve what seems like reasonable learning performance (see figure 3.9). The error spikes are less frequent than before and they appear to die out over time. Still, their existence is not really consistent with what is expected of the learning algorithm. Simulations of n-bit parity problems produce similar results (see figure 3.10).

By carefully examining the weight values before and after each error spike, we determined that very small weight changes are sufficient to cause a *gross error*[3] in the network output, and that the spikes are created by tiny weight changes in the course of the training process, usually when weight adjustments due to presentation of one pattern cause recall of another pattern to fail. This led to an investigation of the amount of weight variation a network can tolerate without making an error after it has learned its task perfectly.

[2]Ten free units is not enough for the mean field approximation to be completely accurate, but other researchers have reported good results with networks of this size.

[3]A *gross error* is defined as a large jump in the mean squared error corresponding one or more output units settling to an activation with the wrong sign.

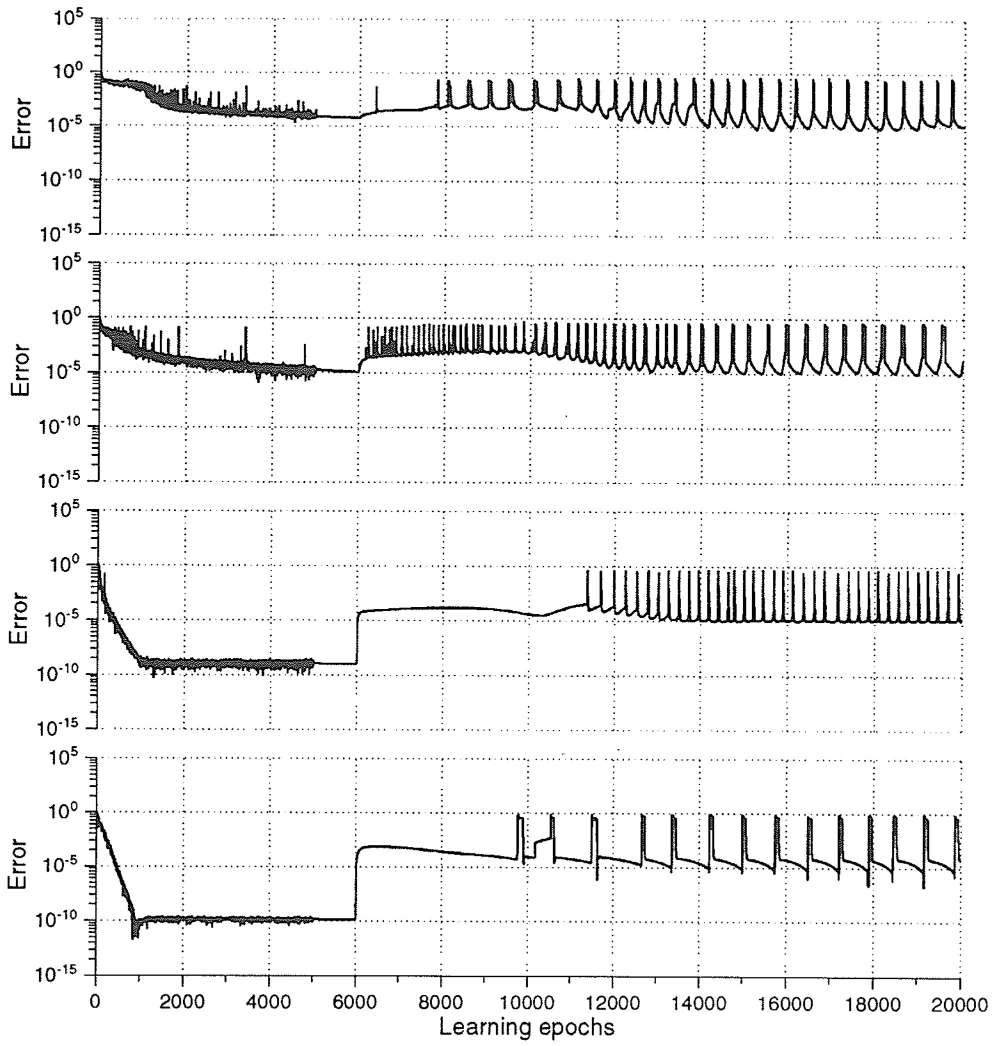


Figure 3.10: From bottom to top: DBM performance on 2, 3, 4, and 5 bit parity problems. In each case, there were as many hidden units as input bits, the learning rate was $\eta = 0.01$ and weight decay of $d = 0.0001$, $d = 0.00005$, $d = 0.000025$, $d = 0.0000125$ respectively was applied after 6000 training epochs. (Decay must decrease as the number of input patterns is increased because decay is per pattern, and we want to compare results on a per epoch basis. The training patterns were presented in random order for the first 5000 epochs, after which they were presented in deterministic order.

In the case of the network of figure 3.9, some weights can only tolerate a change of 3×10^{-5}, or 0.01% of their value before the network makes a gross error. Other networks have been found to be even more sensitive. A requirement for such a high degree of precision makes it necessary to use digital hardware when implementing a DBM. (further discussion in chapter 6)

Another surprising observation is that the network is extremely sensitive to the parameters of the annealing schedule. In the network above (figure 3.8), changing the temperature ratio from $T_{\text{ratio}} = 0.9$ to $T_{\text{ratio}} = 0.889$ (a change of 1.2%) is enough to cause the network to make gross recall errors. This change reduces the number of annealing steps from 124 to 111, which should not be significant. Stranger still, many weight sets have been found to be sensitive to an *improvement* (higher T_{ratio}) in the annealing schedule, where more and smaller steps are taken. A network can be successfully trained with a wide range of different annealing parameters, and performs well as long as it is tested with the same parameters it has been trained with, but fails if they are varied even slightly.

Looking at the energy functions of these networks, it becomes apparent that the energy minima corresponding to the correct and incorrect outputs are of almost exactly equal depth. In some cases, the incorrect minimum is actually deeper than the correct one, but, as long as the annealing parameters are left unchanged, the network recalls the output patterns correctly.

Since one of our goals is to create an analog hardware implementation of a DBM, such requirements for extreme precision are both perplexing and disheartening. Even more perplexing is that some of the experiments produced much better results, with little or no oscillation and little sensitivity to the choice of annealing parameters or small weight perturbations. It seemed that some combinations of starting weights and simulation parameters produced good results, while others did not, but there was no obvious pattern that could lead to a recipe for consistently good results. Clearly, something was lacking in our understanding.

It is evident that simulation alone is not an adequate tool to understand what is happening. The minimal form of a network capable of representing XOR is two inputs, one hidden unit, one output unit, and one bias unit. Much of the behavior observed in the larger networks carries over into the small one, but the small network behaves in a more orderly way because there are only seven independent weights for the learning algorithm to modify. Such a simple network has the advantage that it can be simulated rapidly, but, more importantly, it can be analyzed mathematically to determine the cause of the error spikes. Chapter 4 describes this analysis and resolves the puzzling behavior described here.

Chapter 4

Analysis of a Small DBM

4.1 Introduction

Deterministic Boltzmann machines have usually been analyzed by simulation. Unfortunately, some of the behavior observed during simulation is apparently inconsistent with the results of the mathematical derivation of the DBM in chapter 2 and the literature [Hin89, PA87].

The most striking anomaly is the tendency for the DBM output to become unstable during learning (see figure 4.1). Since the DBM learning algorithm performs gradient descent, oscillation after successful learning is troubling and cannot be ignored. Even if we allow for the possible deleterious effect of updating the weights after each vector is presented instead of averaging the weight update over all the vectors in the environment (as required by the theory), the oscillation is still puzzling. The error spikes in figure 4.1 are very narrow, and can be suppressed in a software simulation simply by averaging the output over time, but this merely covers up an effect that, according to theory, simply should not occur.

The network in figure 4.1 is too large to analyze conveniently in closed form. Therefore, we now move on to a simpler network with a single hidden unit and a single output unit. Figure 4.2 shows the learning curve of such a network with a low learning rate. Note that

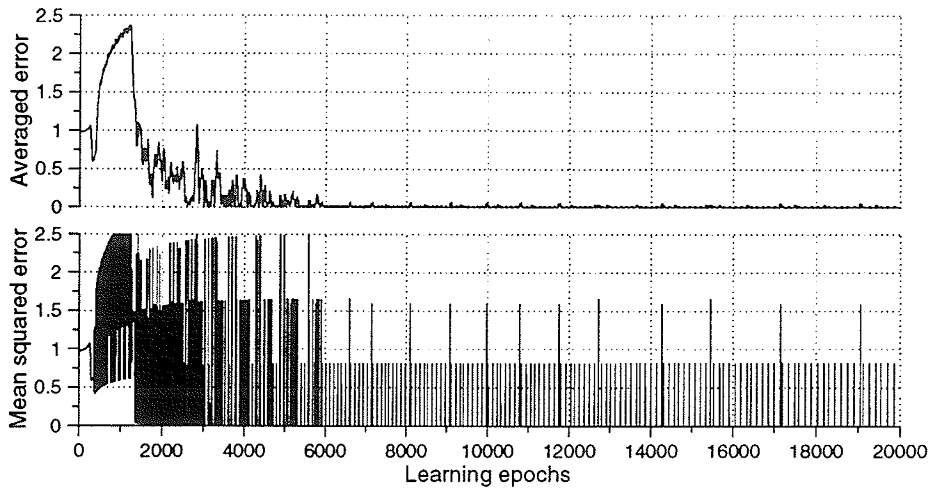
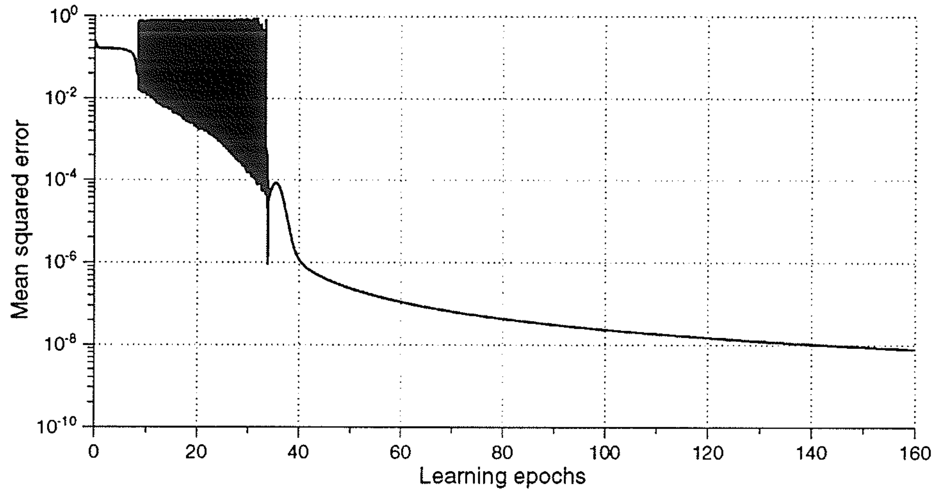


Figure 4.1: This is an example of oscillation in learning. A $2 \times 6 \times 2$ network was used. The outputs were XOR/XNOR, with target values of ± 0.9. Note the *average* error is barely affected by the spikes, especially when the error axis is plotted on a linear instead of logarithmic scale. Learning rate $\eta = 0.0001$.

the oscillation stops part way through the learning process, but, as we shall see, this does not guarantee a stable network.

Close observation of the weights during oscillation in the $2 \times 6 \times 2$ network of figure 4.1 reveals that the difference between the weight values that produce a correct answer and a gross error is very small. By manually varying the value of a single weight in the network of figure 4.2, we can confirm that this network is extremely sensitive to small weight changes. A change of about 1.3% in one weight is enough to generate gross errors. Instead of a gradual degradation in the accuracy of the network output, the answer suddenly jumps from a value of -0.4 to $+0.6$. Not *all* weight solution sets for this problem exhibit such an extreme sensitivity to small weight perturbations, so this behavior cannot be inherent to the network architecture or to the problem being learned, but must be a function of this particular set of learned weights.

Simulation reveals that the use of a moderately high learning rate (η) tends to cause some initial oscillation during learning but results in a far more stable weight set than a low learning rate (see figure 4.3). This observation flies in the face of conventional wisdom



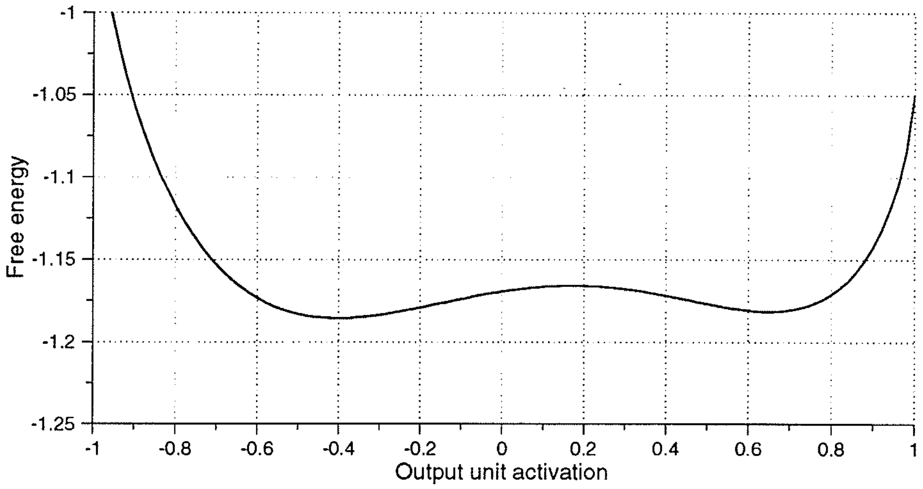


Figure 4.2: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.0005$
Bottom: Free energy through valley between minima, pattern 0. Note the almost equal depth of the minima.

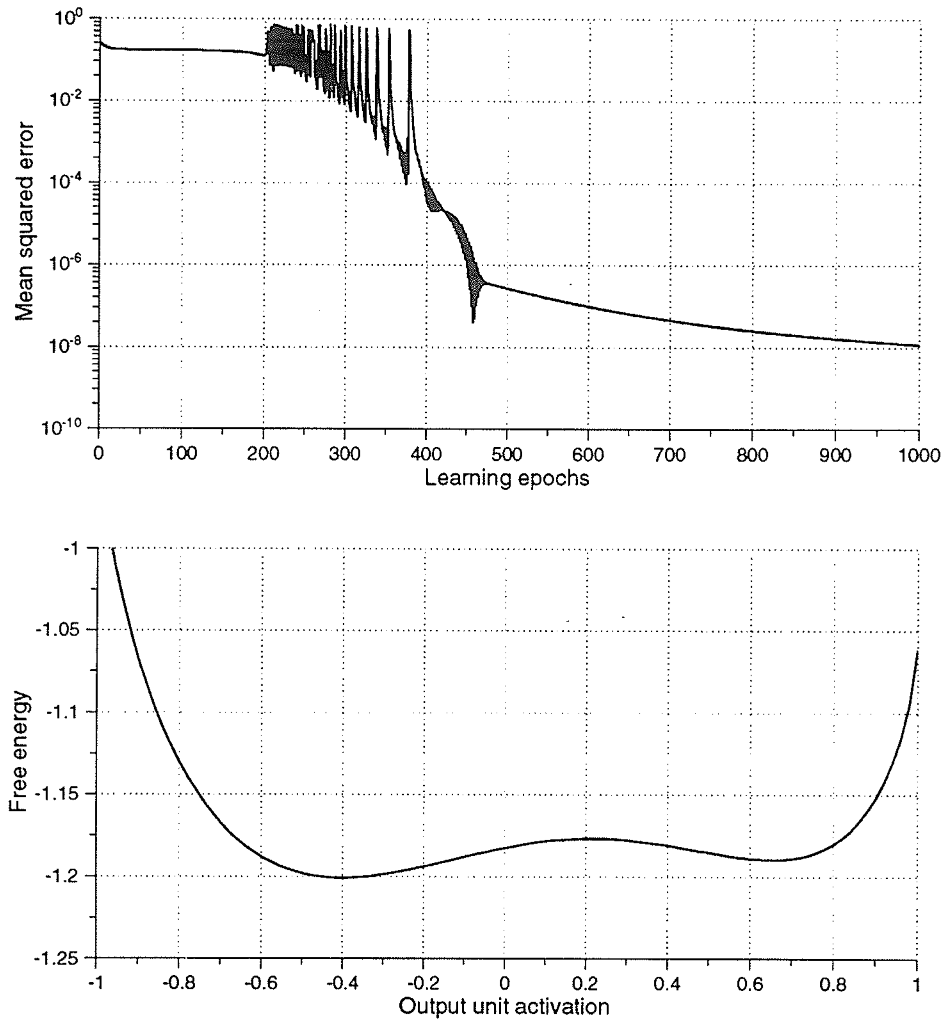


Figure 4.3: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.02$
Bottom: Free energy through valley between minima, pattern 0. Compare the relative depth of the minima here to those in figure 4.2.

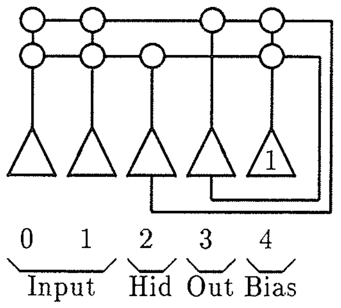


Figure 4.4: DBM network configuration used in this analysis. Triangles are units. Circles are weights.

in numerical simulation, where "go slow" is the standard prescription for a stable, smooth, gradient descent,[1] at the cost of increased simulation time.

The plot of the DBM energy shows that the two energy minima in figure 4.2 are of almost equal depth, while the negative minimum in figure 4.3 is substantially deeper than the positive one. The simulated annealing algorithm ensures that the network always settles in the deepest minimum. If some perturbation, such as a small weight change, shifts the balance between the near equal-depth minima, an incorrect output is produced. If this error is then corrected by the learning algorithm, narrow error spikes, like the ones in figure 4.1, are generated. A proper gradient descent learning algorithm should not exhibit this behavior.

Clearly, our understanding of DBM learning has been incomplete. In this chapter, we analyze a very small DBM in order to explain these phenomena.

4.2 Network Configuration

A fully-connected five unit $2 \times 1 \times 1$ DBM network is used in this analysis (see figure 4.4). Weights are constrained to be symmetric ($w_{ij} = w_{ji}$). Unit activations are calculated by

[1]The number of significant figures in the computer's floating point representation must be taken into account when determining the minimum step size. Roundoff problems have been ruled out here.

$$a_i = \tanh\left(\frac{1}{T}\sum_j w_{ij}a_j\right) \tag{4.1}$$

where a_i is the activation of unit i, w_{ij} is the weight from unit j to unit i, and T is an adjustable parameter (temperature) used in a simulated annealing process to aid in network relaxation. It has been shown [PA87] that this network settles to a minimum of the mean field free energy:

$$F = E - TH,$$

which expands to

$$F = -\sum_{i<j} a_i a_j w_{ij} + T\sum_i \left[\frac{1+a_i}{2}\cdot log\frac{1+a_i}{2} + \frac{1-a_i}{2}\cdot log\frac{1-a_i}{2}\right]. \tag{4.2}$$

The network is trained by clamping the input pattern on the inputs and the desired output pattern on the outputs and allowing the network to settle, giving the vector of activations $\breve{\mathbf{a}}^+$. The outputs are then unclamped and the network is allowed to settle again, giving $\breve{\mathbf{a}}^-$. The weights are updated according to the contrastive Hebbian learning (CHL) rule

$$\Delta w_{ij} = \eta(\breve{a}_i^+\breve{a}_j^+ - \breve{a}_i^-\breve{a}_j^-). \tag{4.3}$$

The following training patterns are used in the analysis presented here:

a_0	a_1	a_3
-1	-1	-0.4
-1	$+1$	$+0.4$
$+1$	-1	$+0.4$
$+1$	$+1$	-0.4

This is a simple two-input XOR problem. The ± 0.4 learning target values are used instead of ± 1.0 to avoid problems with infinite weights.[2]

[2]The activation function of a DBM is $\tanh(net_i/T)$, which reaches ± 1.0 only when $net_i/T \to \infty$. This either requires infinite weights or a zero annealing temperature, neither of which is desirable.

4.3 Network Analysis

It is possible to study the behavior of a network with a single hidden unit analytically to gain further insight into its properties. For now, we are interested only in the activation dynamics during recall, not in the learning process. We want to calculate the set of weights required for the network to recall all the training vectors with arbitrary accuracy. We call this an *optimal weight set*, denoted $\mathbf{W}^*$.

