1992

The relaxation method for learning in artificial neural networks

Hee-Kuck Oh
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation


This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600
The relaxation method for learning in artificial neural networks

Oh, Hee-Kuck, Ph.D.
Iowa State University, 1992
The relaxation method for learning in artificial neural networks

by

Hee-Kuck Oh

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Department: Computer Science
Major: Computer Science

Iowa State University
Ames, Iowa
1992

Copyright © Hee-Kuck Oh, 1992. All rights reserved.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................... viii

1. INTRODUCTION ............................................. 1
   1.1 Foundations of Artificial Neural Networks ............... 2
       1.1.1 Biological neuron .................................. 3
       1.1.2 Artificial neural networks ....................... 4
       1.1.3 Learning in neural networks ..................... 8
   1.2 An Overview of the Dissertation ......................... 10

2. LITERATURE SURVEY ........................................ 14

3. THE RELAXATION METHOD .................................... 21
   3.1 Introduction ........................................... 21
   3.2 The Relaxation Method ................................ 22
   3.3 The Maximal Distance Relaxation Method ............... 26
   3.4 The Pseudo-Relaxation Method ......................... 29
   3.5 Concluding Remarks .................................. 34

4. APPLICATIONS OF THE RELAXATION METHOD TO LEARN-
   ING IN NEURAL NETWORKS .................................. 36
   4.1 Introduction .......................................... 36
4.2 The Hopfield Model ........................................... 38
  4.2.1 Network stability ........................................ 39
  4.2.2 Correlation encoding .................................... 42
  4.2.3 Formulation of learning ................................ 42
  4.2.4 An iterative learning algorithm ....................... 44
4.3 Bidirectional Associative Memory .......................... 47
  4.3.1 BAM stability .......................................... 49
  4.3.2 BAM encoding .......................................... 50
  4.3.3 An iterative learning algorithm ....................... 50
4.4 Dynamic Heteroassociative Neural Memory ............... 55
  4.4.1 The Ho-Kashyap algorithm ............................ 55
  4.4.2 An iterative learning algorithm ....................... 59
4.5 Radial Basis Function Networks .......................... 60
  4.5.1 Multivariable interpolation using RBFs ............... 61
  4.5.2 RBF networks .......................................... 62
  4.5.3 Non-exact interpolation using RBFs ................... 64
4.6 Concluding Remarks ....................................... 67

5. SIMULATION STUDY ........................................ 69
  5.1 Introduction .............................................. 69
  5.2 Performance of Associative Memories .................... 70
    5.2.1 Random patterns .................................... 70
    5.2.2 Correlated patterns: CGA character font .......... 72
  5.3 Sensitivity to Learning Parameters ...................... 75
    5.3.1 Initial weights ..................................... 75
iv

5.3.2 Relaxation factor $\lambda$ ................................................. 77
5.4 Delta Rule vs Relaxation Method ............................................. 79
5.5 Scalability ................................................................. 86
5.6 Concluding Remarks ....................................................... 87

6. CONCLUSION ............................................................... 90

BIBLIOGRAPHY ............................................................. 92
### LIST OF TABLES

| Table 5.1: | Storing random patterns in Hopfield networks | 71 |
| Table 5.2: | Storing random patterns in BAMs | 72 |
| Table 5.3: | Effect of the initial choice of weights on learning: initial values are randomly chosen between $-1$ and $1$ | 78 |
| Table 5.4: | Effect of the initial choice of weights on learning: initial values are randomly chosen between $-500$ and $500$ | 78 |
| Table 5.5: | Effect of the relaxation factor $\lambda$ | 79 |
| Table 5.6: | Sensitivity of learning to initial choice of weights: delta rule vs relaxation algorithm | 84 |
| Table 5.7: | Scalability of learning speed, measured in number of epochs | 88 |
LIST OF FIGURES

Figure 1.1: A simple artificial neuron ............................................. 5
Figure 1.2: Commonly used activation functions ................................. 6
Figure 1.3: Typical neural network topologies ..................................... 7

Figure 3.1: Geometric illustration of relaxation procedure ...................... 25
Figure 3.2: An example of an over-relaxation sequence .......................... 26
Figure 3.3: Various relaxation procedures ........................................ 27
Figure 3.4: Non-terminating relaxation sequences ............................... 29
Figure 3.5: Fast convergence of the pseudo-relaxation method ................. 34

Figure 4.1: A Hopfield network ....................................................... 39
Figure 4.2: BAM architecture ......................................................... 48
Figure 4.3: DAM architecture ......................................................... 56
Figure 4.4: RBF network architecture ............................................. 63

Figure 5.1: The sample training set of 93 patterns ............................... 73
Figure 5.2: Recognition capability in the presence of noise when 10 digits
are stored in a 49-node Hopfield network ...................................... 74
Figure 5.3: Recognition capability in the presence of noise when 26 capital
letters are stored in a 49-node Hopfield network .............................. 75
Figure 5.4: Recognition capability in the presence of noise when 5 vowel pairs are stored in a 49-49 BAM .......................... 76
Figure 5.5: Recognition of "(A,a)" when a BAM is trained by the relaxation algorithm .................................................. 77
Figure 5.6: Effect of varying \( \lambda \) on the learning speed when all of the 93 character fonts are stored in a network of 49 nodes ........ 80
Figure 5.7: Effect of varying \( \lambda \) on the learning speed in BAMs ............................................................. 81
Figure 5.8: Learning curve for the relaxation algorithm when a 100-node network is trained on 150 random patterns ............... 82
Figure 5.9: Learning curve for the delta rule when a 100-node network is trained on 150 random patterns ............................. 83
Figure 5.10: Effect of varying relaxation factor \( \lambda \) on the learning speed of the relaxation algorithm .............................. 85
Figure 5.11: Effect of varying gain factor \( \eta \) on learning speed of the delta rule ......................................................... 86
Figure 5.12: The update \( W^D \) using the delta rule and the update \( W^R \) using the relaxation method are shown for two initial positions of the point \( W \). The distance between \( W^D \) and \( W \) is fixed, whereas the distance between \( W^R \) and \( W \) varies depending on the position of \( W \) ................................................. 87
Figure 5.13: Scalability of the learning speed ............................... 89
AKNOWLEDGEMENTS