We know that the network settles to a minimum of the mean field free energy F (equation 4.2). The minima must be zeros of the derivative of F. Taking partial derivatives with respect to a_2 and a_3, the activations of the two free (hidden and output) units, and keeping in mind that the weights are symmetric, we obtain

$$\frac{\partial F}{\partial a_2} = -w_{20}a_0 - w_{21}a_1 - w_{32}a_3 - w_{24} + T \cdot \operatorname{atanh}(a_2),$$

$$\frac{\partial F}{\partial a_3} = -w_{30}a_0 - w_{31}a_1 - w_{32}a_2 - w_{34} + T \cdot \operatorname{atanh}(a_3). \tag{4.4}$$

Solving $\partial F/\partial a_2 = 0$ for a_2 gives

$$a_2 = \tanh\left[\frac{w_{20}a_0 + w_{21}a_1 + w_{32}a_3 + w_{24}}{T}\right],$$

which is the activation level the hidden unit (unit 2) settles to for any given inputs a_0 and a_1 and a given activation of the output unit, a_3. Substituting into (4.4) we get

$$\begin{aligned}\frac{\partial F}{\partial a_3} &= -w_{30}a_0 - w_{31}a_1 - w_{32}\tanh\left[\frac{w_{20}a_0 + w_{21}a_1 + w_{32}a_3 + w_{24}}{T}\right] \\ &\quad -w_{34} + T \cdot \operatorname{atanh}(a_3).\end{aligned} \tag{4.5}$$

Unfortunately, $\partial F/\partial a_3 = 0$ cannot be solved for a_3 analytically, but the roots can be found numerically, and the equation can be inspected to ascertain certain properties.

The set of weight values the network learns during a simulation depends on the initial weights, the learning rate, and various other network parameters. Since all the final weight sets produce equally accurate answers during testing, there are obviously many different weight solutions to the XOR problem being learned.

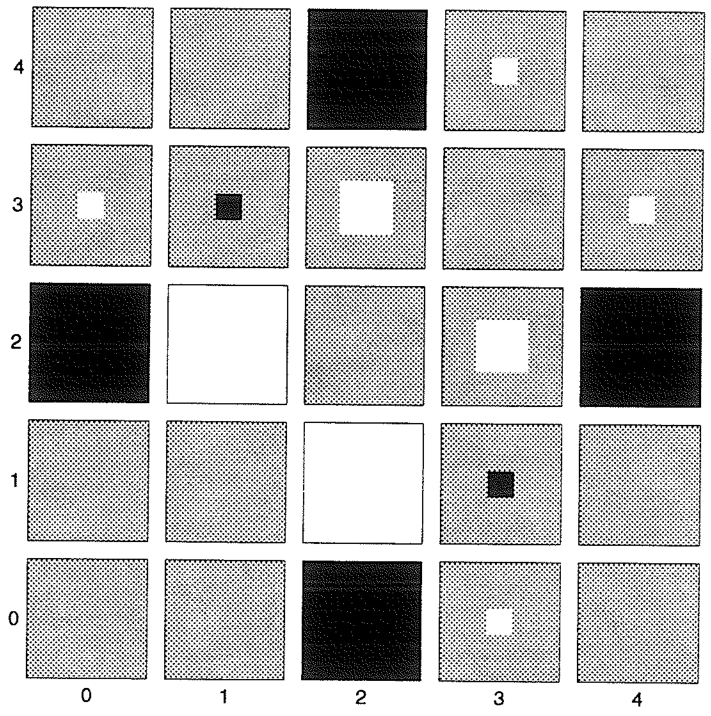


Figure 4.5: A symmetric set of weights that solves the XOR problem. Units 0 and 1 are inputs, unit 2 is a hidden unit, unit 3 is the output, unit 4 is the bias unit.

Inspection of the final weight values in one simulation showed that, in one such optimal weight set, $w_{30} \approx -w_{31} \approx w_{34}$ and $w_{20} \approx -w_{21} \approx w_{24}$. The only remaining weight, w_{32}, was different from any of these (see figure 4.5). Using these constraints, equation (4.5) can be simplified by defining w_a, w_b, and w_c as the three distinct weight magnitudes and making the following substitutions:

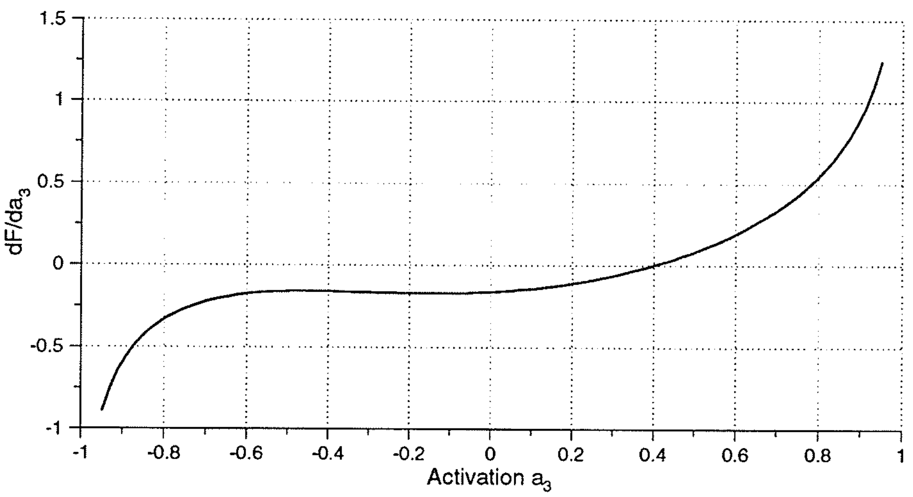


Figure 4.6: Plot of $\partial F/\partial a_3$ (equation 4.6) for typical weights $w_a = 0.482$, $w_b = 0.648$, $w_c = 1.131$, and inputs $a_0 = +1$, $a_1 = -1$. $T = 1$.

$$\begin{aligned}
w_{30} &= w_{03} = w_a \\
w_{31} &= w_{13} = -w_a \\
w_{34} &= w_{43} = w_a \\
w_{20} &= w_{02} = -w_b \\
w_{21} &= w_{12} = w_b \\
w_{24} &= w_{42} = -w_b \\
w_{23} &= w_{32} = w_c.
\end{aligned}$$

Substituting into (4.5), we get

$$\frac{\partial F}{\partial a_3} = -w_a(a_0 - a_1 + 1) - w_c \tanh\left[\frac{-w_b(a_0 - a_1 + 1) + w_c a_3}{T}\right] + T \cdot \operatorname{atanh}(a_3). \qquad (4.6)$$

This equation is plotted in figure 4.6 for a weight set typical of most simulation runs. Notice that $\partial F/\partial a_3$ has only one zero, at $a_3 = +0.4$, so the energy function F has only a single minimum at the desired answer. As we will see later, F can have two minima, which makes it possible to settle to the wrong answer.

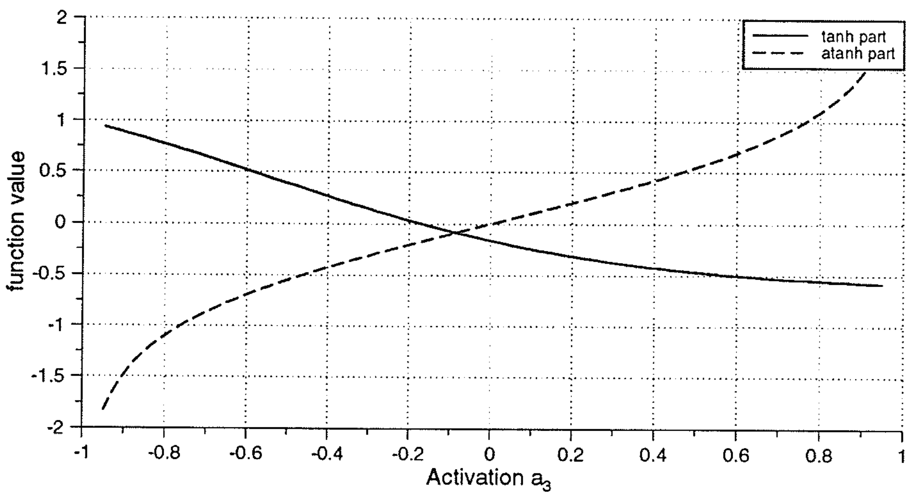


Figure 4.7: Plot of the components of $\partial F/\partial a_3$ (equation 4.6) for the same parameters as in figure 4.6. Note that the tanh function is negative (upside-down) and shifted up and to the left.

Looking at equation (4.6), we see that it is composed of two parts: a hyperbolic arctangent function, independent of input or weight values, and a hyperbolic tangent that is shifted up, down, left, and right by changing the inputs a_0 and a_1. The amount of shift depends on the weights w_a and w_b. The height and width of the hyperbolic tangent are determined by w_c.

To find the weight values for which the network accurately produces the desired outputs in response to the four XOR input patterns, we substitute the values of the XOR vectors for a_0, a_1, and a_3 into equation (4.6) and require $\partial F/\partial a_3 = 0$ for each case. Let A_t be the output target activation ($A_t = 0.4$ in our simulations), and arbitrarily set $T_{\text{final}} = 1$, the final temperature used during annealing.

case $a_0 = -1$, $a_1 = -1 \Rightarrow a_3 = -A_t$:

$$- w_a - w_c \tanh(-w_b - w_c A_t) + \text{atanh}(-A_t) = 0 \tag{4.7}$$

case $a_0 = -1$, $a_1 = +1 \Rightarrow a_3 = +A_t$:

$$w_a - w_c \tanh(w_b + w_c A_t) + \text{atanh}(A_t) = 0 \tag{4.8}$$

case $a_0 = +1$, $a_1 = -1 \Rightarrow a_3 = +A_t$:

$$-3w_a - w_c \tanh(-3w_b + w_c A_t) + \operatorname{atanh}(A_t) = 0 \tag{4.9}$$

case $a_0 = +1$, $a_1 = +1 \Rightarrow a_3 = -A_t$:

$$-w_a - w_c \tanh(-w_b - w_c A_t) + \operatorname{atanh}(-A_t) = 0 \tag{4.10}$$

Equations (4.7) and (4.10) are identical, and, because $\tanh(x) = -\tanh(-x)$ and $\operatorname{atanh}(x) = -\operatorname{atanh}(-x)$, equation (4.8) is just (4.7) multiplied by -1. This leaves us with two equations (4.7 and 4.9) in the three unknowns w_a, w_b, and w_c. Solving (4.7) for w_a gives

$$w_a = w_c \tanh(w_b + w_c A_t) - \operatorname{atanh}(A_t). \tag{4.11}$$

Substituting (4.11) into (4.9) gives

$$w_c \cdot (\tanh(3w_b - w_c A_t) - 3 \cdot \tanh(w_b + w_c A_t)) + 4 \cdot \operatorname{atanh}(A_t) = 0. \tag{4.12}$$

Equation (4.12) cannot be solved for w_b or w_c analytically, but its roots can be found numerically. Taking $w_b > 0$ as the independent variable, we see that the equation has two roots. If $w_b \gg 1$, w_c can be either negative and approximately proportional to w_b, or it can be positive, with a limiting value of

$$\lim_{w_b \to \infty} w_c = 2 \cdot \operatorname{atanh}(A_t). \tag{4.13}$$

The negative root is discarded because it represents an energy maximum at the desired output A_t, instead of a minimum.

Substituting the value of w_c into (4.11), we get

$$w_a = \operatorname{atanh}(A_t) \tag{4.14}$$

Note that the signs of w_b and w_c can be changed, giving an equivalent solution with the meaning of the hidden unit inverted. The values of w_a and w_c are plotted for a range of w_b in figure 4.8.

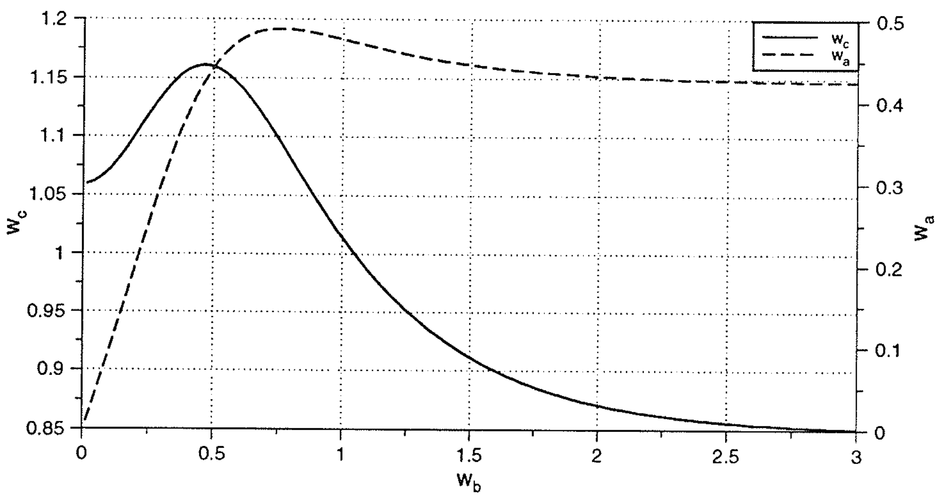


Figure 4.8: Plot of values for weights w_a and w_c for a range of w_b. Any of this family of optimal weights have minima at $a_3 = +A_t$ or $a_3 = -A_t$ as appropriate for the inputs ($A_t = 0.4$ for this plot).

4.3.1 Effects of Different Solutions

It has been shown here that there is not just one, but a continuous range of optimal weight sets that produce an energy minimum at the desired output unit activation $+A_t$ or $-A_t$. If visualized in three dimensions in variables w_a, w_b, and w_c, this range of weight sets appears as a curve. A particular optimal weight set is a single point on this curve. Other weight configurations, such as the one with the meaning of the hidden unit reversed, would appear as distinct curves. We define such a group of weight sets as a *continuous solution*, as opposed to an *isolated solution*, where the optimal weight set is not continuously variable.

Two important questions arise:

1. Are all the optimal weight sets making up a continuous solution equally good?

2. Which optimal weight set does the CHL learning algorithm produce?

To answer the first question, we must decide on a qualitative measure of the desirability of a particular weight set. Since any optimal weight set produces energy minima arbitrarily close to the desired values, the output error is zero in each case and cannot serve as a metric.

Instead, *reliability* is chosen as the performance measure. Reliability can be broken down into three components:

1. The network should tolerate small changes in the weights without producing large changes in the output values. At the very least, the output values should not easily change sign (a *gross error*). If a gross error is due to the output settling to the wrong energy minimum, this is also referred to as *basin hopping.* We define the *diameter* of the optimal weight set, $d(\mathbf{W}^*)$, as the minimum change in a weight required to produce an output with the wrong sign. We generally want $d(\mathbf{W}^*)$ to be as large as possible.

2. The network should be insensitive to variations in the annealing schedule used during settling. Annealing is needed only if there are spurious local minima in the free energy function of the network. The network becomes sensitive to the annealing schedule as spurious minima approach the depth of the desired global minimum.

3. The network should be able to continue learning even after some pre-determined error criterion has been reached. Continued learning is necessary in analog hardware implementations to maintain the charge on the weight-storage capacitors, and also for situations where the network is supposed to track changes in an evolving environment.

In general, the most reliable networks are the ones that have only a single energy minimum, and the most unreliable are those with multiple, nearly equal depth minima.[3] One problem with near equal depth minima is that a small weight perturbation can make the spurious minimum deeper than the desired one, leading to basin hopping and a gross error. The perturbation may be the result of either the learning procedure itself, as illustrated in figure 4.1, or due to weight drift or noise.

The other problem caused by near equal depth minima is that a very thorough, and therefore slow, annealing schedule is required to choose between almost indistinguishable

[3]There are a maximum of two minima in the simple $2 \times 1 \times 1$ network analyzed here.

minima. As we will see later, using a slow annealing schedule during learning is actually counterproductive.

We divide the weight sets in a continuous solution into three classes: *stable solutions*, where there is only one energy minimum; *meta-stable solutions*, where there is a spurious local energy minimum; and *unstable solutions*, where the annealing process (with a particular set of annealing parameters) cannot correctly distinguish between the minima. Annealing is unnecessary for a stable solution set.

A stable solution results as $w_b \to \infty$. This can be verified by substituting (4.13) and (4.14) into (4.6). Because a_3 is a finite number, (4.6) becomes

$$\frac{\partial F}{\partial a_3} = -\operatorname{atanh}(A_t) \cdot (a_0 - a_1 + 1) + 2 \cdot \operatorname{atanh}(A_t) \cdot \operatorname{sgn}(a_0 - a_1 + 1) + T \cdot \operatorname{atanh}(a_3) \quad (4.15)$$

where

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ +1 & \text{if } x > 0 \end{cases}$$

which obviously has only a single zero as a_3 varies for fixed inputs a_0 and a_1.

Figure 4.9 shows how the derivative $\partial F / \partial a_3$ responds to changing the weights. The critical point after which the free energy has only a single minimum is $w_b = 0.48$ in this case. When $w_b > 0.48$, we have a stable solution; when it is less, we either have a meta-stable or an unstable solution depending on how close the two minima are to the same depth.

The minima are of equal depth when $w_a = w_b$. The correct minimum is deepest as long as $w_a < w_b$. Looking at figure 4.10, we see that the solution becomes unstable around $w_b = 0.3$. As the weight set approaches the transition from meta-stable to unstable, $d(\mathbf{W}^*) \to 0$, and the network becomes extremely sensitive to small weight perturbations.

4.4 Learning

The weight change determined by the CHL weight updating rule, equation (4.3), is controlled by the difference between the clamped and unclamped activations of the units.

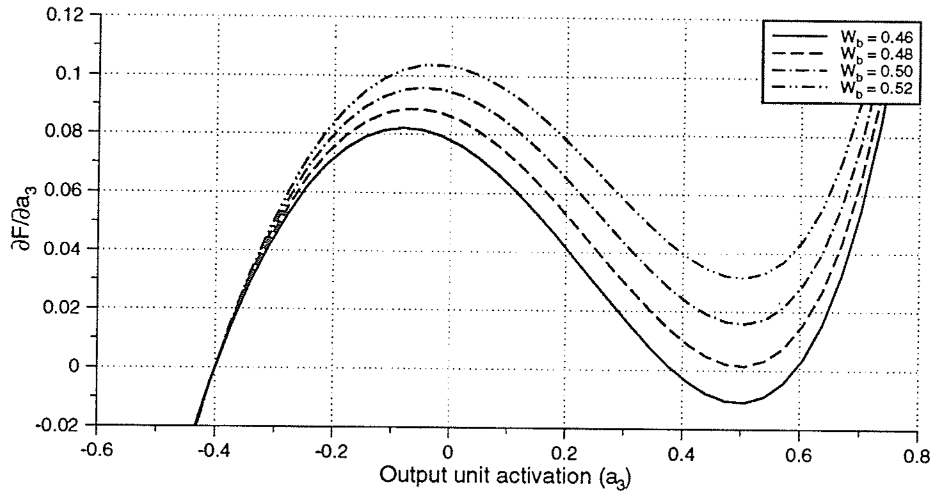


Figure 4.9: The effect of changing w_b (and w_a and w_c according to equations (4.11) and (4.12)). $A_t = 0.4$, desired output $a_3 = -A_t$ for this plot. Notice that $\partial F/\partial a_3$ has three zeros for $w_b = 0.46$, corresponding to two minima in the energy function.

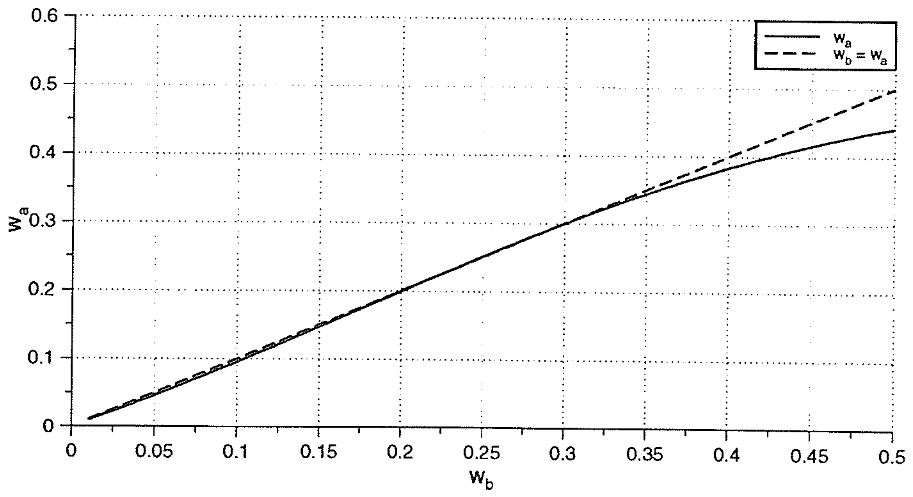


Figure 4.10: w_a as a function of w_b near $w_a = w_b$. We have a correct answer as long as $w_a < w_b$.

Except for a few unlikely, degenerate cases, if the clamped and unclamped output activations match, so do the clamped and unclamped hidden unit activations. Therefore, it is the mismatch in the *output* activations that drives the weight updates during learning.