I would like to express my grateful appreciation to Dr. Suraj Kothari for his guidance and encouragement in my studies at Iowa State University and for his assistance and helpful criticisms in the preparation of this thesis.

I am also deeply indebted to my Graduate Committee: Dr. John Boysen, Dr. David Fernandez-Baca, Dr. William Lord and Dr. Satish Udpa for their helpful comments, advice and suggestions toward the completion of this thesis.

Special thanks go to Dr. Rex Thomas, Dr. John Boysen and their research group in Computation Center at Iowa State University for the financial support and valuable experience in my career.

Appreciation is also extended to my colleagues and friends in Ames, Iowa, especially to Pinaki Shah and Barbara Montgomery for many interesting and enjoyable discussions and above all, for their friendship.

Finally, to my wife, Soomee, my son, Seung-Kyun, and my daughter, Hyun-Joo, I want to express my deepest appreciation for their patience, support and cooperation during my graduate studies.
1. INTRODUCTION

Artificial neural networks are the present day prototypes for imitating the functionality of the human brain. Despite the recent remarkable advances in computer technology, there are certain problems that cannot be effectively solved by conventional digital computers. Computers can handle number crunching tasks very efficiently – almost at a blinding speed in many cases. In contrast, the human brain outperforms digital computers when dealing with intelligent information processing such as pattern recognition and language comprehension. For instance, a one-year old infant can trivially recognize the voice and the face of its mother, while similar perceptual interpretation is still a difficult task for today's most advanced pattern recognition devices and rule based expert systems.

There is no doubt that the human brain is the most efficient machine for visual pattern recognition and language comprehension. What makes the brain superior to conventional computers in intelligent information processing and what are the underlying computational principles that the brain uses? The fundamental discrepancy in information processing capabilities of digital computers and human brains seems to be due to the differences in processing methods. Conventional computers rely on sequential programs and the control is centralized by a complex central processing unit. This style of processing makes conventional computing more suitable for appli-
ations that are deterministic and precisely controllable. Compared to the computers' rigorous and precise processing style, the brain is adaptive and resilient in nature. The brain can make reasonable decisions based on incomplete or often ambiguous knowledge. The computational richness of the human brain comes from billions of neurons that are highly connected to each other and operate in parallel. The brain relies on highly distributed representations and transformations and its control is distributed through neurons. Information storage and retrieval in the brain appears to be taking place at neurons and the connections called synapses between neurons. While individual neurons in the human brain process information at a rate much slower than that of a digital computer, the brain performs its processing feat through massive parallelism, using billions of neurons and more than 1000 times that many interconnections.

The brain has taken millions of years to evolve into its current architecture. Although scientists have studied the brain functions for many years, current understanding of the brain functions is still limited and it may take a long time to get a concrete and complete idea of how the brain works. The goal of neural network research is to understand further the brain functions as well as to use current understanding in the design of practical systems to solve the problems that are not easily solved by digital computers.

1.1 Foundations of Artificial Neural Networks

Artificial neural networks are biologically inspired and use the structure of the human brain to try to emulate the intelligent information processing of the brain. Ashby [13] describes many of these processes in his book, *Design for a Brain*. Since
that time attempts to understand physiological systems and processes and to use this understanding in creating new designs have continued. Such an interest in the use of physiological phenomena led to the study of the basic processing element of nervous systems, the neuron.

1.1.1 Biological neuron

A neuron is the basic functional unit of the nervous system and is interconnected with numerous adjacent neurons in the network of the entire nervous system. Each neuron is an integrator and a transmitter of coded information, reacting to stimuli and transmitting the resulting excitation to other neurons. The neuron receives information through abundant branches, called dendrites, comprising the outer surface of the neuron. A long, thin cylindrical fiber, called the axon, may be considered the output cable of the neuron. The axon is capable of electrochemically transmitting information concerning the state of the neuron.

The axon terminates in endbulbs near the dendrites of other neurons. These near connections are called synapses. A synapse is the region of specialized contact between neurons, where actual neuron-to-neuron communication takes place. There are two kinds of synapses: excitatory synapses and inhibitory synapses. Excitatory synapses relay impulses that have effects on the firing of the next neuron and inhibitory synapses convey the impulses that have the reverse effect. An impulse traveling down the axon terminates at the endbulbs. A neuron produces an output only when sufficient impulses are presented to its dendrites in a short period of time. Since some inputs via inhibitory synapses hinder firing, a neuron fires only when the active excitatory inputs exceed the active inhibitory inputs by an amount equal
to the "threshold value" of the neuron. A portion of the neuron, called the spike initiator locus, performs this summation and comparison, producing a pulsed output signal for the stated conditions of input.