For the case of a $2 \times 1 \times 1$ network learning XOR, we have shown that there is a continuous range of optimal weight sets that produce equally accurate outputs. Once the weights have been adjusted to produce a zero-error output, no further learning takes place. There is nothing in the CHL learning algorithm that causes it to produce a stable weight set — once the desired output is produced, there is no longer any information on how to further evolve the weight set. The final resting point on the solution curve is determined by the random starting weight set in combination with all the simulation parameters. All aspects of the simulation including variations in the algorithm affect the outcome, but they do so in an essentially random way.

If a low learning rate is used, the most common final weight configuration is very near the $d(\mathbf{W}^*) = 0$ point, with the desired global minimum only minutely lower than the spurious minimum for each input vector (see figure 4.2). This makes the network very sensitive to small weight perturbations or changes in the annealing schedule. In fact, even the small weight adjustments caused by continued learning (if the network has not yet reached exactly zero error) may cause a gross error, leading to the learning oscillation observed earlier.

The size of the neighborhood $d(\mathbf{W}^*)$ tends to be proportional to the learning rate η because the learning steps that cause the error spikes are proportional to η, and a step proportional to η is taken every time an error spike occurs. The network eventually settles far enough from the $d(\mathbf{W}^*) = 0$ point to prevent further error spikes. This is why the error spikes tend to die out as learning continues, and also why networks learning with a large η show less oscillatory behavior than those with a small η.

4.4.1 Weight Decay

Weight decay is an unavoidable feature of analog weight storage capacitors. A common model of weight decay is an exponential drift towards zero. For each weight w_{ij} in the network,

$$w_{ij}^{t+1} = d \cdot w_{ij}^{t} \tag{4.16}$$

where $d < 1$ is the decay factor and the superscript represents the simulation time index. Consider what happens when all the weights in the network analyzed in section 4.3 decay. In the three-dimensional visualization of the continuous solution described previously, this is equivalent to moving on a straight line from a point on the solution curve toward the origin.

We saw in section 4.3.1 that if w_b is large to start with (stable solution set), changing it has little effect on the activations of the units. Changing w_a and w_c changes the location of the energy minimum. If the CHL learning algorithm is applied to the decayed weights, w_a and w_c are adjusted to move the minimum back to $\pm A_t$. If the decayed w_b is still large, the hidden unit still acts as a ± 1 step function, so there is *no change* in w_b due to re-learning after the decay.

Eventually, w_b decays until it is no longer large, and the solution set enters the meta-stable range. The decay process continues, with $d(\mathbf{W}^*)$ becoming smaller and smaller, until the spurious minimum is the same depth as the desired minimum ($d(\mathbf{W}^*) = 0$) and the network makes a mistake (see figure 4.11). At this point, the clamped and unclamped network outputs are no longer the same, and the CHL rule causes a large change in the weights, resulting in a correct output again. The decay process repeats, and the network oscillates. The oscillation frequency is roughly proportional to the the decay rate and inversely proportional to η, but there is no threshold below which decay can be tolerated. Weight decay will be discussed again in chapter 6.

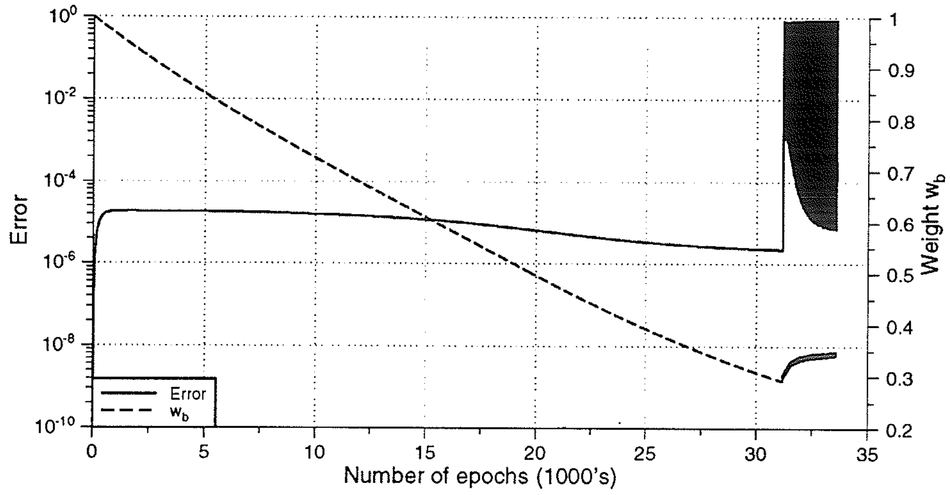


Figure 4.11: The effect of slow weight decay on a network starting with a "good" solution. The relatively high analog error is due to the weight decay. Notice that the error level is essentially constant until the moment when oscillation begins. The digital error (sign of the output) remains zero until the point where the network begins to oscillate. Note that equations (4.11) and (4.12) no longer hold after onset of oscillation, and therefore w_b no longer represents the weight set after that point. ($\eta = 0.01$, $d = 1.0 \times 10^{-5}$)

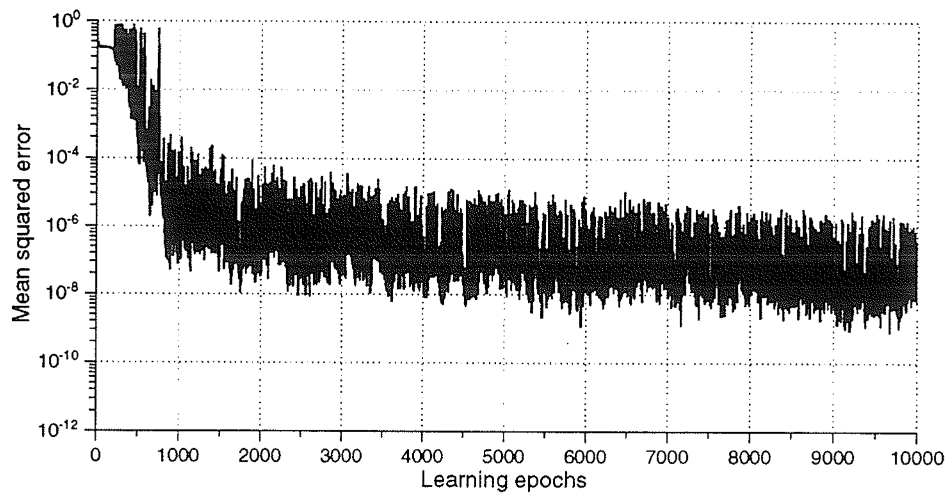
4.5 Solutions

The reason that the CHL algorithm cannot deal with weight decay, or with other related behavior, such as hypersensitivity to weight perturbations, is that learning is based only on the *final* activations of the units after settling. If a thorough annealing schedule is used, only the location of the deepest energy minimum is used in adjusting the weights. The existence, or development (through weight decay), of a spurious minimum goes undetected until it approaches the depth of the desired minimum and causes a mismatch between the clamped and unclamped outputs.

One solution to this problem is to dispense with annealing entirely during learning and start the units at random activations, apply an input or input/output pattern, and then settle using a damped updating rule with no annealing. Because this causes the network to randomly settle into the spurious minima in the unclamped phase, the weights are adjusted until those minima disappear. The possibility of becoming trapped in any shallow spurious minima that may remain or form later is avoided by using normal annealing during recall operations.

Our simulations have shown that this procedure works well for small networks, reliably producing single-minimum energy functions. See figure 4.12 for an example. Figure 4.13 shows that further improvements can be achieved by reducing the number of settling passes at each temperature. Figures 4.14 and 4.15 show how disabling annealing also improves the problems associated with weight decay. Unfortunately, while this technique is promising for small networks, it often causes learning to fail completely in larger networks, likely due to the large number of spurious minima a large network can contain. When a solution is found in a large network, it has good, single-minimum energy functions, but the procedure has too low a success rate to be practical. The procedure is in any case sub-optimal because, while the solutions it produces are reasonably good, they are still far from the ideal $w_b \to \infty$ result determined previously.

Another possible solution comes from the observation that "large weights are good." By using a reverse weight decay procedure where $d > 1.0$ in equation (4.16), unconstrained



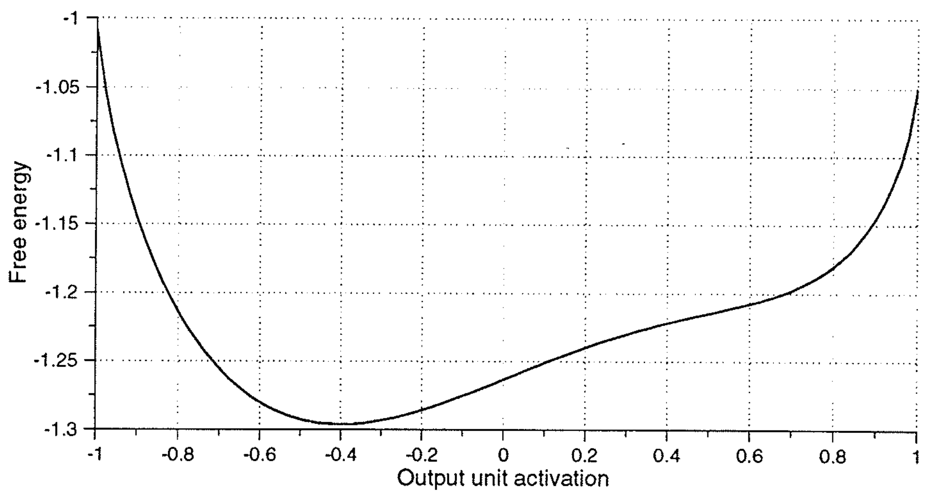
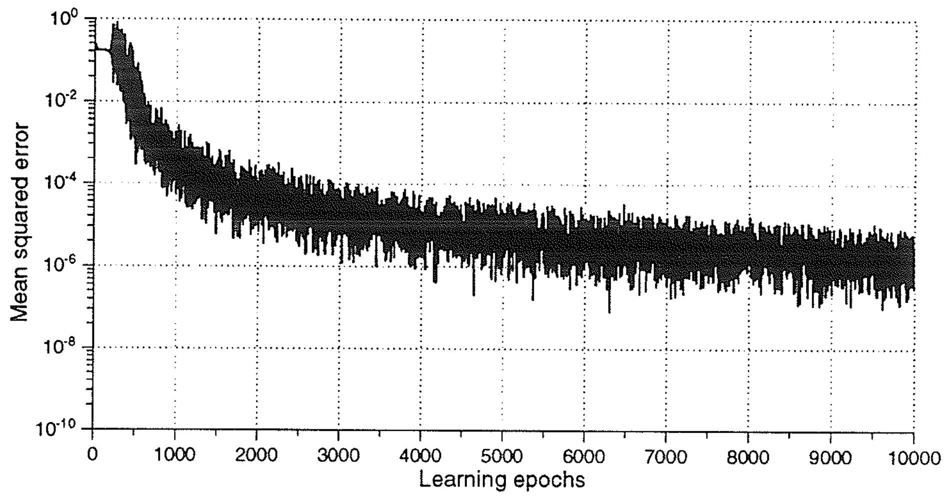


Figure 4.12: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.02$, no annealing, 50 settling passes per input pattern. Note log scale on error axis – noise in error is over a fairly small range. The error does not represent oscillation or basin hopping. Bottom: Free energy through valley between minima, pattern 0. Compare with figure 4.3 on page 45. Notice that there is only a single minimum in the energy function.



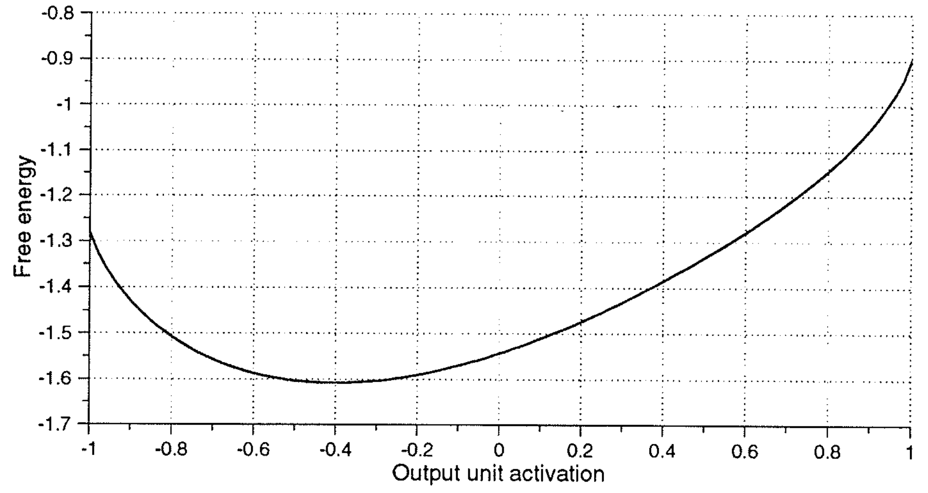


Figure 4.13: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.02$, no annealing, 12 settling passes per input pattern. The error is noisy but exhibits no basin hopping. Bottom: Free energy through valley between minima, pattern 0. Note absence of any hint of a second energy minimum.

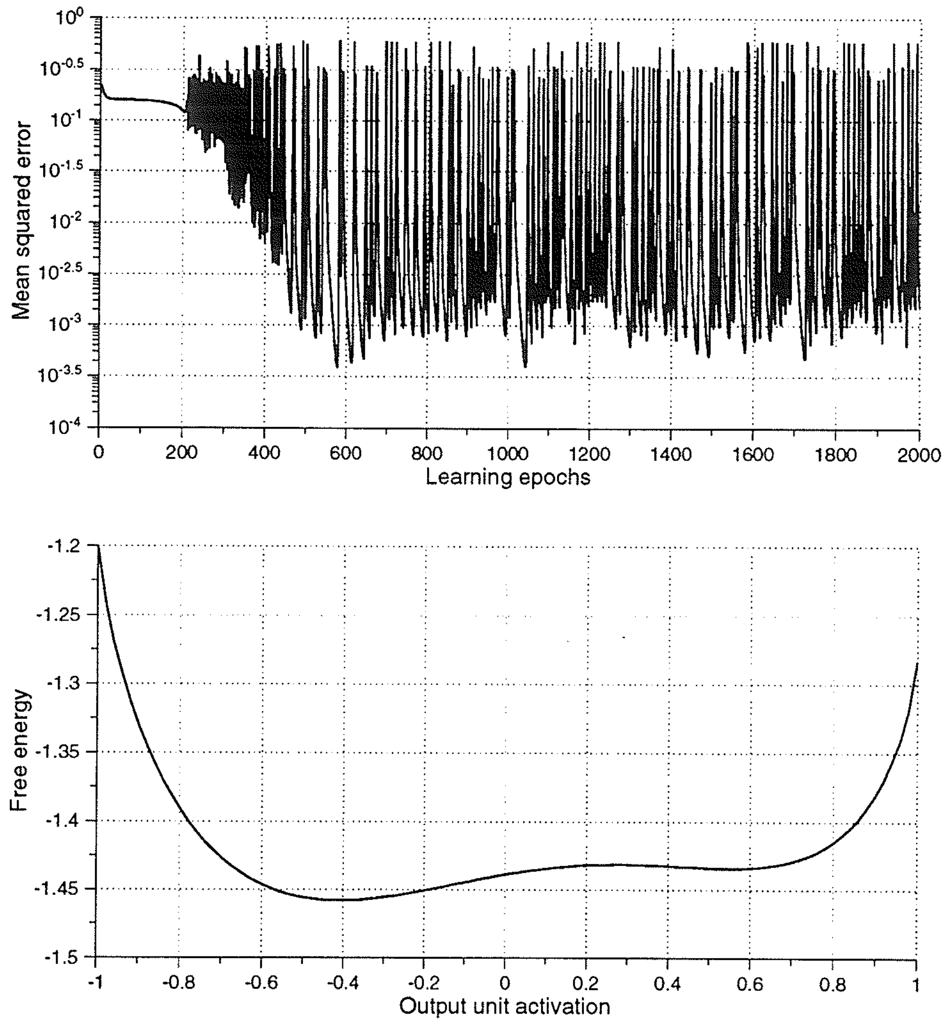
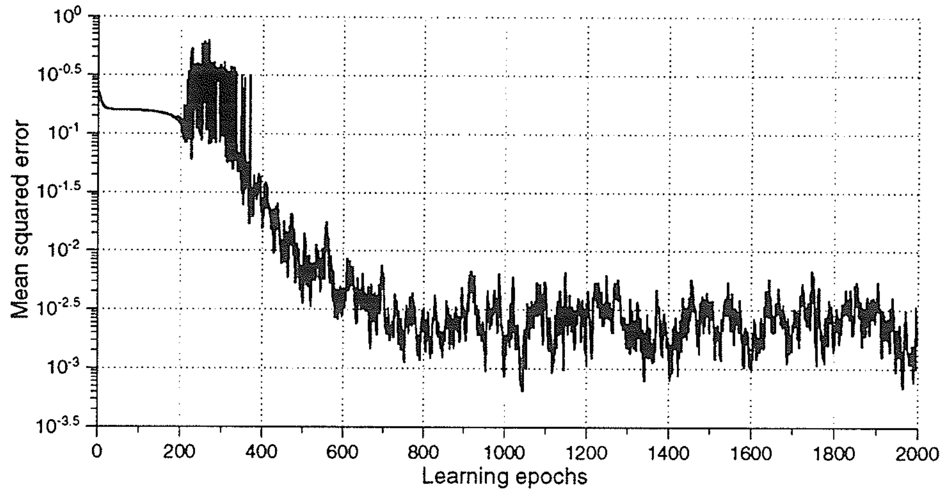


Figure 4.14: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.02$, weight decay $d = 0.0002$, normal annealing. The large error swings are due to basing hopping. Bottom: Free energy through valley between minima, pattern 0. Note the spurious local minimum.



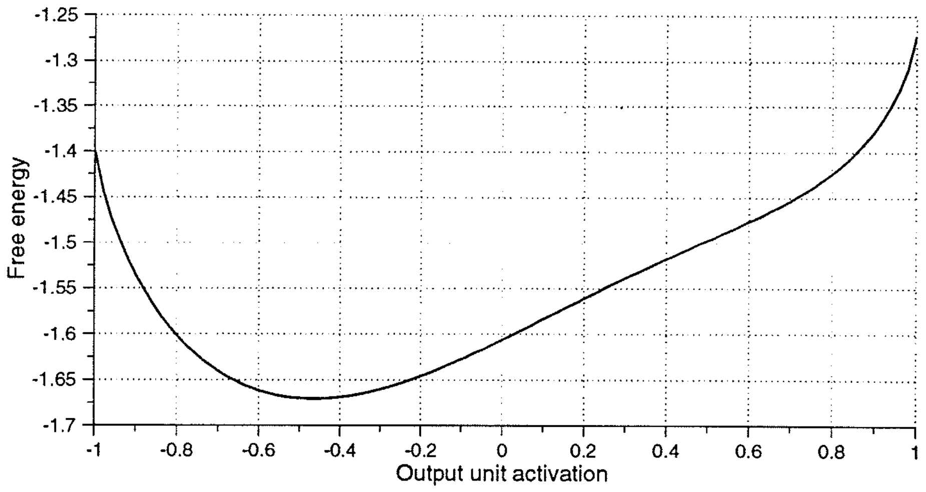


Figure 4.15: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.02$, weight decay ($d = 0.0002$), no annealing, 12 settling passes (Compare with figure 4.14. Note the lack of basin hopping here.)
Bottom: Free energy through valley between minima, pattern 0.

weights like w_b drift toward infinity (or the weight limit), resulting in very good solutions. Unfortunately, once again, this procedure is unstable for larger networks, and it is questionable whether large weights would be beneficial in all problems.

The most useful technique for avoiding the instability problem is to choose the learning rate very carefully. A high learning rate has a beneficial effect because it causes the network to take a large, although random, leap into the stable part of the solution space, where it can then settle to a good, single energy minimum, solution (see figure 4.16). The problem is that the best learning rate depends on the problem being learned as well as the initial weights, and is therefore difficult to determine a priori. Also, our simulations show that the choice of the learning rate becomes more critical in a larger network where the solution space has more dimensions. It may be possible to develop an automatic heuristic technique for adjusting the learning rate, but this has not been explored. High learning rates do not have a beneficial effect on weight decay related problems, other than that they decrease the oscillation frequency. Adjusting the learning rate is therefore an unsatisfactory solution because it is not a *general* solution that works in every case.

Other approaches, including using low-initial-temperature annealing schedules, learning with annealing at first and later disabling it to attempt to remove spurious minima, and other variations have been tried without great success.

4.6 Discussion

The weight drift problem presented here is a fundamental property of the DBM itself. While there are ad hoc fixes in particular situations, a general solution remains elusive. The root of the problem is the existence of spurious local minima, and the inability of the learning algorithm to detect and remove them, even when the network is capable of producing single-minimum energy functions.

An important caveat to the analysis here is that the simple two-input XOR problem is by no means representative of "real" applications. However, it is the simplest problem requiring hidden units, and the problem that led to the demise of the original perceptron

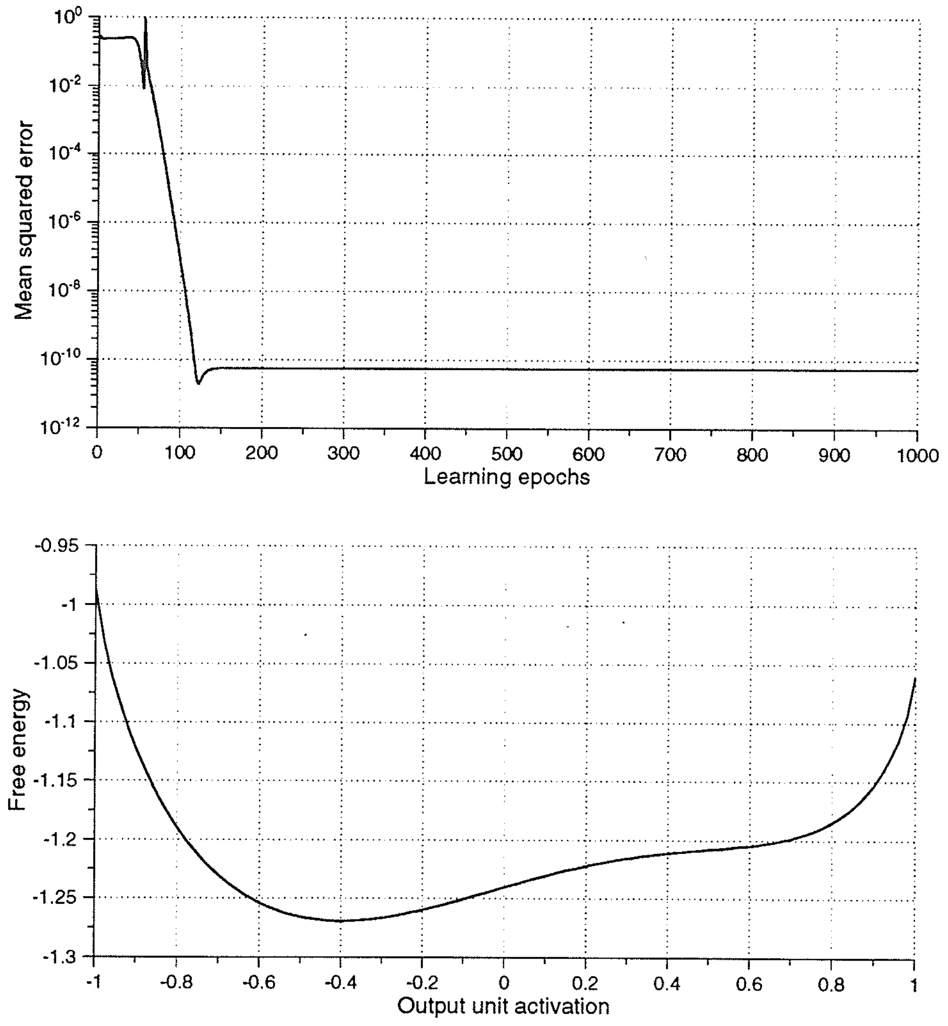


Figure 4.16: Top: Learning test, $2 \times 1 \times 1$ network, $\eta = 0.1$. The high learning rate produces good results quickly in a small network.
Bottom: Free energy through valley between minima.

[MP88]. There are some problems that have isolated solution sets and are immune to the difficulties described here. The 4-2-4 encoder is an example of a problem with a partially isolated solution set. It is used in chapter 6 to demonstrate that the CHL algorithm can deal with weight decay when the output error rises gradually as the weights decay.

There is the hope that "real" problems will tend to have isolated weight sets, unlike XOR and n-bit parity, but it is far from clear whether a system that fails on small problems will succeed on large ones. It is of some significance that almost every problem we have simulated, from the simple XOR, to n-bit parity, to randomly generated mappings, on a wide range of network sizes and connection patterns, has proven to be prone to instability from weight decay.

Unless a better learning algorithm is found, there is always the concern that some small (XOR-like) part of a large network may exhibit the behavior described here. This will cause the entire network to fail to learn its task reliably, or, because of weight decay or other effects, fail some time *after* learning successfully.

Another caveat is the definition of "success" in a learning task. One of the strengths of neural networks is that they are able to deal with self-contradictory input data, where 100% correct learning is by definition not possible. The level of accuracy and reliability in the network outputs required for "success" is then necessarily lower, and the effects of the behavior described here may be entirely submerged in the errors produced by the nature of the problem itself.

The question of whether or not the DBM algorithm functions properly cannot be answered without reference to the implementation of the network and the performance required for a particular application. If the goal is to find a set of weights that allow the network to perform within a certain predefined error criterion, and to terminate the learning process at that point, the DBM algorithm certainly works. On the other hand, if the goal is to build a DBM using analog VLSI hardware, with the requirement that the learning algorithm generate and maintain a stable solution in the presence of weight decay, our analysis shows that the DBM algorithm is not usable. In between these two extremes are the applications where occasional errors can be tolerated, or where the error rate inherent in the problem is large enough to make the weight-drift-related error negligible.

Chapter 5

The Boltzmann Machine Revisited

5.1 Introduction

If it is viewed as a non-linear analog feedback network, the existence of a double-minimum energy function in the deterministic Boltzmann machine analyzed in the previous chapter seems to make sense. However, since the DBM is, in effect, a model of the stochastic Boltzmann machine (BM), the meaning of the second minimum is unclear.

A BM has many local energy minima in its activation space, and moves among them with a probability determined by their relative energies [AHS85]. At thermal equilibrium, there can be only *one* average value for the output of a BM, corresponding to the global minimum in the DBM. What then does the meta-stable local minimum discussed in chapter 4 represent?

One answer is that mean field theory does not hold for small networks, and therefore no analogy is expected. Does this mean that the problem described in chapter 4 disappears in larger networks, where the DBM approximation of the BM becomes more exact? It will be argued here that it does not. While the networks *do* become equivalent as the network size approaches infinity, the behavior of the DBM does not improve, instead, the behavior of the BM *deteriorates.*

5.2 A Small Boltzmann Machine

A stochastic Boltzmann machine can be analyzed in much the same way as the DBM was in the previous chapter. We again use a five-unit $2 \times 1 \times 1$ network (figure 4.4). The problem to be learned is exclusive or, as in the DBM analysis. Because of the analogy between the two types of networks, we make the same weight symmetry assumptions:

$$\begin{aligned}
w_{30} &= w_{03} = w_a \\
w_{31} &= w_{13} = -w_a \\
w_{34} &= w_{43} = w_a \\
w_{20} &= w_{02} = -w_b \\
w_{21} &= w_{12} = w_b \\
w_{24} &= w_{42} = -w_b \\
w_{23} &= w_{32} = w_c
\end{aligned}$$

By performing the above substitutions, the Boltzmann machine energy equation (eq. 2.2) can be rewritten as

$$E = -\left[w_a S_3(S_0 - S_1 + 1) + w_b S_2(-S_0 + S_1 - 1) + w_c S_2 S_3\right]. \tag{5.1}$$

The energy of all sixteen states of the network can be evaluated in terms of w_a, w_b, and w_c. S_0 and S_1 are the states of the inputs, S_2 is the state of the hidden unit, and S_3 is the state of the output.

state $(\alpha\beta\gamma)$	S_0	S_1	S_2	S_3	Energy
0	-1	-1	-1	-1	$E_0 = w_a - w_b - w_c$
1	-1	-1	-1	$+1$	$E_1 = -w_a - w_b + w_c$
2	-1	-1	$+1$	-1	$E_2 = w_a + w_b + w_c$
3	-1	-1	$+1$	$+1$	$E_3 = -w_a + w_b - w_c$
4	-1	$+1$	-1	-1	$E_4 = -w_a + w_b - w_c$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
15	$+1$	$+1$	$+1$	$+1$	$E_{15} = -w_a + w_b - w_c$

(5.2)

From equation (2.6) we know

$$P^{\alpha\beta\gamma} = \frac{e^{-E^{\alpha\beta\gamma}/T}}{Z},$$

where

$$Z = \sum_{\alpha\beta\gamma} e^{-E^{\alpha\beta\gamma}/T}.$$

In order to produce the desired probability distributions on the outputs given the input γ, we need $\langle S_3\rangle^\gamma$ as follows:

$$\begin{aligned}
\langle S_3\rangle^{-1,-1} &= -0.4 \\
\langle S_3\rangle^{-1,+1} &= +0.4 \\
\langle S_3\rangle^{+1,-1} &= +0.4 \\
\langle S_3\rangle^{+1,+1} &= -0.4
\end{aligned}$$

We calculate the expected value $\langle S_3\rangle^\gamma$ for each input state γ as

$$\langle S_3\rangle^\gamma = \sum_{\alpha\beta} S_3 P^{\alpha\beta|\gamma}. \tag{5.3}$$

There are four states $\alpha\beta$ for each input state γ corresponding to the four values of S_2, S_3. Equation (5.3) can be expanded as

$$\langle S_3\rangle^{-1,-1} = \frac{-e^{-E_0} + e^{-E_1} - e^{-E_2} + e^{-E_3}}{e^{-E_0} + e^{-E_1} + e^{-E_2} + e^{-E_3}} \tag{5.4}$$

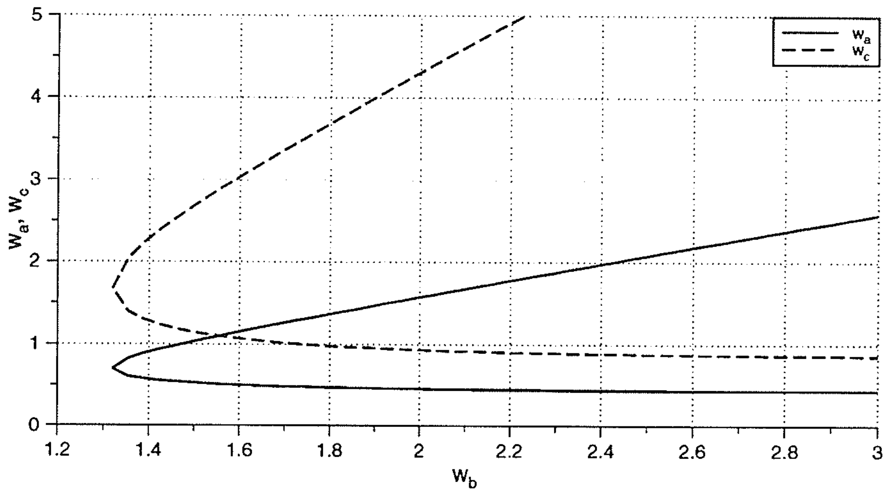


Figure 5.1: Calculated values for weights w_a and w_c for different values of w_b in a one hidden unit, one output unit BM implementing an XOR function with ± 0.4 average outputs.

$$\langle S_3\rangle^{-1,+1} = \frac{-e^{-E_4}+e^{-E_5}-e^{-E_6}+e^{-E_7}}{e^{-E_4}+e^{-E_5}+e^{-E_6}+e^{-E_7}} \tag{5.5}$$

$$\langle S_3\rangle^{+1,-1} = \frac{-e^{-E_8}+e^{-E_9}-e^{-E_{10}}+e^{-E_{11}}}{e^{-E_8}+e^{-E_9}+e^{-E_{10}}+e^{-E_{11}}} \tag{5.6}$$

$$\langle S_3\rangle^{+1,+1} = \frac{-e^{-E_{12}}+e^{-E_{13}}-e^{-E_{14}}+e^{-E_{15}}}{e^{-E_{12}}+e^{-E_{13}}+e^{-E_{14}}+e^{-E_{15}}} \tag{5.7}$$

Substituting for the energies from (5.2), we find that the expressions for $\langle S_3\rangle^{-1,-1}$ and $\langle S_3\rangle^{+1,+1}$ are identical and that $\langle S_3\rangle^{-1,+1} = -\langle S_3\rangle^{+1,-1}$. We therefore have two equations in the three unknowns w_a, w_b, and w_c. Solving equations (5.6) and (5.7) numerically with w_b as the independent variable, as we did for the DBM, we get the results plotted in figure 5.1.

As with the DBM, there is an entire range of solutions that produce exactly the same (time-averaged) output values. Unlike the DBM, however, the solutions do not extend to $w_b = 0$, and there is no degradation in reliability when moving from one solution to another.

The behavior of the BM is different from the equivalent DBM because the mean field approximation does not hold for small numbers of units. This is not surprising, since the

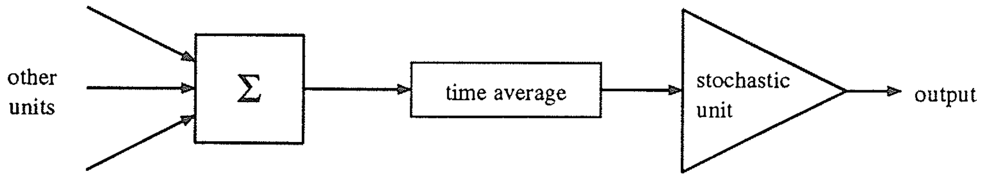


Figure 5.2: A unit with a time-averaging function on its input. The time averaging function could simply be a low-pass filter

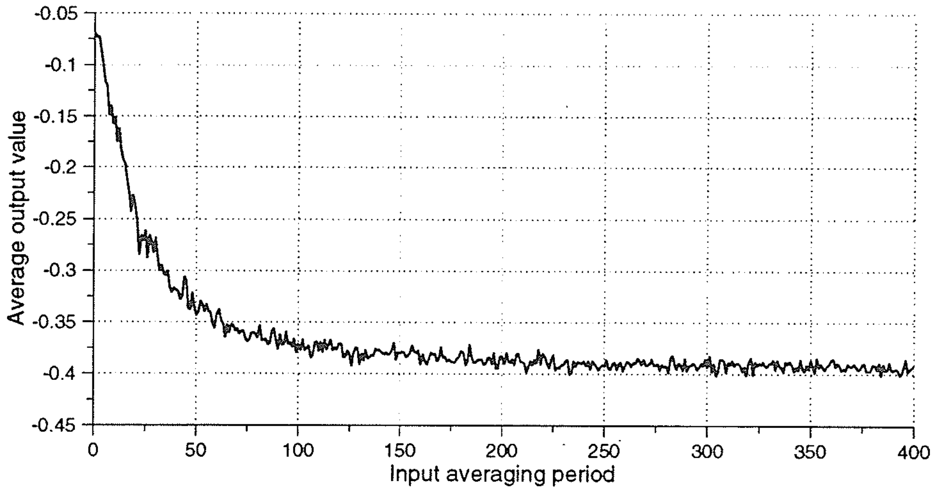


Figure 5.3: Time averaged output of a BM network with weights $w_a = 0.488$, $w_b = 0.750$, $w_c = 1.098$ (a fairly "good" DBM weight set). The output is averaged over 200 000 cycles.

probability of setting S_i to ± 1 in a BM depends on the *instantaneous* activation of every other unit in the network for a BM, and the *expected value* of the activation in a DBM.

Consider a network with units like the one shown in figure 5.2. This is a standard BM network with a time averaging function on the input of each unit. Figure 5.3 shows the average output of a BM with weights set to values calculated for a DBM according to equations (4.11) and (4.12) for different input averaging periods. Notice that the average output approaches the desired -0.4 value as the averaging period increases.

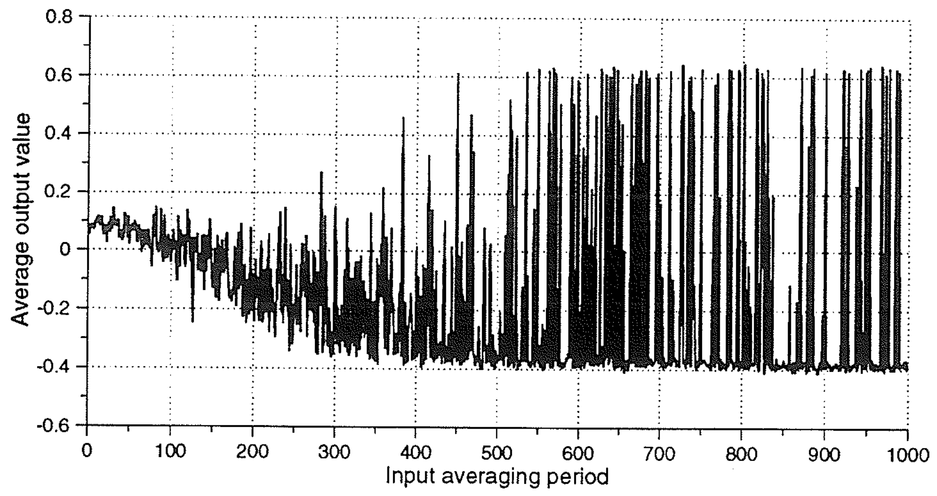


Figure 5.4: Time averaged output of a BM network with weights $w_a = 0.383$, $w_b = 0.400$, $w_c = 1.157$ (A less "good" weight set than in figure 5.3). The output is averaged over 200 000 cycles. Notice the unreliability of the network output.

Comparing figures 5.3 and 5.4, we see that the network output becomes less reliable for a less "good" set of weights (in the sense of section 4.3.1). In figure 5.4, the output sometimes settles into the local positive minimum instead of the global negative one.

As the input averaging period is increased, the time-averaged BM output becomes the same as that of a DBM with the same weights. This occurs because the DBM activation function is identical to the expected value of the BM activation. After all, this is how the DBM was derived. When the net input to each unit is averaged over a long enough period of time, it approximates the expected value, and the networks are equivalent.

5.3 Expanding the Network

There is an alternate way to generate an approximation to an expected value for the inputs in a BM. Consider the simple one-hidden, one-output unit network used in the previous analysis (page 70). Expanding equation (2.1) for the output unit, we get the net input to the output unit:

$$net_3 = w_a S_0 - w_a S_1 + w_a + w_c S_2.$$

Now replace the hidden unit (unit 2) with n hidden units, and replace the output unit (unit 3) with n output units. All the weights from the inputs and bias unit are replicated, and the hidden and output units are interconnected with weight $w_c' = w_c/n$. Each hidden unit is connected to each output unit and vise versa, but neither the hidden nor the output unit sets are internally connected (see figure 5.5 and figure 5.6). The net input to each output unit i is then

$$\begin{aligned} net_i &= w_a S_0 - w_a S_1 + w_a + \sum_{j \in \text{hidden}} \frac{w_c}{n} S_j \\ &= w_a S_0 - w_a S_1 + w_a + w_c \frac{\sum_j S_j}{n}. \end{aligned}$$

The contribution of the hidden unit to the net input has been averaged over n identical but stochastically varying units. The input to the hidden units similarly includes the average of the output units. Instead of the average over time in the previous analysis, we now have an average over many units at each point in time. The averages are approximations of the expected values in the original one-hidden, one-output unit network:

$$\begin{aligned} \frac{\sum_{j \in \text{hid}} S_j}{n} &\approx \langle S_2 \rangle \\ \frac{\sum_{i \in \text{out}} S_i}{n} &\approx \langle S_3 \rangle. \end{aligned}$$

We can see the effect of changing the network size by varying n, the number of hidden and output units, and taking the network output to be the average over the n output units. The results are shown in figure 5.7. Notice that the average of the network outputs appears to oscillate between two stable values: -0.4 and $+0.65$. These correspond to the global and local minima of the DBM energy function for the set of weights in this example (see figure 5.8). Figure 5.9 verifies that the time average of a single output unit is a close approximation to the instantaneous average over all the output units.

As the network becomes larger, the frequency of the oscillations decreases. Recall that this is stochastic Boltzmann machine with a deterministic Boltzmann machine weight set.