1.1.2 Artificial neural networks

Artificial neural networks (or simply neural networks) adopt the brain metaphor of information processing. Currently, neural networks consist of simple processing elements, often called nodes or units, operating in parallel, and numerous interconnections between these nodes. A processing element emulates the axons and dendrites of its biological counterpart with wires; the synapses by using registers with weighted values. Processing elements contain combinations of excitatory or inhibitory weights that act on the inputs in a summation function. If the sum of inputs to a processing element exceeds the threshold value of the processing element, it will produce an output signal based on an activation function.

Figure 1.1 illustrates a processing element. \( X_i \) is an input to the processing element and \( W_i \) is the connection strength. In many practical systems, an activation function \( f \) produces an output after summing the weighted inputs and the threshold \( \theta \). Mathematically, given an input vector \( (X_1, \ldots, X_i, \ldots, X_N) \), a weight vector \( (W_1, \ldots, W_i, \ldots, W_N) \) and a threshold \( \theta \), the output \( Y \) is computed by

\[
Y = f\left( \sum_{i=1}^{N} W_i X_i - \theta \right)
\]

Figure 1.2 shows a few typical activation functions. The linear function, shown in Figure 1.2(a), is often used for the activation function of the output node in multi-layer networks such as backpropagation networks [100] and Radial Basis Function
networks [19][85][96]. All the remaining functions are non-linear. The step function, shown in Figure 1.2(b), has been widely used in several architectures including perceptrons [99], the Hopfield model [62], Bidirectional Associative Memory [76], Dynamic Heteroassociative Neural Memory [54] and many other binary associative memories. If the linear function is bounded to a certain range, it becomes the ramp function shown in Figure 1.2(c). The ramp function is used in Brain-state-in-a-box [10]. The remaining three functions, sigmoid, hyperbolic tangent and gaussian functions, share the following common characteristics: they are all non-linear, continuous and differentiable. These functions are commonly used in multi-layer networks where the differentiability of the activation function is necessary.

Depending on the direction of propagation, neural networks can be further classified as feedforward networks or feedback networks. Typically, a feedforward network consists of several layers of processing nodes. Every node in any layer receives inputs
from lower layers and sends outputs to higher layers. Thus, in feedforward networks, the inputs are fed into the lowest layer and the outputs are produced at the highest layer. Unlike feedforward networks, the propagation direction in feedback networks is not necessarily unidirectional. Furthermore, feedback networks may not have separate input and output nodes. Inputs to the network could be fed into either all the nodes or some of the nodes. For a given input, a feedback network, because of its feedback nature, continuously updates its states, and then produces the outputs

Figure 1.2: Commonly used activation functions
Figure 1.3: Typical neural network topologies
when the network is stabilized. Figure 1.3 illustrates typical networks.

1.1.3 Learning in neural networks

There is no notion of programming in neural networks; instead, they are trained to give acceptable answers. In the training phase, normally known information is fed into the network to determine weights for the connections, i.e., synaptic strength between two nodes. Typically the training phase requires many learning cycles running repeatedly until the output is satisfactorily accurate. After the training is completed, information is encoded in a weighted matrix associated with interconnections of the network. A detailed taxonomy of neural network models and their learning algorithms can be found in [79]. Some of the basic types of learning rules are:

- **Hebbian learning**: A learning strategy, attributed to D. O. Hebb [55], that suggests the adjustment of connection weight between two nodes according to the correlation of the values of the two nodes. Hebb originally described a concept of learning as follows:

  "When an axon of cell $A$ is near enough to excite a cell $B$ and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that $A$'s efficiency as one of the cells firing $B$ is increased."

Examples of Hebbian learning are the first-order correlation encoding [62][76], Sejnowski’s covariance correlation learning rule [105], Sutton and Barto’s learning rule [108], Klopf’s discrete time correlation rule [70], Grossberg’s signal Hebb law [44][45], and the differential Hebb law [73][70].
• **Error-correction learning:** A supervised learning procedure that adjusts the connection weights to reduce the error determined by comparing the actual response to a given input with the desired response. Let $Y_j$ be the desired output at the $j$-th node and $A_j$ be the actual output at the same node. Then a general error-correction learning equation is

$$\Delta W_{ij} = \eta G(Y_j - A_j)X_i$$

where $W_{ij}$ is the connection weight from $i$-th node to $j$-th node, $\eta$ is a small constant, and $G$ is a function of the error $Y_j - A_j$. Examples of error-correction learning are the perceptron learning algorithm [99], the delta rule [117], and the backpropagation algorithm [115][77][91][100].

• **Reinforcement learning:** This model is similar to error-correction learning in that the weights are modified in proportion to the difference between the desired and actual responses. The difference is that while error-correction learning requires a vector of error values, reinforcement learning requires a measure of the adequacy of the actual response, which is scalar. A general reinforcement learning equation is

$$\Delta W_{ij} = \eta(r - t_j)e_{ij}$$

where $r$ is the measure of the adequacy of the actual response, $t_j$ is the reinforcement threshold value for the $j$-th output node, and $e_{ij}$ is the canonical eligibility [123] of the weight, which is dependent on the probability of the desired output equaling the actual output. Examples of this model can be found in [121][16][123].
• *Stochastic learning:* This model uses random processes, probability, and stochastic relaxation to adjust the connection weights. The Boltzmann Machine [2][57][104] uses stochastic learning based on a process called simulated annealing [41][68].