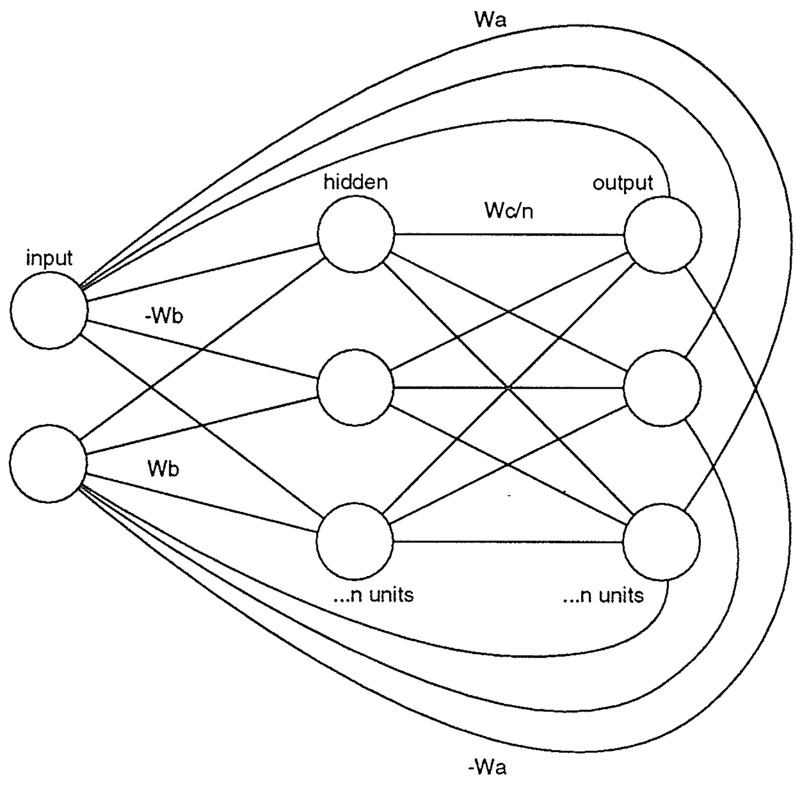


Figure 5.5: A Boltzmann machine with the hidden and output units each replicated n times. In this diagram, the circles represent the units and the lines represent the weights. There are no interconnections between the hidden units or between the output units. The bias unit has been omitted for clarity.

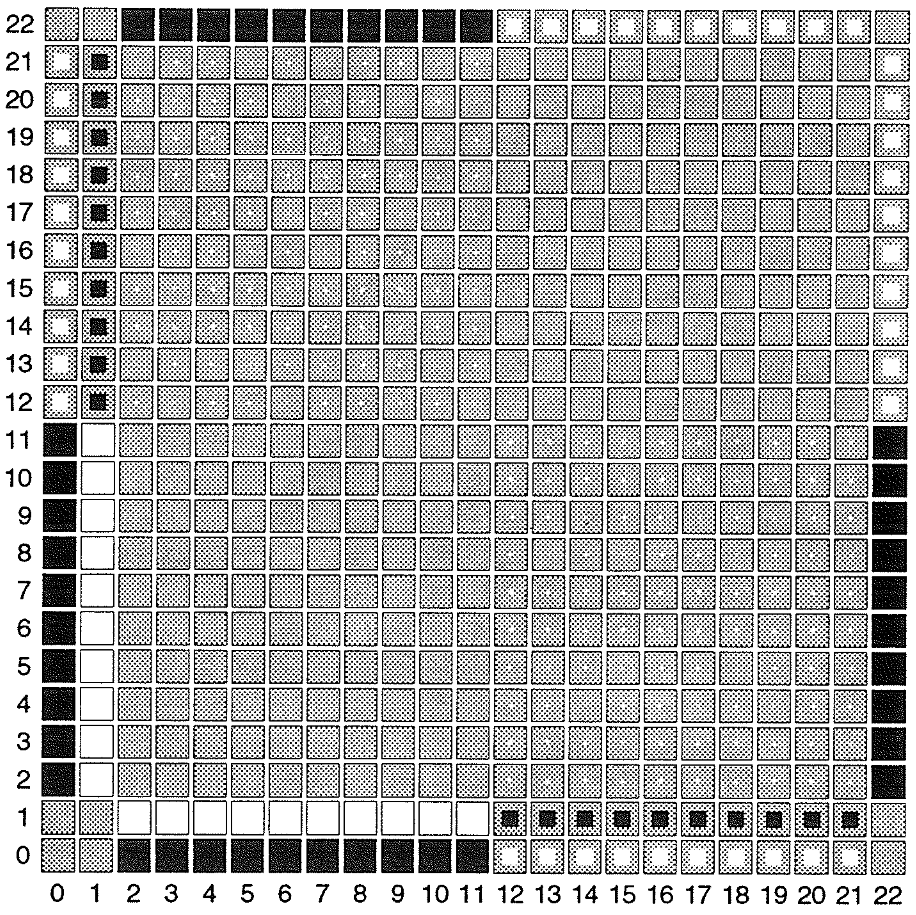


Figure 5.6: Weight set of an expanded Boltzmann machine with $n = 10$. Units 0 and 1 are inputs, units 2 to 11 are hidden, units 12 to 21 are output, and unit 22 is the bias unit.

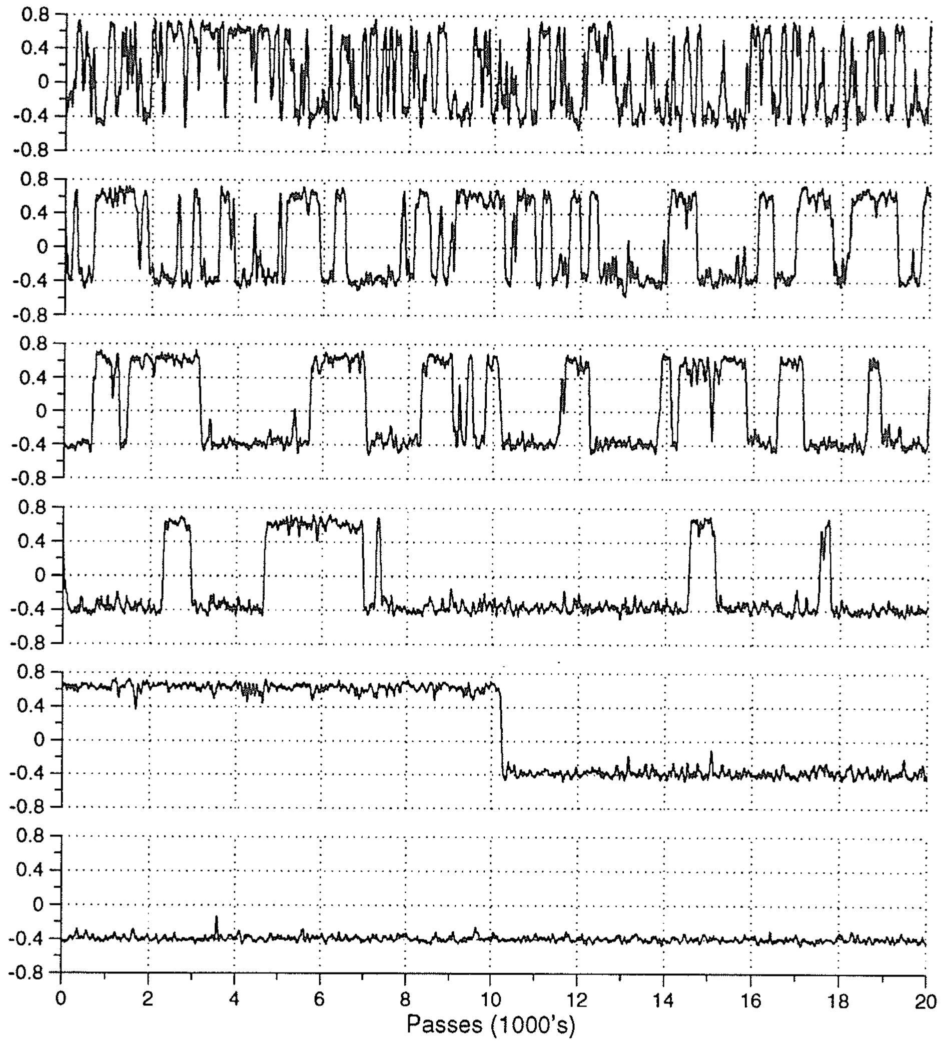


Figure 5.7: Output of a Boltzmann machine with replicated hidden and output units. Output values are averaged over all output units and then smoothed by averaging over 50 time steps. Network sizes are, from top to bottom, $n = 50, 100, 150, 200, 250, 500$. Weights are $w_a = 0.327$, $w_b = 0.330$, $w_c = 1.143$.

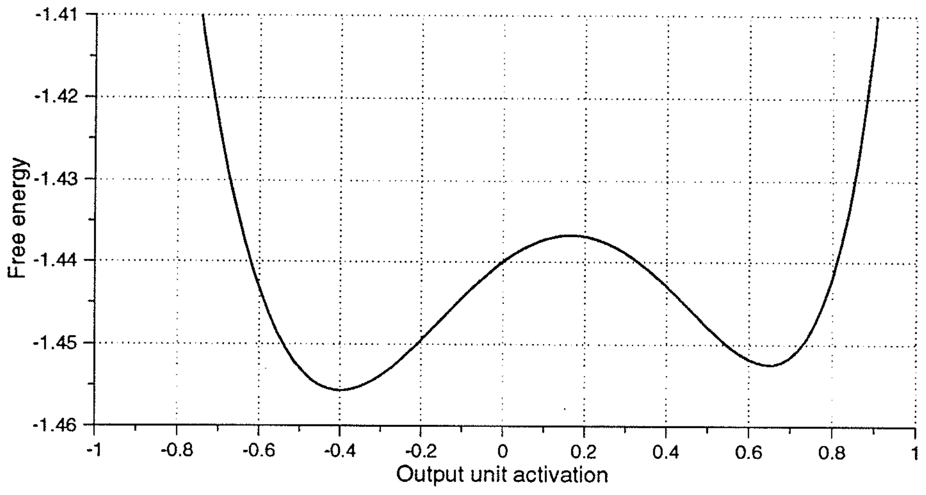


Figure 5.8: Cross section of energy minimum valley of DBM network with $w_a = 0.327$, $w_b = 0.330$, $w_c = 1.143$. (The same weight values used to generate figure 5.7)

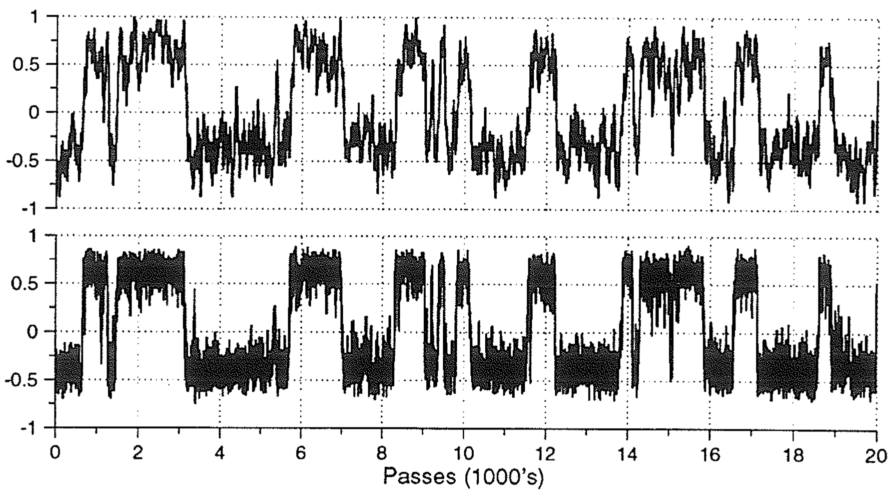


Figure 5.9: Comparison of output of a single unit averaged over time and the instantaneous average of all output units in an $n = 150$ network. The top trace is the activation of a single output unit, averaged over 50 passes. The bottom trace is the average over all the output units with no smoothing.

The average output value should be near zero in all cases. All 2^{2n} states of the network are possible, but those sets of states resulting in the two dominant average output values are strongly favored, and the network spends long periods of time in one or the other of these two sets of states, with relatively rapid transitions between them.

In order to see the true near-zero time-average output of the network, it becomes necessary to observe it for increasingly longer periods of time as the number of units is increased. If the observation (statistics gathering) period is too short, we will get a false impression of the actual behavior of the network. For example, if the $n = 200$ case in figure 5.7 is observed for the first 2000 update passes, the conclusion is that the average output value is -0.4. If weight updates during learning are based on such false information, and -0.4 is indeed the value the network is supposed to learn, then the weights are not changed when they should be. In effect, the other (meta-stable) state is not observed in this case because the observation period is truncated prematurely.

Thus, the local minimum in the DBM energy function results from the second, partially stable activation set in a large BM. Put another way, the second energy minimum in the DBM is analogous to observing the BM network for too short a period of time.

The DBM activations represent the expected values of the BM activations. The only way to achieve a *perfect* calculation of the expected value from observation is to average over an *infinite* number of units. The DBM therefore, is equivalent to the BM of figure 5.5 with $n = \infty$. To avoid the problem of local minima in the DBM energy function thus requires observation of the BM over an infinite period of time, which is impossible. In other words, the local minimum problem is a fundamental characteristic of the DBM, not an artifact of the mean field approximation applied to a network that is too small, and we cannot expect improvement simply from increased network size.

5.4 Discussion

The simple XOR problem explored here is quite artificial, as is the method used to expand the size of the BM. Whether the same behavior predicted and simulated here would be encountered in a "real" DBM problem, where the network actually learns the weight set on its own, is a matter for debate, although our simulations have not been promising. In a real (finite) BM, it is always possible to avoid the local minimum problem by collecting statistics over a sufficiently long period of time. Judging from the simulations performed here (figure 5.7), that period increases rapidly with increasing network size.

Using a short statistics gathering period during learning causes the network to adjust the weights according to incorrect statistics. If a learning step is based on statistics gathered while the network is in the spurious set of states, the resulting large weight adjustment decreases the likelihood of that set of states. This is analogous to disabling annealing during learning in a DBM and allowing the network to occasionally settle in a local energy minimum. DBM simulations have shown that while disabling annealing is effective in a small network, it results in unstable behavior in a larger network with a more complex learning task.

There are two conclusions to be drawn from this analysis:

1. It is important to ensure that a sufficiently long statistics gathering period is used as the size of a Boltzmann machine increases.

2. Simply increasing the size of a DBM does not ameliorate the problem of spurious minima and unstable learning presented in chapter 4.

Chapter 6

Hardware Issues

6.1 Introduction

One of the most inviting features of deterministic Boltzmann machines is the possibility of implementing their simple, local learning rule in a massively parallel architecture. However, as a network with N units contains N^2 weights, the weights and associated circuitry (the synapses) must necessarily be small and simple, and may consequently exhibit behavior far removed from ideal mathematical models.

In digital implementations, the number of bits used to store the weights is limited. Because digital multipliers will tend to be fixed rather than floating point, both the resolution and the dynamic range of the weights is severely restricted. In an analog system, the weight resolution is limited by noise and parasitic effects, and weight values stored as voltages on capacitors are subject to decay over time. In addition, analog multipliers saturate and have zero offset problems.

One of the features of neural networks is that they not only learn to represent some features of their environment, but are also able to learn to compensate for internal problems. In this chapter, we explore which non-ideal analog hardware characteristics are automatically dealt with by the learning algorithm and which seriously degrade network performance.

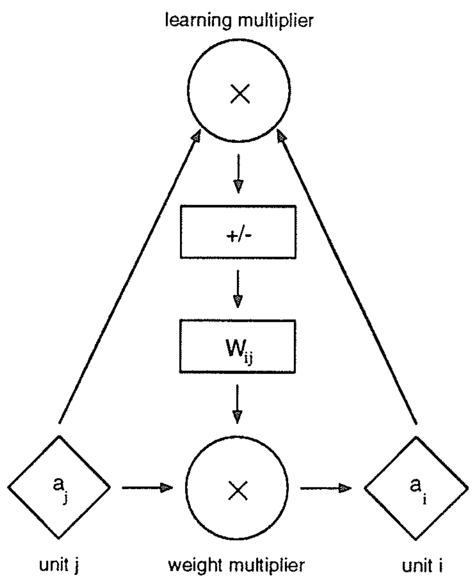


Figure 6.1: A single analog synapse consists of two multipliers, weight storage, and a weight add/subtract circuit.

The goal is to focus hardware design efforts on the important issues and to let the learning algorithm deal with the rest.

6.2 Non-ideal analog behavior

Figure 6.1 shows the components of an analog synapse. Ideally, the multipliers would perform true multiplication operations, the weight storage would hold any value with infinite precision for an infinitely long time, and the add/subtract circuit would update the weight values accurately by any desired amount. The only non-linear component in the network would be the logistic unit activation function itself.

In reality, nothing is linear, and saturation and zero offsets are prevalent in all components. We now explore a number of sources of error and their effect on the performance of the network.

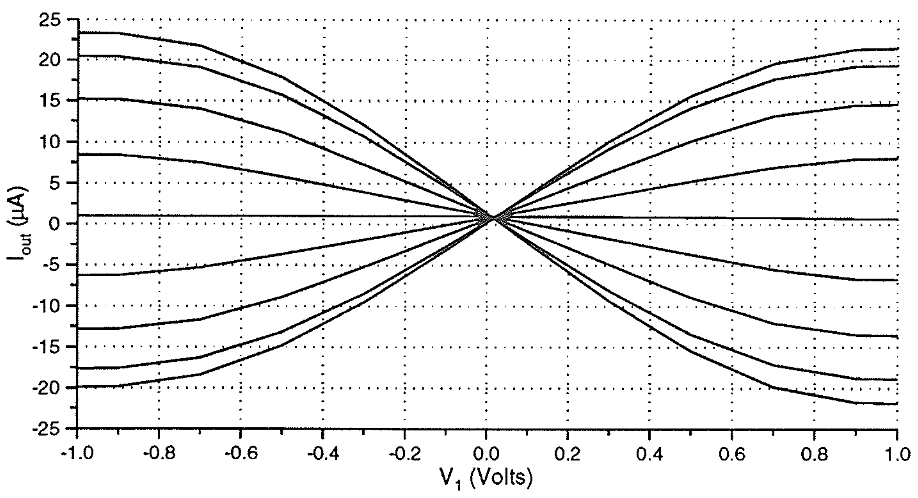


Figure 6.2: Measured values from a CMOS implementation of a Gilbert multiplier [Sch91]. Each trace represents an input (V_2) step of 0.2V from $V_2 = -0.8$V to $V_2 = +0.8$V.

6.2.1 Multipliers

The Gilbert multiplier is a convenient, compact circuit for multiplying two analog values [Mea88]. Figure 6.2 shows the measured characteristic of a real Gilbert multiplier [Sch91], shown schematically in figure 6.3. Three features are immediately obvious:

1. The multiplier characteristic saturates in both inputs and is substantially non-linear any significant distance from zero.

2. There is a zero offset; that is, the output is not zero when one or both of the inputs are zero. Also, the sign of the output may be wrong for small input values.

3. The characteristic is unbalanced in that the $V_2 = +0.8$ and $V_2 = -0.8$ traces are not exact mirror images of each other. This is not the same as the offset problem.

For simulation purposes, the multiplier was modeled by the function

$$\mathrm{mult}(V_1, V_2) = \phi \tanh(\theta V_1 - O_\theta) \tanh(\lambda V_2 - O_\lambda) + O_y \tag{6.1}$$

where ϕ adjusts the function range, θ and λ adjust the degree of nonlinearity in the two inputs, and O_θ, O_λ, and O_y adjust the offsets.

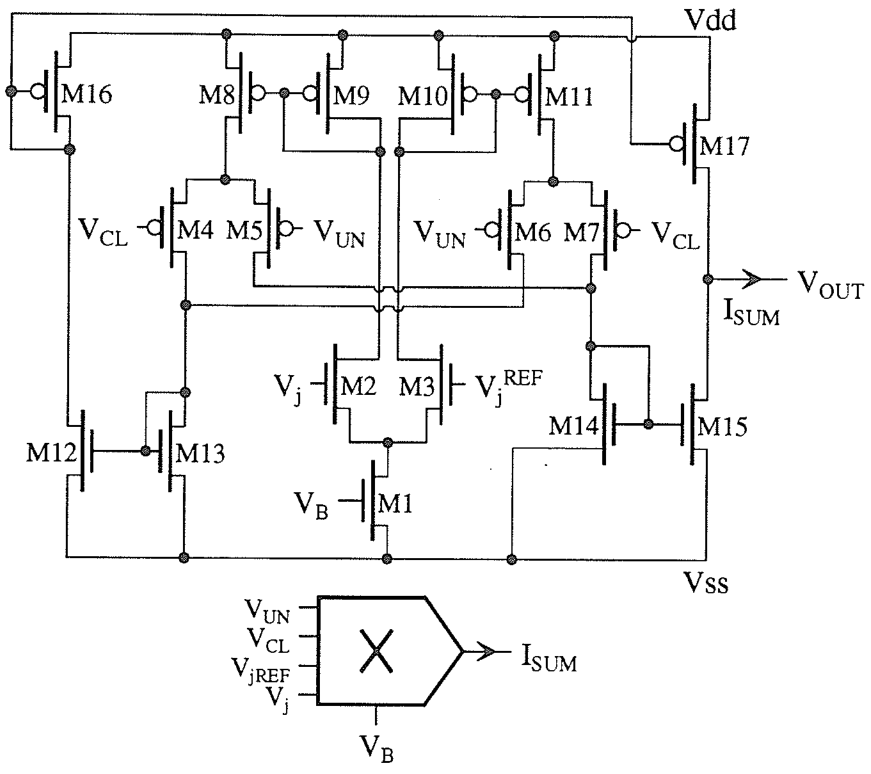


Figure 6.3: Schematic circuit diagram of an analog CMOS Gilbert multiplier (taken from [Sch91]). A Gilbert multiplier multiplies the differences of two pairs of voltages. $V_1 = V_{CL} - V_{UN}$ and $V_2 = V_j - V_{jREF}$.

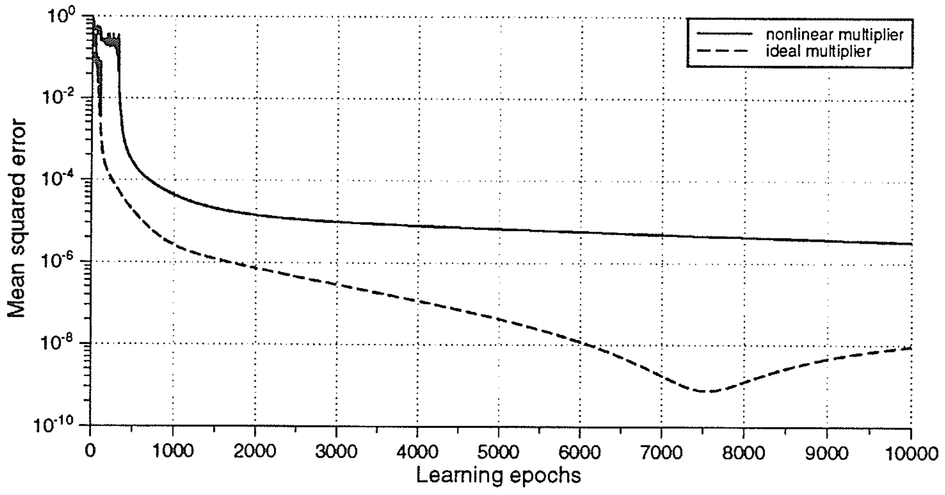


Figure 6.4: Learning results for a $4 \times 5 \times 1$ network learning 4-bit parity using the model in equation (6.1) for the weight multiplier with the model parameters set to match the measured multiplier characteristic in figure 6.2. The error values in both traces are acceptably small.

Non-ideal Weight Multiplier

Figure 6.4 shows the learning behavior of a $4 \times 5 \times 1$ network on a 4-bit parity problem where the weight multiplier has been replaced by the model in equation (6.1) with parameters set to match the measured multiplier characteristic in figure 6.2. The parameters are shown below. Both the original value of each parameter and its scaled value (for use in simulation) are shown.

Parameter	Value	Sim. value	Calculation
ϕ	29×10^{-6}	2.8	calculated so the mult. 2×1 gives 2
θ	1.7	0.68	weight range -2 to $+2$ maps to ± 0.8 V
λ	1.35	1.08	act. range -1 to $+1$ maps to ± 0.8 V
O_θ	0.017	0.0068	
O_λ	0.017	0.0136	
O_y	-1×10^{-6}	-0.1	

Performance is not seriously degraded by the non-ideal multiplier characteristic, but learning slows down. Other tests using more severely nonlinear multiplications show that most of the degradation that does occur is due to a change in the effective learning rate, and that the learning rate η can be adjusted to counteract this effect.

It is not hard to see why the network can deal with the non-ideal behavior of the weight multiplier. The effect of the non-linearity is that net_i, the input to unit i, is no longer the sum of the $w_{ij}a_j$ terms, but the sum of a 'squashed' version of these.

The process of learning can be viewed as adjusting the equilibrium activations of the units by modifying the weights connected to their inputs. To see how the effective learning rate is changed by the nonlinear multiplier, consider the effect of a weight change dw_{ij} on $\breve{a}_i$, the equilibrium activation of unit i. For ideal multipliers, at equilibrium

$$\breve{a}_i = f_i(net_i),$$

$$net_i = \sum_k w_{ik}\breve{a}_k, \tag{6.2}$$

and

$$\frac{\partial \breve{a}_i}{\partial w_{ij}} = f'(net_i)\breve{a}_j. \tag{6.3}$$

In the nonlinear case (ignoring offsets because they don't affect the learning rate),

$$\breve{a}_i^* = f_i(net_i^*),$$

$$net_i^* = \sum_k \phi \tanh(\theta w_{ik}) \tanh(\lambda \breve{a}_k),$$

and

$$\frac{\partial \breve{a}_i^*}{\partial w_{ij}} = f'(net_i^*)\frac{\phi\theta \tanh(\lambda \breve{a}_j)}{\cosh^2(\theta w_{ij})}. \tag{6.4}$$

If the units are producing the same outputs in the ideal and nonlinear cases, then their inputs are also the same and we can neglect the difference between net_i and net_i^*. Equations (6.3) and (6.4) then differ by the factor

$$\frac{\phi\theta \tanh(\lambda \breve{a}_j)}{\cosh^2(\theta w_{ij})\breve{a}_j} \tag{6.5}$$

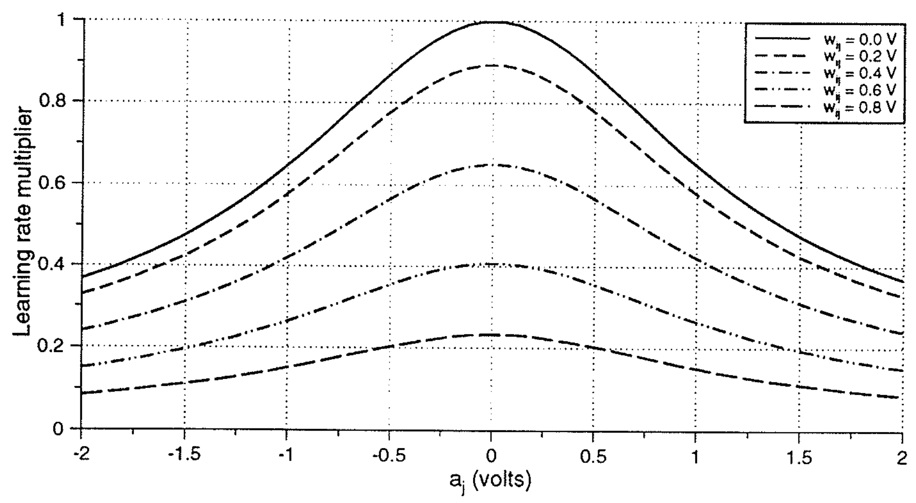


Figure 6.5: Effective learning rate multiplier (equation 6.5) with the model parameters set to match the measured multiplier characteristic in figure 6.2. Note that the activations and weights are expressed as voltages, and that the maximum learning rate factor of one is arbitrary.

which reaches a maximum value of $\phi\theta\lambda$ at $w_{ij} = 0$, $\breve{a}_j = 0$ and decays to zero as the activation $\breve{a}_j$ or weight w_{ij} increase in magnitude (see figure 6.5). The effective learning rate, determined by the anticipated change in $\breve{a}_i$ when w_{ij} changes, therefore varies with both the weight and activation values. While this is undesirable, the effect can be dealt with by choosing the voltage and current ranges in the network so that the multipliers stay in their relatively linear central region, and adjusting η to achieve the desired average effective learning rate.

The zero offset in the multiplier does not cause a problem because its effect is merely the addition of a constant term to the net_i summation in equation (6.2). This is mathematically identical to the effect of the bias unit, so the learning rule adjusts the weights connecting the bias unit to the other units to compensate for the offset.

The unbalanced nature of the multiplier was not modeled, but is not expected to have much of an effect since it simply changes the effective learning rate slightly as a function of the activation a_j and the weight w_{ij}.

It is evident that deficiencies in the weight multipliers do not have much impact on the performance of a DBM. This is not surprising since the learning procedure adjusts activations by adjusting the net inputs to the units, automatically compensating for offsets and nonlinearities in the weight multipliers. As long as the weight multipliers are monotonic, weight changes are always in a direction that decreases error, and the network functions properly.

Non-ideal Learning Multiplier

The function of the learning multiplier is to calculate the product of the activations $\breve{a}_i$ and $\breve{a}_j$ so that the weight w_{ij} can be updated according to

$$\Delta w_{ij} = \eta(\breve{a}_i^+ \breve{a}_j^+ - \breve{a}_i^- \breve{a}_j^-). \tag{6.6}$$

The learning multiplier suffers from the same three problems as the weight multiplier, but the effects on network performance are different. Using the model in (6.1), equation (6.6) becomes

$$\begin{aligned}\Delta w_{ij} &= \eta\Big((\phi\tanh(\theta\breve{a}_i^+ - O_\theta)\tanh(\lambda\breve{a}_j^+ - O_\lambda) + O_y) \\ &\quad - (\phi\tanh(\theta\breve{a}_i^- - O_\theta)\tanh(\lambda\breve{a}_j^- - O_\lambda) + O_y)\Big) \\ &= \eta\phi\left(\tanh(\theta\breve{a}_i^+ - O_\theta)\tanh(\lambda\breve{a}_j^+ - O_\lambda) - \tanh(\theta\breve{a}_i^- - O_\theta)\tanh(\lambda\breve{a}_j^- - O_\lambda)\right).\end{aligned}$$

The offset O_y is not a problem as it is canceled by the subtraction,[1] and ϕ is absorbed into the learning rate. The tanh(·) functions distort the size of the weight adjustment steps. Since the functions are monotonic, the sign of Δw_{ij} is the same as in the ideal case and the major effect is that learning progresses at a different rate.

[1]It is assumed that there is only one learning multiplier per weight. If there are separate learning multipliers for the clamped and unclamped phases, then the offsets do not cancel, and the effect is the same as an offset in the add/subtract unit (discussed later).

6.2.2 Add/Subtract Unit

The add/subtract unit is responsible for calculating the weight change Δw_{ij} from the difference of the products of the clamped and unclamped activations and adding it to the weight w_{ij}. Typically, analog weights are stored as charge on a capacitor, so the add/subtract unit either adds or removes charge to or from the capacitor. (In reality, part of the function of the add/subtract unit can be performed by the Gilbert multiplier circuit.)

It is notoriously difficult to make an ideal analog add/subtract unit, as the amount of charge added to or removed from the capacitor should be independent of the current weight value (capacitor voltage). This means the add/subtract unit must be an ideal current source. Because it is not, weight steps towards zero are larger than steps away from zero. It is easier to build a circuit which adds or subtracts a constant amount of charge to or from the capacitor, depending only on the sign of $\breve{a}_i^+\breve{a}_j^+ - \breve{a}_i^-\breve{a}_j^-$. This is called *Manhattan learning* [PH89, Sch91]. In general, the exact size of the weight steps is not of great importance so long as the steps are small.

A common deficiency of the add/subtract unit is the addition of a small offset to the result. Consider the weight updating rule (equation 6.6) where there is an offset of δ in the subtraction operation:

$$\Delta w_{ij} = \eta(\breve{a}_i^+\breve{a}_j^+ - \breve{a}_i^-\breve{a}_j^- + \delta_{ij}).$$

Without the offset, learning is complete (weight changes stop) when the clamped and unclamped activations match ($\breve{a}_i^+\breve{a}_j^+ = \breve{a}_i^-\breve{a}_j^-$) and $\Delta w_{ij} = 0$. Now, however,

$$\Delta w_{ij} = 0 \quad \text{when} \quad \breve{a}_i^+\breve{a}_j^+ + \delta_{ij} = \breve{a}_i^-\breve{a}_j^-.$$

Assume unit j is an input, which is always clamped ($\breve{a}_j^+ = \breve{a}_j^- = \breve{a}_j$), and unit i is an output. Then, after learning is complete,

$$\begin{aligned} \breve{a}_i^- &= \frac{\breve{a}_j\breve{a}_i^+ + \delta_{ij}}{\breve{a}_j} \\ &= \breve{a}_i^+ + \frac{\delta_{ij}}{\breve{a}_j}. \end{aligned} \qquad (6.7)$$

We would therefore expect an error of $\delta_{ij}/\breve{a}_j$ in the output $\breve{a}_i^-$ (the output during recall) when learning is complete. However, consider the effect of another input unit, k, with $\delta_{ik} = 0$:

$$\Delta w_{ik} = 0 \text{ when } \breve{a}_i^+ \breve{a}_k^+ = \breve{a}_i^- \breve{a}_k^-$$

or, since $\breve{a}_k^+ = \breve{a}_k^- = \breve{a}_k$, the final value after learning is

$$\breve{a}_i^- = \breve{a}_i^+ \tag{6.8}$$

which contradicts (6.7).

Does this mean that the weight values oscillate up and down while attempting to satisfy both (6.7) and (6.8)? Unfortunately not. Weight w_{ij} appears only in equation (6.7), while w_{ik} appears only in equation (6.8), so the two weights do not directly compete with each other. Instead, both w_{ij} and w_{ik} continuously increase in magnitude, one positively and one negatively, to infinity (or saturation) in an attempt to cancel each other's effects on $\breve{a}_i$. In simulation, it is typically weights from the hidden units that counteract the δ_{ij} mismatch.

The weight saturation effect is not really a problem as long as the network is performing correctly. Rather, it is a symptom of weight drift in a continuous solution set. The CHL algorithm counteracts any moderate weight drift caused by an offset in the add/subtract circuit as long as there is a gradual degradation in performance with changing weights ($\mathbf{W}^*$ is isolated) If the solution set is continuous, however, we can expect saturated weights, basin hopping, and oscillation (see figures 6.6 and 6.7).

Typically, each add/subtract unit attached to each weight in the network has a random mismatch due to variation among devices, superimposed on a global mismatch due to fabrication process variations. This does nothing to improve the situation. Because of these variations, it is impossible to build a *perfectly* balanced analog add/subtract unit, and *any* offset, no matter how small, eventually leads to oscillation in a continuous weight set.

It beneficial to introduce a learning threshold in some cases. This prevents any weight update if $\Delta w_{ij} < \Delta w_{TH}$, where Δw_{TH} is a threshold value chosen to be somewhat larger than the largest δ_{ij}. This threshold prevents the slow but continuous weight drift due to an add/subtract unit offset, at the cost of a somewhat higher residual error (see figure 6.8).

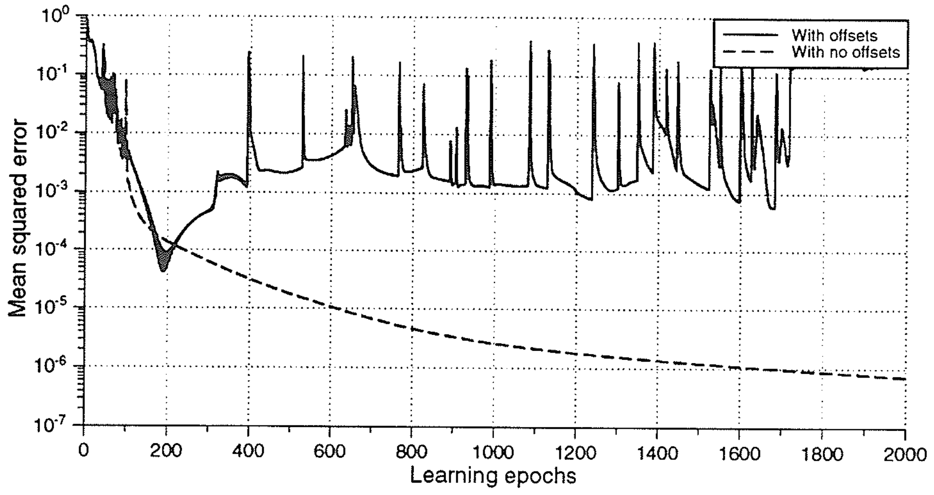


Figure 6.6: Training results with random add/subtract offsets in the range -0.001 to 0.001, learning rate $\eta = 0.1$, weights limited to ± 2.0. Notice the slower learning and oscillation in the trial with offsets.

6.2.3 Weight Decay

If weight values are stored as a voltage on a capacitor, charge leaks to ground at a rate proportional to the voltage on each capacitor, leading to an exponential decay of the weights towards zero.[2] In a CMOS implementation, the leakage is towards the substrate voltage – the most negative voltage on the chip. Thus the weights tend to drift towards their most negative value.

The effect of weight decay is in many ways similar to the effect of the offset in the add/subtract unit described above. One difference is that whereas the offset can be in any direction, decay is always towards the origin or towards some other single value. Again, if the solution set is isolated, the learning algorithm compensates for moderate decay rates, and the effect is merely an increase in the residual error.

Figure 6.9 shows the effect of weight decay in a 4-2-4 encoder. The 4-2-4 encoder is interesting because, unlike the XOR and parity problems, the restricted architecture of the encoder allows the learning algorithm to compensate for weight decay. In this case,

[2]This assumes a ground-referenced weight value, but the argument holds in any case.

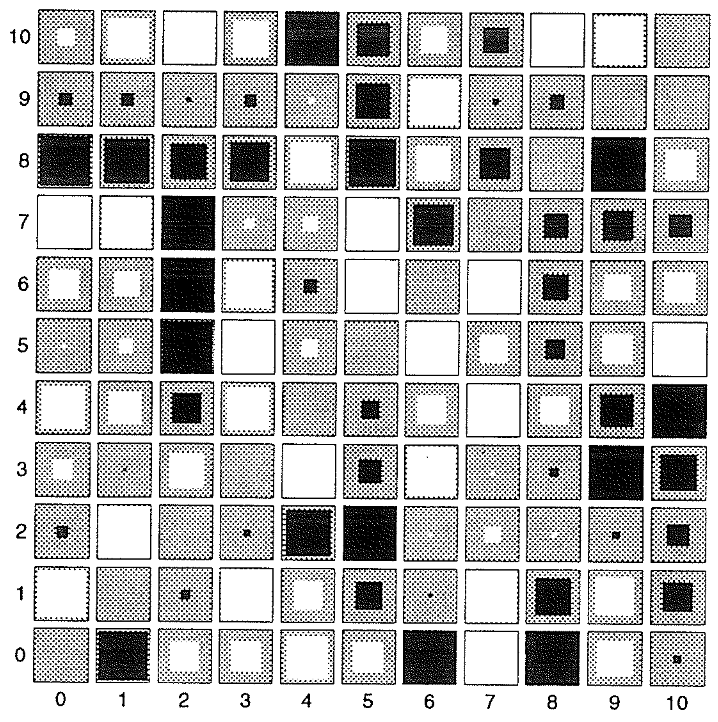


Figure 6.7: Weight values for the same network as figure 6.6. The horizontal axis is 'From', the vertical is 'To'. The first four units are inputs, followed by 5 hidden units, one output unit, and the bias unit. Note the small (useful) weight values connecting to the output (row 9).

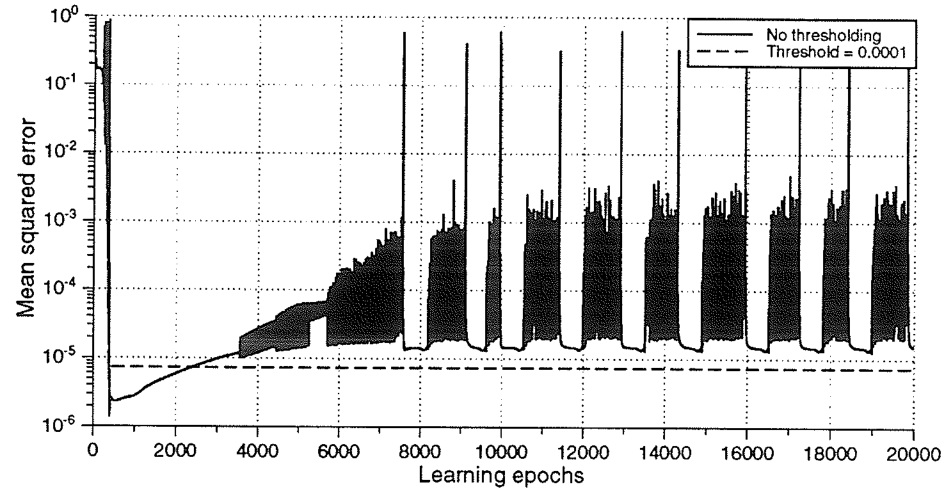


Figure 6.8: The effect of a learning threshold on a network with add/subtract unit offsets. Because learning is disabled if the weight change is below the threshold, the effect of the offset (eventual oscillation) is prevented at the cost of some residual error. For this trial, offset varies randomly between ± 0.002, threshold is 0.0001.