### 1.2 An Overview of the Dissertation

Neural network models are specified by network architecture, node characteristics, and learning rules. Today, many different models of neural networks are available, and many new models will be invented in the future. In order to solve real-world problems, where massive parallelism is essential for high performance, there is a need to develop techniques to allow these models to be implemented on VLSI hardwares. It is also important to develop good learning algorithms that are fast, reliable and robust, and can maximize the potential capabilities of networks.

This dissertation discusses an iterative learning technique, called the *relaxation method*, its mathematical foundation and its applications to well-known neural network models such as the Hopfield model [62], Bidirectional Associative Memory [76], Temporal Associative Memory [6], Brain-State-in-a-Box [10], functional-link net [90], Radial Basis Function network [97][19][85][96][101], etc. This research has been carried out to develop a better learning technique for those models and to establish a new and effective learning paradigm.

The first half of Chapter 2 presents a brief history of neural network research and the rest pertains to those articles of immediate relevance to the subject matter of this thesis. Chapter 3 describes the relaxation method and its special variation, called the *pseudo-relaxation method*, which is suitable for parallel implementations with
guaranteed convergence. The relaxation method is an error-correction algorithm that iteratively adapts network parameters by comparing the response to a given input with the desired response to improve performance. There are many existing iterative learning algorithms and they are mostly based on the gradient descent technique. However, the relaxation method is significantly different in several respects:

• *Non-gradient-descent-based technique:* The mathematical basis of the relaxation technique is not the gradient descent which serves as a common basis for many existing iterative learning algorithms. The gradient descent method is a *heuristic* algorithm with many difficulties in its implementation including no guarantee of finding a solution. The relaxation method is a *deterministic* algorithm that directly solves a system of linear inequalities. A major benefit of using the relaxation technique is that it will not find a solution only if there is no solution.

• *No local minimum:* One of the major difficulties in conventional gradient-descent-based algorithms is the danger of getting trapped in a local minimum which is not a true solution. Since the gradient descent method is never guaranteed to find a global minimum, a careful choice of the initial network configuration and other learning parameters is vital to avoid local minima. In our formulation of learning, no concept of function minimization is introduced. Consequently, getting stuck in a local minimum will never occur in the new approach.

• *Dynamic learning rate:* The learning rate determines the degree of weight adjustment. Typically, gradient-descent-based techniques employ a *fixed* learn-
ing rate. For some other techniques \cite{99,20,111,36,40}, the fixed learning rate is mandatory because of guaranteed convergence. In general, a small learning rate is safe, but it slows down the learning unacceptably. On the other hand, if the learning rate is too large, the algorithm may oscillate. The choice of an appropriate learning rate is problem dependent and often very difficult to determine. The relaxation method utilizes a dynamically varying adjustment technique, which, in turn, contributes to fast and guaranteed convergence.

In Chapter 4, new iterative learning algorithms for various neural network models are derived based on the relaxation technique. One important issue which should never be overlooked in the design of a learning algorithm is its suitability for a neural network implementation. For example, Kanter and Sompolinsky’s algorithm \cite{67} for the Hopfield model failed to gain popularity because of its nonlocal updating procedures. The same difficulty arises with Wang, Cruz and Mulligan’s method \cite{113} for Bidirectional Associative Memory because of its inability to provide a parallel implementation. The learning algorithms derived in Chapter 4 are well-suited for a neural network implementation:

- They can be easily implemented in parallel and the internal parameters are adapted based on locally available information.

- They are computationally feasible for a large problem because the amount of local memory required is relatively fixed regardless of the sample size.

- \textit{On-line} learning is possible because each sample is presented one by one in a systematic manner during the training phase.
Various computer simulations are presented in Chapter 5. The simulation study addresses the following performance issues:

- **Learning speed**: The currently available iterative learning algorithms are often prohibitively slow, especially, for a large number of training samples. The major cause of the slowness is their inability to choose appropriate learning parameters, especially the learning rate. If a technique can utilize a varying learning rate, it can outperform the conventional methods. The relaxation method does utilize a varying learning rate. To provide concrete evidence, the speed of newly developed learning algorithms is carefully measured by the number of training iterations and compared with existing learning techniques.

- **Parameter sensitivity**: Parameter sensitivity is an important issue in iterative learning algorithms. The purpose of this study is to provide an in-depth analysis of newly developed algorithms and to suggest a guide line for the subsequent users of the algorithms. Our simulation study includes (i) the effect of the initial weights on learning and (ii) the sensitivity of learning to the learning parameters.

- **Scalability**: An algorithm is scalable if the learning speed remains constant when the number of training vectors and the number of processing elements are both increased in the same proportion. The scalability of an algorithm is clearly an important advantage for large applications. Along with the parameter sensitivity, the scalability issue is addressed in Chapter 5.
2. LITERATURE SURVEY

The first effort in mathematical neural modeling dates back to the work in 1940 by McCulloch and Pitts [80]. The McCulloch-Pitts model is not a complete physiological model of an actual nerve cell but rather a simple two-state representation of the logical processing that occurs in nerve cells. Theoretically, the McCulloch-Pitts model could be configured to compute any Boolean function, but the model failed because of the lack of proper learning capability. In the late 1950's, Rosenblatt [99] introduced a model of neural networks called the perceptron. The perceptron is a single-layer network based on the original McCulloch-Pitts model, but its introduction was significant because of the network's learning capability. Rosenblatt proved in his book [99] that the perceptron can learn anything that it can compute by adjusting the weights between the input and output layers in proportion to the error between the desired and actual outputs. This learning theorem, also known as the perceptron convergence theorem, was clear and simple enough to attract attention of many researchers until attacked by Minsky and Papert in their book, *Perceptrons* [84]. Minsky and Papert showed that the computational power of the perceptron is severely limited by arguing that perceptrons would only work for linearly separable problems. Their criticism contributed to a major setback in further neural-network research.
An early example of applications of neural networks is the Adaline (adaptive linear element) proposed by Widrow [116][118]. The Adaline is similar to the perceptron in its configuration and adjusts the weights between the input and output layers based on the error between the actual and desired outputs. The Adaline is being successfully used for adaptive signal processing [122], control systems [119], and adaptive antenna systems [120]. Since the Adaline has similar configuration to the perceptron, it shares the perceptron's drawback of linear separability of input classes. A multiple adaptive linear element, or Madaline [118][60], was attempted to overcome the functional difficulties in the Adaline. However, the weight adjustments in the Madaline are only possible on the connections between the input and hidden layers because of the lack of a proper learning algorithm.

During the 1970s and 1980s, the study of neural networks stepped forward to a new era with Grossberg [46][47][48], Amari [4][5][6], Kohonen [71][72], Fukushima [37][38], Hopfield [62][64], Rumelhart and the PDP group [100], and many others. Grossberg studied self-organizing and self-stabilizing neurodynamic systems including instar, outstar and avalanche [46][47][48]. Grossberg’s work also includes competitive learning [49][27], adaptive resonance theory [50], and the ART implementation [22][24][25]. Recently, ART networks were further characterized and their dynamic properties were extensively studied [23][52]. Amari pioneered the work on a rigorous mathematical foundation for neurodynamic systems based on the earlier McCulloch-Pitts model. His studies include the mathematical analysis of associative memory [7][8]. Several models of associative memory were also proposed by Kohonen [71][72] and Anderson [10][11]. Fukushima worked on a model of neural networks called neocognitron and demonstrated the use of his model for recognizing handwritten
characters [38].

In 1982, Hopfield created immense interest in the field of neural networks by introducing a class of neural networks [62], often referred to as the Hopfield model. The Hopfield model can be used as a content-addressable memory or to solve combinatorial optimization problems [65]. Using energy functions to describe the global dynamics of the system, Hopfield added to the McCulloch-Pitts model an alternative stability procedure, and proposed implementations on analog circuits. Although his description of the system can also be found in the earlier work by Amari [7][8], Hopfield is credited for initiating the recent interest in the area of neural networks. The Hopfield model offers many interesting features that are not found in conventional digital memory: (i) distributed representation, (ii) parallel and distributed control, (iii) content-addressability, and (iv) fault tolerance. The Hopfield model utilizes a simple learning rule called the first-order correlation encoding scheme, a simple mathematical form of Hebbian learning [55]. Despite the exciting features of the Hopfield model, the networks prescribed by the first-order correlation encoding scheme are confronted by several difficulties in real applications: (i) low information capacity, (ii) lack of ability to recall all training samples, and (iii) lack of ability to recover gracefully from connection failures.

The information capacity of the Hopfield model has been studied extensively by many researchers [8][1][9][114][110][81][78], and proven to be very limited. Moreover, the correlation encoding scheme does not guarantee a perfect recall of all training samples unless they are encoded as orthogonal vectors [1]. Several attempts have been made to find alternatives to the correlation encoding scheme. Kanter and Sompolinsky [67], and Wong [125] were able to store linearly independent patterns. Crick
and Mitchison [30] proposed a hypothesis stating that the unnecessary memories in the brain are removed during the REM sleep by an unlearning mechanism. This idea was pursued by Hopfield, Feinstein and Palmer [63]. Improvements in the storage density of Hopfield networks are possible by combining the concept of unlearning with the correlation encoding scheme [69]. Kanter and Sompolinsky [67] show that $N$ linearly independent patterns can be stored in an $N$-node network using a Hamiltonian version of the network of Personnaz et al. [93]. Wong [125] proposed another technique which permits the recognition of linearly independent patterns.

While the Hopfield model can be classified as an autoassociative memory, Kosko's bidirectional associative memory (BAM) [74][75][76] is a heteroassociative memory that can store and recall pattern pairs. The BAM is a two-layer network and can be viewed as an extension of the single-layer Hopfield network, using popular Hebbian learning which was also used in Grossberg's ART systems [46][47][48] and the Hopfield model. Kosko [76] proved that the BAM is always bidirectionally stable regardless of connection strength between the two layers. The BAM shares similar difficulties as those arising in the Hopfield model. The most prominent limitation of the BAM is its low information capacity. Furthermore, unless the training samples are orthogonal, the recall of all training pairs is not guaranteed. Several learning schemes have been proposed for improving the storage capacity of the BAM. Simpson [106] proposed an orthogonal encoding of the training samples. Parker [92] developed a variation of second-order Hebbian learning. Wong, Cruz and Mulligam [112] proposed a multiple training concept that makes the energies of the training vectors to be the local minima in a given system. Recently, Wang et al. [113] suggested the use of linear programming technique for learning in the BAM.
The discovery of the backpropagation algorithm [100] by Rumelhart, McClelland, and the PDP group at MIT was another forward step in neural-network research. In fact, the backpropagation algorithm was derived independently by several other researchers [115][77][91], but its popularity is mainly due to the efforts of the PDP group. The backpropagation algorithm solves the problems that the perceptron algorithm cannot handle, i.e., multi-layer networks can be trained to perform an arbitrary non-linear mapping. Today, the number of applications employing the backpropagation algorithm has increased. For example, Sejnowski and Rosenberg [105] used the backpropagation algorithm to develop NETtalk, a program which learns to read aloud English text without the benefit of any preprogrammed linguistic rules. NETtalk provided an impressive demonstration of the potential of neural-network technology.