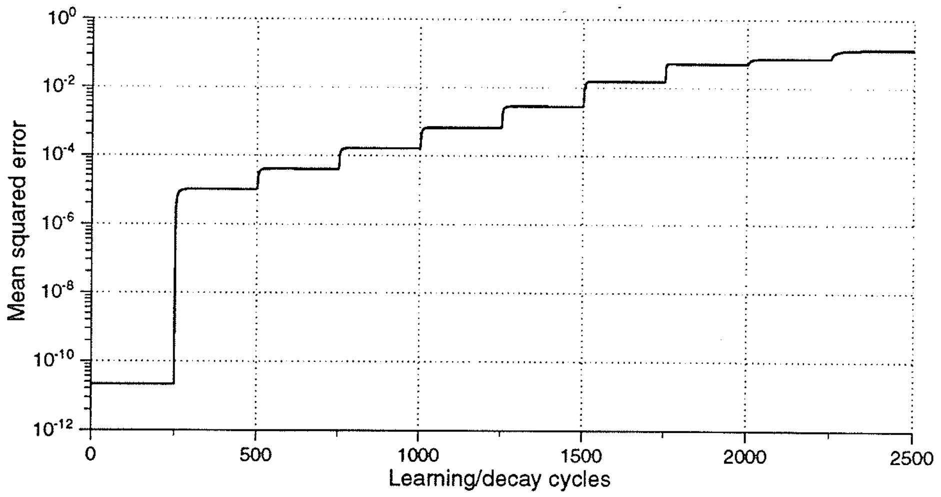


Figure 6.9: Effects of various levels of weight decay on a 4-2-4 encoder problem. Each step represents an increased level of weight decay, starting at $d = 0.0001$, followed by $d = 0.0002$, $d = 0.0004$, etc. up to $d = 0.0256$. The learning algorithm is able to handle decay rates up to about $d = 0.0064$, which represents about 13% of the $\eta = 0.05$ learning rate.

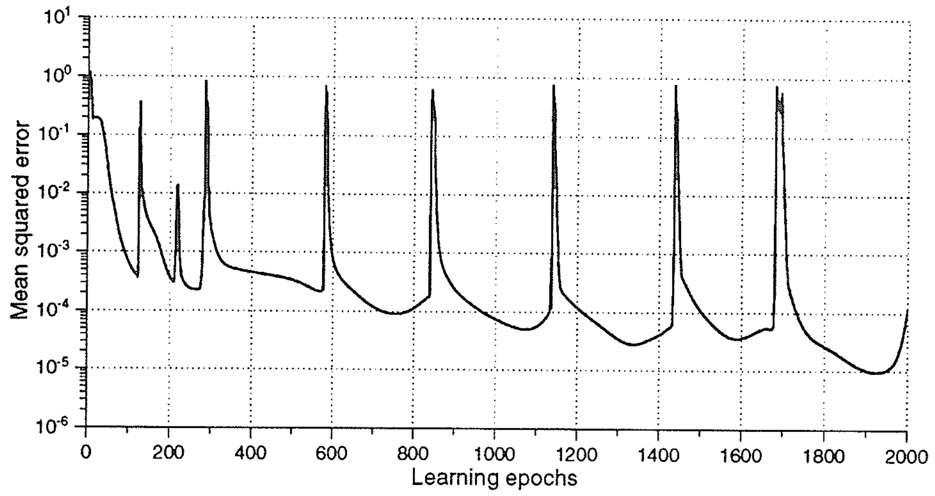


Figure 6.10: The effect of weight decay on learning performance for a $4 \times 5 \times 1$ network learning a 4-bit parity problem. For this test, $\eta = 0.1$, $d = 1 \times 10^{-4}$ (decay is 0.1% of the learning rate).

the solution set is not isolated, but it is bounded below (towards the origin). This makes the encoder immune to weight decay but not to add/subtract unit offsets. If the solution set is continuous and unbounded, weight decay causes oscillation to occur (see figures 6.10 and 6.11).

The oscillation procedure in a continuous solution set is as follows:

1. The weights start at some learned value and decay towards the origin with the CHL procedure making small weight adjustments to keep the error low. The general drift is toward the origin.

2. The weight set reaches the point where $d(\mathbf{W}^*) = 0$ and no further weight decay is possible without the network making a large error. This is the point where two energy minima in the network are of equal depth.

3. The weights continue to decay and the network makes a large error by basin hopping into the wrong energy minimum.

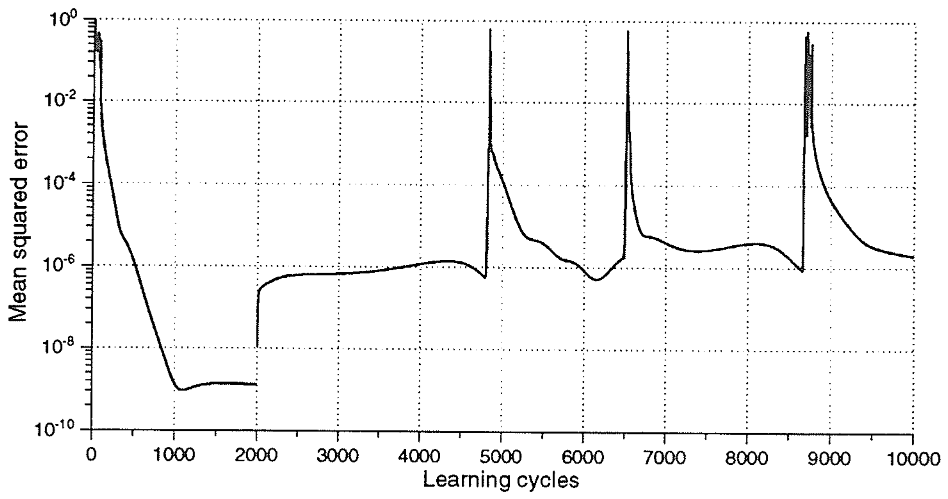


Figure 6.11: The effect of weight decay after correct learning for a $4 \times 4 \times 1$ network learning a 4-bit parity problem. Decay is turned on at 2000 epochs. For this test, $\eta = 0.1$, $d = 1 \times 10^{-5}$ (decay is 0.01% of the learning rate).

4. The CHL algorithm makes a large weight adjustment to correct the error, moving the weight set well away from the $d(\mathbf{W}^*) = 0$ point.

5. The entire process repeats from step 1.

As mentioned in chapter 4, keeping the weight decay rate and add/subtract offsets to a minimum decreases the frequency of oscillation because it slows the drift toward $d(\mathbf{W}^*) = 0$. Using a large learning rate further improves the situation by making the weight step taken in response to an error as large as possible, giving the weights a greater distance to drift before the next error occurs. There does not appear to be a complete solution to weight drift problems in continuous solution sets.

6.2.4 Weight Saturation[3]

When a weight value is represented as an analog voltage stored on a capacitor, there is a limited range of values it can assume. The absolute upper and lower bounds are determined by the supply voltages, but the need to stay within the linear portion of the multiplier characteristic determines the practical weight limits. The design of the multipliers, as well as the multiplier control voltages, finally determines the scaling of the voltages to effective weight values.

The need to limit the effect of noise in the circuit requires the use of as wide a voltage range as possible in the weight storage capacitor. This leads to the possibility of weight saturation, where the learning algorithm is no longer able to increase the magnitude of a weight.

Whether or not weight saturation affects the performance of the network depends on the problem being learned and on the severity of the saturation. If the network has sufficient free parameters to compensate for the saturation of one or more weights, the effect is usually an increase in training time, but the final error performance does not deteriorate (see figure 6.12).

If the network is given more freedom in choosing the weights through the addition of extra hidden units, it can tolerate more severe weight limits (see figure 6.13). Figure 6.14 shows that most of the weights in the severely limited network of figure 6.13 have saturated. Weight saturation is not a serious problem as long as the network retains enough freedom to compensate for the saturated weights. Fortunately, the scaling of the weights in a hardware implementation can be controlled relatively easily by adjusting the unit activation gain function,[4] which must be variable to implement simulated annealing. At the final annealing temperature, the net input to all units is scaled by $1/T_{\text{final}}$, so the weight range can be adjusted to optimize the saturation/noise tradeoff simply by varying T_{final}.

[3]Weight saturation, or "clipping" is discussed in [HKP91] for dynamical Hopfield networks with a given set of weights. In a Hopfield network, the effect is a reduction in the capacity of the network. We want instead to find its effect on *learning* in a DBM

[4]$a_i = \tanh((1/T)\sum w_{ij}a_j)$. Changing T effectively scales all the w_{ij}.

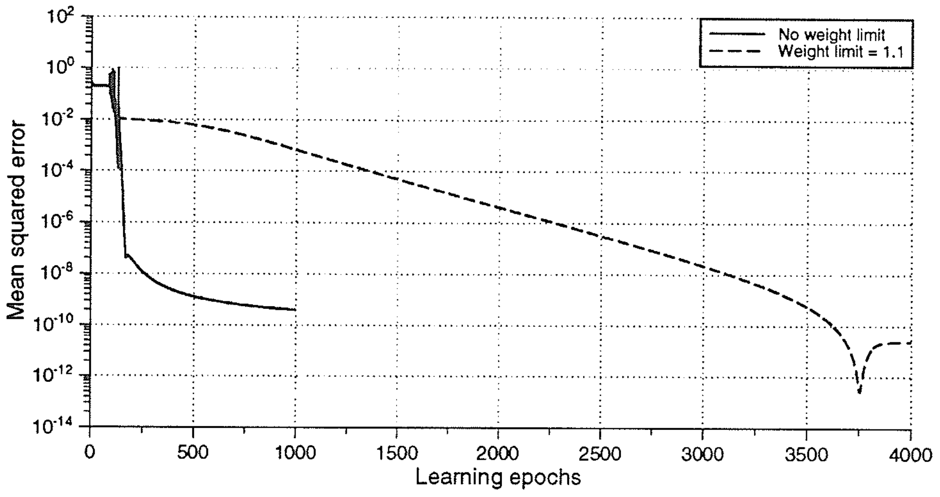


Figure 6.12: Performance of a $2 \times 1 \times 1$ network on an XOR problem. Weight limit is set at 1.1. The maximum weight size in the unlimited network is 1.15. The tolerated saturation is small and learning is greatly slowed because of the small number of adjustable parameters in this network.

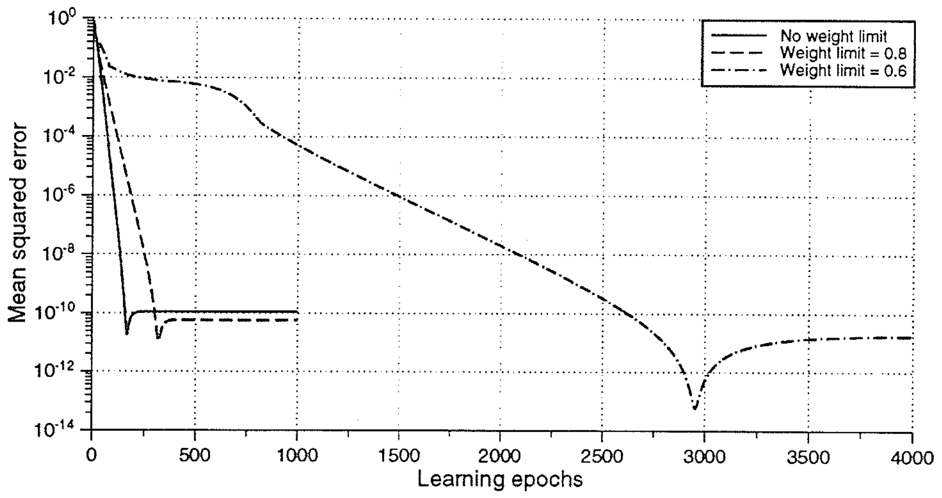


Figure 6.13: Performance of a $2 \times 2 \times 1$ network on an XOR problem. Weight limit is set at 0.8 and 0.6. The maximum weight size in the unlimited network is 0.95. This network has one more hidden unit than the one in figure 6.12.

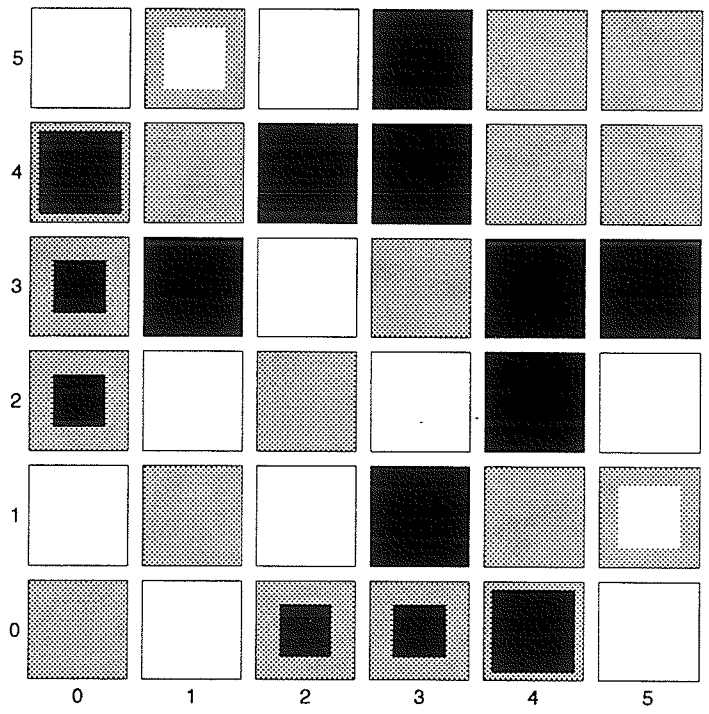


Figure 6.14: Weight set for the network of figure 6.13 with a weight limit of 0.6. The solid white and solid black weights are all saturated at ±0.6.

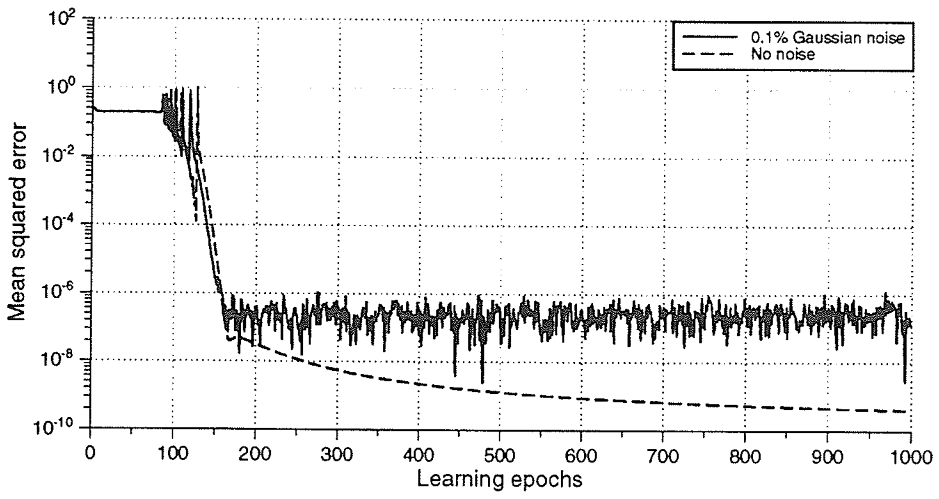


Figure 6.15: The effect of learning noise on a $2 \times 1 \times 1$ network on the XOR problem. Learning rate is $\eta = 0.05$, noise is Gaussian with an amplitude of 5×10^{-5}.

6.2.5 Noise

No analog system is free from noise. Noise levels can be controlled through careful design, but some noise is inevitable. The effect of noise in the weight multiplier, input summation, and unit activation function is a noisy error in the network outputs. If there are multiple minima of near equal depth in the energy function, the network may jump or settle into spurious minima, resulting in incorrect answers and an unreliable network. This effect is not remedied by averaging, but the learning algorithm may remove the spurious minima. Simulations show that the minima are removed in small networks, while larger ones tend to become unstable.

Noise in the learning circuitry has a different effect. It causes weight adjustments of the wrong size or even of the wrong sign. If the noise level is significant as compared to the learning rate, training is severely affected. In most cases, the effect of noise is an increase in the residual error that cannot be removed by averaging (see figure 6.15).

6.3 Discussion

Determining the effect of non-ideal hardware behavior is difficult. It appears that a DBM can tolerate moderate levels of most hardware deficiencies if the problem being learned has isolated solutions. Conversely, a network learning a problem that has a continuous solution set, like XOR, cannot tolerate even minuscule levels of weight decay and add/subtract offsets[5] in the learning circuitry.

As a general rule, weight decay and add/subtract offsets should be kept to a minimum. This lowers residual error rates in networks that have isolated solution sets and minimizes the oscillation frequency in networks that have continuous solution sets. It is also useful to use as large a learning rate as possible in order to decrease the oscillation frequency.

[5]unless learning thresholds are used. See section 6.2.2.

Chapter 7

Learning Analog Values

7.1 Introduction

Deterministic Boltzmann machines normally learn by adjusting the minima of an energy surface so that the unit activations relax to a desired output pattern given a particular input stimulus. In this chapter it is shown that a DBM can learn to adjust its settling *dynamics* so that desired analog outputs are produced without ever reaching an energy minimum.

For some types of problems, it is desirable for a DBM to generate continuous analog rather than digital output values in response to analog input patterns. While a digital output is considered "correct" as long as it is on the proper side of some threshold. Analog outputs must match their teaching patterns to within some application-dependent performance criterion to be useful.

When an input pattern is applied to a digital-output DBM, the energy minima of the network change so that the outputs settle to the values learned during training, or at least to the right side of the threshold. Since an analog problem has a continuous, and therefore infinite, set of input and output vectors, the locations of the energy minima would have to be continuously adjustable. Simulations show that this does not happen.

7.2 Network Configuration

A fully-connected eleven unit $2 \times 6 \times 2$ DBM network is used in these experiments. The two outputs are always complements of each other in the training set, so there is effectively only a single output. Weights are constrained to be symmetric ($w_{ij} = w_{ji}$). Except as noted, all simulation procedures are the same as in the experiments of previous chapters.

7.3 Observed Behavior

The network is trained with an analog version of a 2-input XOR problem. The training set consisted of an 11×11 array of analog input/output pairs generated by the function

$$P(x, y) = \frac{(x-y)^2 - (x+y)^2}{4} \tag{7.1}$$

where x and y are the two inputs, ranging from -1 to $+1$, and P is the output, also ranging from -1 to $+1$.

After the network had been trained for 6000 epochs with $T_{\text{error}} = 0.0001$ and another 1000 epochs with $T_{\text{error}} = 0.00002$, it was tested and the activation values during annealing recorded. The results for two different input vectors are shown in figure 7.1.

Looking at the energy surface in figure 7.2, it is obvious that neither of the test cases settles to the stable energy minimum at $a_{out} = 0.64$. However, the network has learned to produce outputs close to the training values by adjusting the activation dynamics so that the activations are correct at the end of the finite number of annealing/settling passes allowed in this simulation. If the network were given more time to settle, the output would eventually reach 0.64 for both test cases. (Actually, the energy minimum is not exactly the same for both cases, but it is very close.)

7.3.1 Geometric Annealing

The same experiment was performed for a network trained with a geometric annealing schedule and multiple-pass asynchronous activation updating at each temperature (see figure 7.3). The mean squared error only decreases to its minimum value after the network

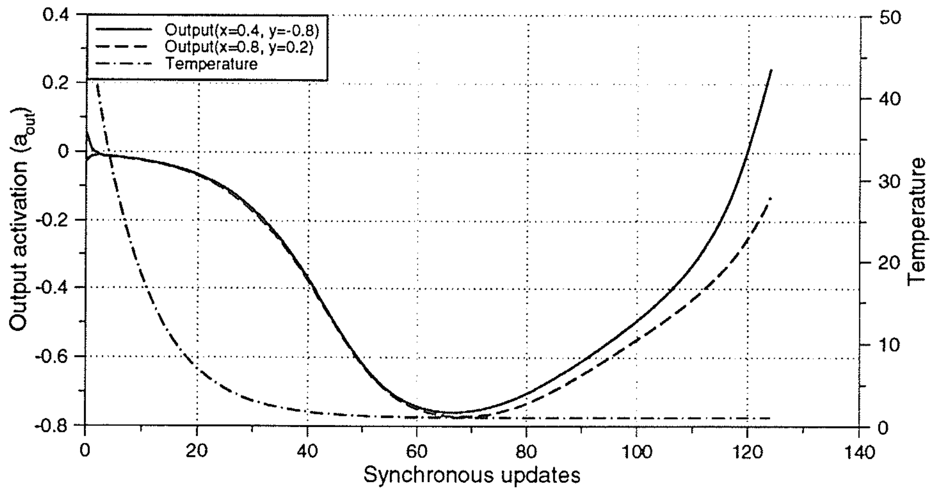


Figure 7.1: Tests of two training patterns after 7000 training epochs. The outputs have not settled for either test case, but the results are correct to within 4% of the ±1 unit range.