The backpropagation algorithm relies on a surprisingly simple mathematical technique called the gradient descent method which is a heuristic procedure to find a local minimum of a given criterion function. The backpropagation algorithm has many practical difficulties. No guarantee of finding a solution and extremely long training times are typically encountered in many applications. Another difficulty with the backpropagation networks is determining the number of layers and the number of hidden nodes necessary to perform an accurate mapping. If a network has a simpler structure than necessary, it simply cannot learn the desired mapping. On the other hand, if a network has a more complicated structure than necessary, it will lead to poor generalization. Generative feedforward architectures [61][12][35] have been proposed to determine the network topology adaptively, relying on incremental addition of hidden nodes. Honavar and Uhr [61] proposed the generation method
that modifies the network topology by growing links and recruiting nodes whenever performance stops improving during weight adjustment using the backpropagation algorithm. Another strategy called the *tiling algorithm* was introduced by Mézard and Nadal [82] to learn any binary function. In their approach, new hidden nodes and layers are added to the network to give a strictly better approximation of the desired output than the previous one. The *dynamic node creation* methods proposed by Ash [12] and Hirose *et al.* [58] add new hidden nodes to the layer based on the rate of decrease of error. Both the methods are only applicable to networks with one hidden layer. Fahlman’s *cascade-correlation architecture* [35] begins with no hidden nodes and eventually constructs a multi-layer network with a cascade of hidden nodes. If the desired mapping cannot be learned by the current configuration of a network, a hidden node is added and trained while the previously trained weights are frozen. Each new hidden node receives a connection from each of network’s original inputs and also pre-existing hidden nodes.

Recently, the Radial Basis Function (RBF) technique for interpolating in a high dimensional space was proposed by Broomhead and Lowe [19] and others [96][85] to provide an alternative tool to learning in feedforward networks. The study of Radial Basis Functions was motivated by a review article by Powell [97] and the theoretical contribution of Micchelli [83]. The important property of this class of networks is that they can be learned by solving a set of linear equations. This is quite different from the backpropagation network whose learning rule performs complicated non-linear function optimization. The generalization performance of RBFs was studied by Botros and Atkerson [18] and Wong [126]. Moody [85] and Kadirkamanathan *et al.* [66] proposed learning algorithms for RBF networks for predicting chaotic time
series. Moody [85], Saha and Keeler [101], and Musavi et al. [88] used clustering algorithm for determining the basis function centers. Other researchers have used the RBF networks to solve practical problems including phoneme classification [98] and image coding and analysis [102].
3. THE RELAXATION METHOD

3.1 Introduction

This chapter describes a mathematical technique called the relaxation method for solving systems of linear inequalities. Linear inequalities were extensively studied in the early 20th century by several researchers [31][32][33][17]. These studies were primarily concerned with the development of the theory of linear inequalities including conditions for the existence of a solution. Numerical techniques for obtaining solutions were later developed with the invention of digital computers. The relaxation method for solving systems of linear inequalities was first introduced by Agmon [3] and Motzkin and Schoenberg [87]. The early development of other methods can also be found in [89]. The term “relaxation” has been used to name this method because it uses the same orthogonal projection that is used in the relaxation method for solving systems of equations [109]. The former method could be considered to be an extension of the latter method to inequalities.

The relaxation method is an iterative procedure which attempts to solve a system of linear inequalities by examining one inequality at a time. Starting with an arbitrary vector, a halfspace (defined by one of the linear inequalities of the system) is examined during each iteration to check whether it contains the current vector. If it does not, the vector is changed by moving in the direction of the inner normal to the halfspace.
Each iteration guarantees that the newly obtained vector is always closer to a solution than the previous vector if the amount of movement is bounded to a certain range. The range is a varying quantity and depends on the relative position of the current vector to the hyperplane.

Details of the general relaxation procedure and its mathematical properties are presented in Section 3.2. In Section 3.3, the maximal distance relaxation method, an implementation of the relaxation procedure which was originally described in [3][87], and its results are provided. The maximal distance relaxation method is known to converge geometrically if there exists a solution. Although the method solves general systems of linear inequalities, it has two major drawbacks: (i) it may take infinitely many steps to reach a solution (ii) it must maintain global information which is not desirable for parallel and distributed processing. A variation of the maximal distance relaxation method, called the pseudo-relaxation method, is proposed in Section 3.4 to overcome such drawbacks when implemented on neural networks. The pseudo-relaxation method always converges in a finite number of steps to a solution and its calculation is solely based on local information. Thus, it avoids the difficulties posed by the maximal distance relaxation method. The proof of convergence of the pseudo-relaxation method is also given in the same section.

### 3.2 The Relaxation Method

The notation $\langle \cdot, \cdot \rangle$ is used to denote the inner product of two vectors, and $\| \cdot \|$ to denote the Euclidean norm. Consider a consistent system of $m$ linear inequalities

$$\langle a^i, x \rangle + b^i \geq 0 \quad \text{for } i = 1, \ldots, m \quad (3.1)$$
where \( a^i \in \mathbb{R}^n \), \( b^i \in \mathbb{R} \) and \( x \in \mathbb{R}^n \) is a variable vector. Each inequality defines a halfspace \( H^i \) in \( \mathbb{R}^n \):

\[
H^i = \{ x \in \mathbb{R}^n | (a^i, x) + b^i \geq 0 \} \tag{3.2}
\]

The feasible solution set for (3.1) is a convex polyhedron given by:

\[
C = \bigcap_{i=1}^{m} H^i \tag{3.3}
\]

To find a point in \( C \), the relaxation method relies on a simple geometric property. Let \( x_q \) be a point in \( \mathbb{R}^n \) such that \( x_q \notin H^i \) for some \( i \). Let \( x_p^q \) be the orthogonal projection of \( x_q \) on the hyperplane defined by \( H^i \) as illustrated in Figure 3.1. The relaxation procedure chooses the next point \( x^{q+1} \) as follows:

\[
x^{q+1} = x^q + \lambda (x_p^q - x^q) \tag{3.4}
\]

where the relaxation factor \( \lambda \) is a constant between 0 and 2. Let \( d(x, H^i) \) denote the Euclidean distance between \( x \) and \( H^i \). The relaxation method is partly based on the following geometric lemma.

**Lemma 3.2.1** Let \( x_q \in \mathbb{R}^n \) and \( x_q \notin H^i \). Let \( x^{q+1} \) be a new point obtained by the relaxation procedure (3.4). Then, if \( \lambda \in [0, 2] \), \( \forall x^* \in H^i \)

\[
\begin{align*}
(i) \quad \|x^{q+1} - x^*\|^2 & \leq \|x^q - x^*\|^2 - \lambda(2 - \lambda)d^2(x_q, H^i) \\
(ii) \quad \|x^{q+1} - x^*\| & \leq \|x^q - x^*\| 
\end{align*} \tag{3.5} \tag{3.6}
\]

where equality for (i) holds only for \( \lambda = 0 \) or \( x^* \) on the boundary of \( H^i \), and equality for (ii) holds only for \( \lambda = 0 \) or \( \lambda = 2 \) and \( x^* \) on the boundary of \( H^i \).
Proof

\[ \|x^{q+1} - x^*\|^2 = \|x^q + \lambda(x^q_p - x^q) - x^*\|^2 \]

\[ = \|x^q - x^*\|^2 + \lambda^2 \|x^q_p - x^q\|^2 + 2\lambda(x^q - x^*, x^q_p - x^q) \]

\[ = \|x^q - x^*\|^2 + \lambda^2 \|x^q_p - x^q\|^2 + 2\lambda(x^q - x^q_p + x^q_p - x^*, x^q_p - x^q) \]

\[ = \|x^q - x^*\|^2 - \lambda(2 - \lambda)\|x^q_p - x^q\|^2 + 2\lambda(x^* - x^q_p, x^q - x^q_p) \]

Since the angle between the two vectors \((x^* - x^q_p)\) and \((x^q - x^q_p)\) is greater than or equal to \(90^\circ\),

\[ \langle x^* - x^q_p, x^q - x^q_p \rangle \leq 0 \]

It follows that

\[ \|x^{q+1} - x^*\|^2 \leq \|x^q - x^*\|^2 - \lambda(2 - \lambda)\|x^q_p - x^q\|^2 \]

Since \(\|x^q_p - x^q\| = d(x^q, H^i)\), (i) holds. The equality for (i) trivially holds when \(\lambda = 0\). When \(x^*\) is on the boundary of \(H^i\), the equality in (i) also holds because \(\langle x^* - x^q_p, x^q - x^q_p \rangle = 0\).

If \(\lambda \in [0, 2]\), then \(0 \leq \lambda(2 - \lambda) \leq 1\). It follows that

\[ \|x^{q+1} - x^*\|^2 \leq \|x^q - x^*\|^2 \]

Thus, (ii) holds. Since \(\|x^{q+1} - x^*\|^2 = \|x^q - x^*\|^2 - \lambda(2 - \lambda)\|x^q_p - x^q\|^2 + 2\lambda(x^* - x^q_p, x^q - x^q_p)\), the equality in (ii) holds only for \(\lambda = 0\) or \(\lambda = 2\) and \(x^*\) on the boundary of \(H^i\). \(\square\)
Corollary 3.2.1 Let $x^q \in \mathbb{R}^n$ and $x^q \not\in H^i$. Let $x^{q+1}$ be the point obtained by the relaxation procedure (3.4). Then, $\forall x^* \in C$, both the inequalities (3.5) and (3.6) hold.

Proof $x^* \in C \Rightarrow x^* \in H^i$ and the proof follows from Lemma 3.2.1. \hfill \Box

Corollary 3.2.1 implies that if $\lambda \in (0, 2)$, then the relaxation procedure (3.4) guarantees that $\forall x^* \in C, \|x^{q+1} - x^*\| < \|x^q - x^*\|$. Note that $(x^q_p - x^q)$ in (3.4) can be substituted by $\frac{|(a^i, x^q) + b^i|}{\|a^i\|^2} a^i$. To solve the system of inequalities (3.1), the general relaxation procedure iteratively performs the following operation:

$$x^{q+1} = x^q - \lambda \frac{(a^i, x^q) + b^i}{\|a^i\|^2} a^i \quad (3.7)$$

by choosing a sequence of $H^i$ such that $d(x^q, H^i) > 0$ and $0 < \lambda < 2$. We will call the sequence of points \{x^q\} a relaxation sequence. Figure 3.2 shows an example of a relaxation sequence.
Figure 3.2: An example of an over-relaxation sequence

The method is called under-relaxation if $\lambda \in (0, 1)$, over-relaxation if $\lambda \in (1, 2)$, or the projection method if $\lambda = 1$. Figure 3.3 illustrates examples of the various relaxation procedures. Note that over-relaxation guarantees $x^{q+1} \in H^i$ after a single iteration of (3.7). Agmon [3] observed that over-relaxation accelerates the convergence since overprojecting has an effect of opening the “solid angle” of the convex polyhedron $C$.