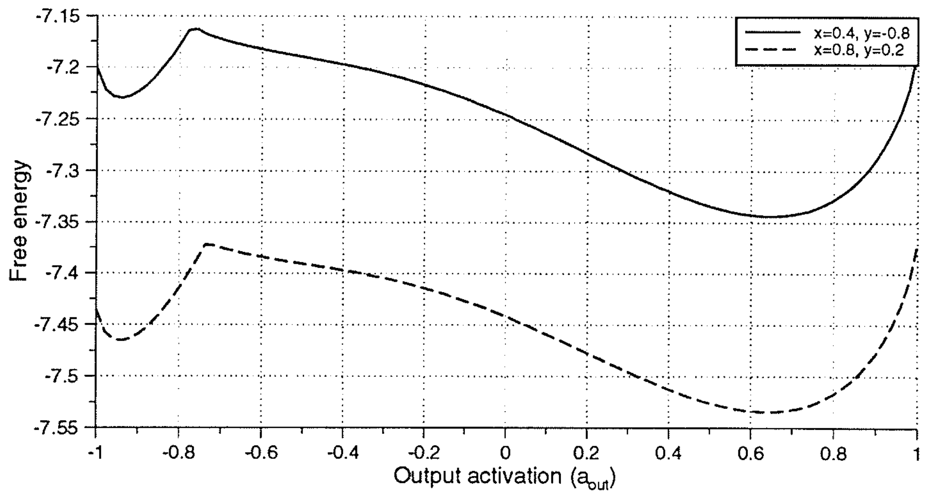


Figure 7.2: Network free energy as a function of output activation. Produced by holding output unit activation at points between -1 and $+1$ and allowing the network to settle.

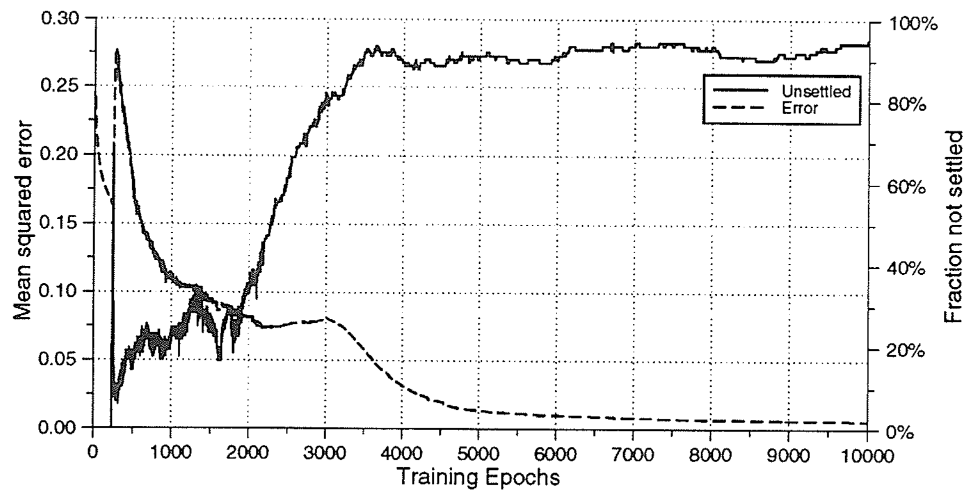


Figure 7.3: Performance of a network using a 10 step geometric annealing schedule with $T_{\text{initial}} = 20$, $T_{\text{final}} = 1$, and 100 passes allowed at each temperature. *Error* measures the distance between the network outputs and the training values. *Unsettled* is the fraction of the input patterns for which the network did not reach a final, stable, energy minimum.

has adjusted its dynamics so that it fails to settle on almost all of the input patterns. Conversely, the "Fraction Correct" (in the table below), which measures whether or not the output pattern has the correct sign, barely increases after 2000 training epochs. Figure 7.4 shows how the network dynamics change as the network is trained.

Epochs	MSE	Fraction Correct	Not Settled
2000	0.0837	79%	38%
10000	0.0078	80%	96%
20000	0.0058	82%	96%

7.4 Activation Dynamics

The activation updating rule is

$$a_i^{t+1} = (1-\tau)a_i^t + \tau \tanh\left(\frac{1}{T} net_i^t\right). \tag{7.2}$$

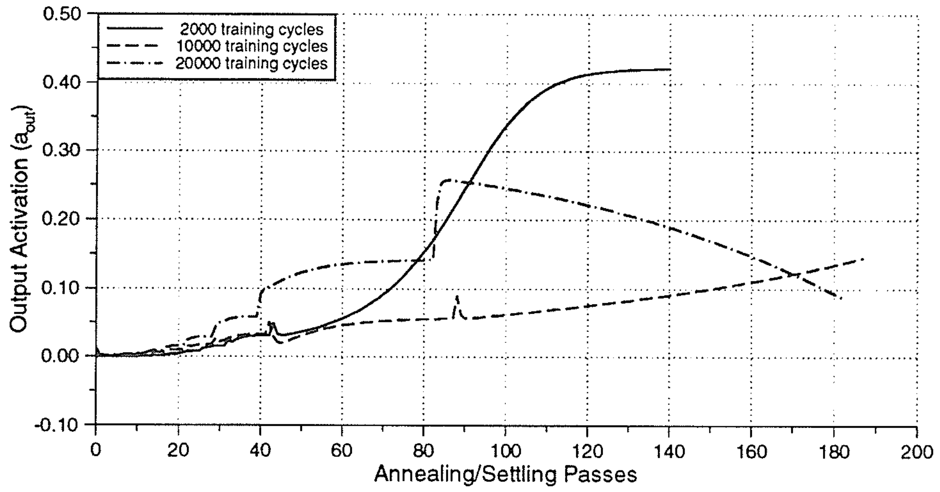


Figure 7.4: Annealing/relaxation dynamics for the network of figure 7.3 using test pattern $x = -0.4$, $y = 0.2$, $P(x, y) = 0.1$ after different amounts of training. The 2000 cycle case settles to a stable state while the later cases do not. The 20000 cycle case settles to a value very close to the training data.

The differential equation solved by (7.2) is

$$\frac{da_i}{dt} = \tau(-a_i + f_i(net_i)) \tag{7.3}$$

where $f_i(\cdot) = \tanh(\cdot)$ is the sigmoid unit transfer function.

Normally, the activations are updated until they settle to an equilibrium condition where $f_i(\check{net}_i) = \check{a}_i$ and $da_i/dt = 0$. Movellan [Mov90b] derives the contrastive Hebbian learning (CHL) procedure from this equilibrium condition.

The question arises: what happens if, after a fixed number of annealing/settling passes, the network fails to reach equilibrium and the CHL procedure is applied anyway? Imagine a case where the the final unclamped state (vector of activations) is $\tilde{\mathbf{a}}^-$ and the final clamped state is $\tilde{\mathbf{a}}^+$, neither being at equilibrium. See figure 7.5 for an illustration ($\tilde{}$ indicates the *final non-equilibrium* state). The vectors $\tilde{\mathbf{a}}^-$ and $\tilde{\mathbf{a}}^+$ are likely to be close together because some learning has already taken place. They will usually at least have the same sign. The CHL weight updating rule is

$$\Delta w_{ij} = \eta(a_i^+ a_j^+ - a_i^- a_j^-).$$

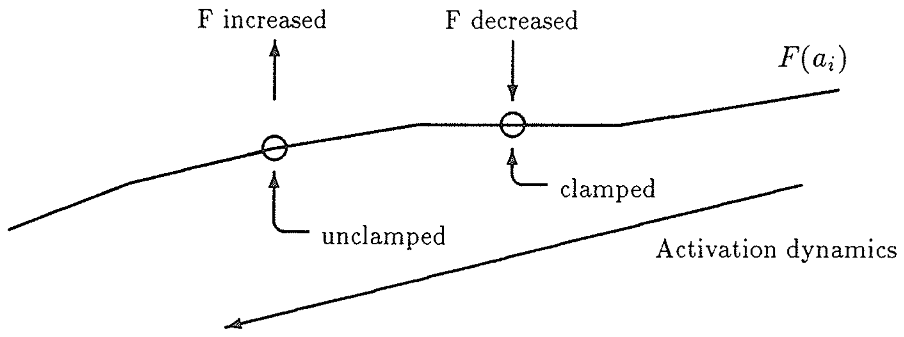


Figure 7.5: Effect of applying CHL when not at equilibrium

Assume $\tilde{a}_j^- \approx \tilde{a}_j^+ \doteq \tilde{a}_j^\pm$ near $\tilde{\mathbf{a}}^+$. We then have

$$\begin{aligned}
\Delta w_{ij} &= \eta \tilde{a}_j^\pm (\tilde{a}_i^+ - \tilde{a}_i^-), \\
\tilde{net}_i + \Delta net_i &= \sum_j (w_{ij} + \Delta w_{ij}) \tilde{a}_j, \\
\tilde{net}_i + \Delta net_i &= \sum_j [w_{ij} + \eta \tilde{a}_j^\pm (\tilde{a}_i^+ - \tilde{a}_i^-)] \tilde{a}_j.
\end{aligned}$$

If $\tilde{a}_j \approx \tilde{a}_j^\pm$ then

$$\begin{aligned}
\tilde{net}_i + \Delta net_i &= \sum_j w_{ij} \tilde{a}_j + (\tilde{a}_i^+ - \tilde{a}_i^-)(\eta) \sum_j (\tilde{a}_j)^2, \\
\Delta net_i &= (\tilde{a}_i^+ - \tilde{a}_i^-) \eta \sum_j (\tilde{a}_j)^2. \qquad (7.4)
\end{aligned}$$

Replacing net_i in (7.3) with the updated net input, we obtain

$$\frac{da_i}{dt} + \Delta \frac{da_i}{dt} = \tau(-a_i + f_i(net_i + \Delta net_i)). \qquad (7.5)$$

Since $f_i(\cdot)$ is a monotonically increasing function, $\Delta \partial a_i / \partial t$, the change in the slope of the activation dynamics due to the weight update, always has the same sign as Δnet_i, the change in the net input to unit i. Now, if $\tilde{a}_i^- < \tilde{a}_i^+$ and $\partial a_i / \partial t < 0$, as in figure 7.5, $\Delta net_i > 0$ so $\Delta \partial a_i / \partial t > 0$, and the network dynamics slow down so that $\tilde{a}_i^-$ is closer to $\tilde{a}_i^+$ after the fixed number of settling passes, and the network error performance increases.

A similar argument can be made for cases where $\partial a_i/\partial t > 0$ or $\tilde{a}_i^- > \tilde{a}_i^+$, always adjusting the network dynamics so that $\tilde{a}_i^-$ moves closer to $\tilde{a}_i^+$.

Note that $\tilde{a}_i^+$ is a clamped value if unit i is an output, so the change in network dynamics does not affect $\tilde{a}_i^+$ at all. Even for a hidden unit, the clamped outputs cause unit i to settle rapidly near $\tilde{a}_i^+$ despite the modified network dynamics.

7.5 Discussion

It has been shown that a DBM can learn to recall analog input/output mappings without settling into energy minima. It is interesting that the arguments presented here do not apply only to analog problems, but also to discrete ones, where the desired outputs are a few distinct values instead of a continuous range. In fact, the behavior described here has been observed when a network was trained with analog input patterns and discrete output patterns, although less frequently than with analog outputs. In applications where only two discrete output values are required (the binary case), only the sign of the output need be correct. However, looking at figure 7.1, we can see that the output would have the wrong sign if the network were allowed to settle completely.

It might be argued that learning by adjustment of the network dynamics results from an inadequate annealing schedule or from an insufficient number of passes through the network to achieve reliable relaxation. This is not the case. In experiments where the network is allowed a large number of settling passes at each temperature, the learning process simply slowed down the dynamics until all the allowed passes are used up.

This type of analog learning mechanism only applies to network input/output relationships that are continuous and smooth. It does not apply to arbitrary analog problems.

Learning by adjustment of network dynamics is possible when a DBM is implemented using a digital computer because a repeatable annealing/relaxation schedule is easily produced. However, in an analog VLSI implementation, where the network may settle in less than a microsecond, it is unlikely that the annealing/settling time can be controlled accurately enough to allow the network to learn by this method.

Chapter 8

Conclusion

8.1 Learning Instability

The analyses in chapters 4 and 5 are restricted to a very specific problem and network configuration. The following results have been demonstrated:

1. There exist a continuous range of optimal weight sets solving the XOR problem on a single hidden, single output unit deterministic Boltzmann machine network.

2. Since every point in the continuous solution set is optimal, meaning that the outputs of the network exactly match all the training patterns, the CHL learning algorithm cannot distinguish one ideal weight set from another. Therefore, the weights are free to drift to a point where there are two equal-depth energy minima and $d(\mathbf{W}^*) = 0$, causing small weight perturbations to generate gross output errors as the network randomly settles to one of the two minima.

3. The weight drift phenomenon is not due to failure of the mean field approximation for small sized networks. It is possible to construct a stochastic Boltzmann machine that exhibits behavior equivalent to basin hopping in the DBM as the BM approaches infinite size. This implies that DBM behavior cannot be expected to improve with increases in network size, and that the BM statistics gathering period must be in-

creased as the size of the BM increases to avoid behavior analogous to the basin hopping observed in the DBM.

The most frequently encountered manifestations of these findings are error spikes in a DBM learning curve in the latter part of the learning process after the initial random hunting behavior but before the weights have settled down to their final optimal values. In simulation, it may not be obvious that there is a problem because the spikes tend to die out as the weight set is optimized. If learning is arbitrarily terminated when a predetermined performance criterion is reached, the occurrence of further spikes is prevented. The error spikes can also be suppressed by other ad hoc fixes, such as adjusting the learning rate and other network parameters, but there does not appear to be a reliable general solution.

Weight drift is of particular concern in analog hardware implementations of DBMs, where weight decay and learning offsets cannot be entirely eliminated. As there is no threshold level of weight decay that is tolerated, there is no way to eliminate occasional error spikes in an analog system.

Although the existence of continuous solution sets has been conclusively demonstrated only for the 2-bit XOR case on a $2 \times 1 \times 1$ network, the behaviors predicted, namely sensitivity to weight perturbations, decay, and offsets, have been observed in simulation of XOR on larger networks, as well as in n-bit parity and other problems on various sizes of networks. These results lead to the hypothesis that this is not a problem unique to XOR, or even to n-bit parity, but one that is common to many DBM applications.

At the root of the problem is under-constraint of the weight set and the number of adjustable parameters (weights) generally increases as the square of the number of units. It is therefore suspected that these problems will grow worse on still larger[1] networks (attempting larger problems). At some point, error spikes during learning may grow so prevalent that it becomes difficult to learn a large task completely, even on digital hardware.

[1]above the size which can be conveniently simulated, but which are of most interest for hardware implementations.

There are tasks, such as learning overlapping Gaussians, where the training set is self contradictory, and 100% correct performance is not even theoretically possible. It has not been determined whether weight drift increases the error rate in such cases, or whether any such increase would be significant compared to the error inherent in the problem.

8.2 Hardware Issues

It has been determined that a DBM network is able to tolerate moderate amounts of most non-ideal behavior, with the exception of weight decay and weight update offsets (without thresholding), as mentioned above.

In particular, most deficiencies in the weight multiplier, input addition, and unit activation function are well tolerated because the learning algorithm easily compensates for them. Essentially, the learning algorithm cannot determine whether the activation of a unit needs to be adjusted because of an incorrect weight value, or because of non-ideal analog circuitry. As long as a weight adjustment moves the activation in the anticipated direction, the network performs well.

DBM networks are particularly sensitive to non-ideal behavior in the learning and weight storage circuitry because it directly affects the ability of the network to make the required weight adjustments. Most serious are offsets in the weight update circuit and weight decay. The lack of a threshold level of allowable decay or offset makes these problems persistent. It is possible to minimize the frequency of error spikes by minimizing offset and decay, and by using a fairly high learning rate to ensure that large weight steps are taken whenever an error occurs.

8.3 Future Work

It would be useful to characterize the types of learning tasks and network architectures that lead to continuous solution sets, and therefore to all the problems discussed above. The goal would be to develop a modified learning rule, or some form of restriction on the network weights, that creates isolated optimal weight sets.

Another outstanding issue is whether weight drift effects cause increased error rates in real, noisy, self-contradictory problems. It is our belief from simulations that they do, but this has not been proven.

Further characterization of hardware design issues is probably best left until the weight drift problem has been resolved, since isolated solution sets would automatically solve the most serious hardware problems, namely weight decay and add/subtract offsets. It is also very difficult to quantitatively evaluate the consequences of hardware design tradeoffs when the network is as unstable as most of those simulated in this thesis.

The most desirable property of the DBM nerual network architecture is its simple, local, learning rule. This makes it an ideal candidate for use in highly parallel digital, analog, or mixed VLSI hardware. If a way can be found to reliably generate isolated weight solution sets, most of the outstanding hardware design issues will be automatically solved. It would then be possible to build truely large neural network systems with many thousands of units and apply it to real-world problems.

Bibliography

[AA90] Robert B. Allen and Joshua Alspector. Learning stable states in stochastic asymmetric networks. *IEEE Transactions on Neural Networks*, 1(2), June 1990.

[AAJ90] Joshua Alspector, Robert B. Allen, and Anthony Jayakumar. Relaxation networks for large supervised learning problems. In *Proceedings of NIPS-90, in press*, 1990.

[AdF91] Bruno Apolloni and Diego de Falco. Learning by asymmetric parallel Boltzmann machines. *IEEE Transactions On Neural Networks*, 2(1), January 1991.

[AHS85] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[AK89] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley and Sons, 1989.

[AZL92] Joshua Alspector, Torsten Zeppenfeld, and Stephan Luna. A volatility measure for annealing in feedback neural networks. *Neural Computation*, 4:191–195, 1992.

[BP91] Pierre Baldi and Fernando Pineda. Contrastive learning and neural oscillations. *Neural Computation*, 3:526–545, 1991.

[Cho86] Debashish Chowdhury. *Spin Glasses and Other Frustrated Systems.* World Scientific Publishing Company, 1986.

[Dor80] Richard C. Dorf. *Modern Control Systems.* Addison-Wesley, 1980.

[Gal] Conrad Galland. Personal communication.

[GH90] Conrad C. Galland and Geoffrey E. Hinton. Deterministic Boltzmann learning in networks with asymmetric connectivity. In *Connectionist Models: Proceedings of the 1990 Summer School*, pages 3–9, San Mateo, California, 1990. Morgan Kaufmann Publishers, Inc.

[Hin87] Geoffry E. Hinton. Connectionist learning procedures. Published in the journal *Artificial Intelligence*, December 1987.

[Hin89] Geoffry E. Hinton. Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation*, 1(1), 1989.

[HKP91] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Co., 1991.

[Hop84] John J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci.*, 81:3088–3092, 1984.

[Hop88] John J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, September 1988.

[HP87] Geoffry E. Hinton and David C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the Cognitive Sciences Conference*, Seattle, Washington, 1987.

[KS85] Eric R. Kandel and James H. Schwartz. *Principles of Neural Science.* Elsevier Science Publishing Co., 1985.

[Liv91] Mike Livesey. Clamping in Boltzmann machines. *Neural Computation*, 3:402–408, 1991.

[Ma85] Shang-Keng Ma. *Statistical Mechanics.* World Scientific Publishing Company, 1985.

[MC89] Gagan Mirchandani and Wei Cao. On hidden nodes for neural nets. *IEEE Transactions on Circuits and Systems*, 36(5):661–664, May 1989.

[Mea88] Carver A. Mead. *Analog VLSI and Neural Systems.* Reading: Addison-Wesley, 1988.

[Mov90a] Javier R. Movellan. Contrastive Hebbian learning in interactive networks. Connectionists Mailing List, April 1990.

[Mov90b] Javier R. Movellan. Contrastive Hebbian learning in the continuous Hopfield model. In *Connectionist Models: Proceedings of the 1990 Summer School*, pages 10–17, San Mateo, California, 1990. Morgan Kaufmann Publishers, Inc.

[MP88] Marvin L. Minsky and Seymour A. Papert. *Perceptrons.* MIT Press, Cambridge, Massachusetts, 1988.

[MR88] James L. McClelland and David E. Rumelhart. *Explorations in Parallel Distributed Processing.* MIT Press, Cambridge, Massachusetts, 1988.

[PA87] Carsten Peterson and James R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsy, and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988.

[PH89] Carsten Peterson and Eric Hartman. Explorations of the mean field learning algorithm. *Neural Networks*, 2:475–494, 1989.

[RM87a] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of cognition - Foundations*, volume 1. MIT Press, Cambridge, Massachusetts, 1987.

[RM87b] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of cognition - Psychological and Biological Models*, volume 2. MIT Press, Cambridge, Massachusetts, 1987.

[Sch91] Christian Schneider. *Analog CMOS Circuits for Artificial Neural Networks.* PhD thesis, University of Manitoba, 1991.

[SSC90] Christian Schneider, Roland Schneider, and Howard Card. Analog cmos synapse with in situ hebbian learning. In *Proceedings of CCVLSI'90*, Ottawa, Ontario, Canada, 1990.

[YK89] C. H. Youn and S. C. Kak. Continuous unlearning in neural networks. *Electronics Letters*, 25(3), February 1989.