3.3 The Maximal Distance Relaxation Method

The relaxation procedure (3.7) is known to converge geometrically if $\lambda \in (0, 2)$ and the relaxation sequence is chosen properly. Define

$$d_{max}(x) = \max\{d(x, H^i) | i = 1, \ldots, m\}$$  \hspace{1cm} (3.8)

If the relaxation sequence is such that $d(x^q, H^i) = d_{max}(x^q)$, then the procedure (3.7) is called the maximal distance relaxation method. Agmon [3] and Motzkin
Figure 3.3: Various relaxation procedures

(a) under-relaxation

(b) projection

(c) over-relaxation
and Schoenberg [87] have proven convergence of the maximal distance relaxation method. The results of Agmon, Motzkin and Schoenberg are summarized in the next three theorems.

**Theorem 3.3.1 (Agmon)** Let $C = \bigcap_{i=1}^{m} H^i$ be the feasible solution set for the system of $m$ linear inequalities (3.1), and let $\{x^q\}$ be the sequence of points obtained by the maximal distance relaxation method. Then the sequence $\{x^q\}$ converges to a solution $x^* \in C$. Furthermore,

$$\|x^q - x^*\| \leq 2d(x^0, C)\rho^q,$$

where $0 < \rho < 1$ and $d(x^0, C)$ is the Euclidean distance between $x^0$ and the polyhedron $C$.

The proof of Theorem 3.3.1 is somewhat lengthy, but the main idea of the proof is that the ratio between the distance of $x^q$ from the nearest solution and the distance from the farthest hyperplane is bounded. Agmon showed that $\forall x \not\in C, \exists \mu > 0$ such that

$$\mu \leq \frac{d_{\text{max}}(x)}{d(x, C)} \leq 1 \quad (3.9)$$

**Theorem 3.3.2 (Motzkin & Schoenberg)** Assume that $C = \bigcap_{i=1}^{m} H^i$ is full dimensional, i.e., not contained in any hyperplane of $H^i$. Let $\{x^q\}$ be the sequence of points obtained by the maximal distance relaxation method. Then there are two cases:

1. If $0 < \lambda < 2$ either $\{x^q\}$ terminates or $\{x^q\}$ converges to a point on the boundary of $C$. 
2. If $\lambda = 2$ the sequence $\{x^q\}$ always terminates.

Theorem 3.3.3 (Motzkin & Schoenberg) Assume that $C = \bigcap_{i=1}^{m} H^i$ is not full dimensional. Let $\{x^q\}$ be the sequence of points obtained by the maximal distance relaxation method. Then there are two cases:

1. If $0 < \lambda < 2$ either $\{x^q\}$ terminates or $\{x^q\}$ converges to a point of $C$.

2. If $\lambda = 2$ the sequence $\{x^q\}$ either terminates or there exists a number $q'$ such that for $q \geq q'$ the points $x^q$ are on a spherical surface having the affine hull of $C$ as its axis.

As stated in the previous two theorems, the maximal distance relaxation method may require infinitely many iterations to reach a solution. Figure 3.4 illustrates such examples.

![Figure 3.4: Non-terminating relaxation sequences](image)

3.4 The Pseudo-Relaxation Method

The pseudo-relaxation method solves two major difficulties posed by the maximal distance relaxation method without sacrificing the high performance offered by the
relaxation procedure. It guarantees termination of the relaxation sequence \( \{x^q\} \). It is also suitable for parallel and distributed processing, especially for a neural network implementation.

The pseudo-relaxation method cycles through the sequence of halfspaces \( \{H^i\} \) and performs the relaxation procedure (3.7) if \( d(x^q, H^i) \geq \delta^i \) for some predetermined \( \delta^i > 0 \). This is different from the maximal distance relaxation method where \( d(x^q, H^i) = d_{\max}(x^q) \). The pseudo-relaxation method does not necessarily give a solution for (3.1). Instead, when it terminates, \( x^q \) is in \( \delta^i\text{-neighborhood} \) of \( H^i \), i.e., \( \forall i \ d(x^q, H^i) < \delta^i \). The convergence of the pseudo-relaxation method is established in the next theorem. The proof of convergence is much simpler compared to Agmon's proof for the maximal distance relaxation method.

**Theorem 3.4.1** Let \( \{x^q\} \) be the sequence obtained by the pseudo-relaxation method. If \( \lambda \in (0,2) \), then the sequence \( \{x^q\} \) always terminates.

**Proof** Let \( \delta_{\min} = \min\{\delta^i \mid i = 1, \ldots, m\} \). It is sufficient to show that for a given \( x^0 \in \mathbb{R}^n \), \( \exists q \) such that \( \forall i \ d(x^q, H^i) < \delta^i \). From Corollary 3.2.1, \( \forall x^* \in C \)

\[
\|x^q - x^*\|^2 \leq \|x^q-1 - x^*\|^2 - \lambda(2 - \lambda)d^2(x^q-1, H^i)
\]

Since \( d(x^q-1, H^i) \geq \delta^i \geq \delta_{\min} \), it follows that

\[
\|x^q - x^*\|^2 \leq \|x^q-1 - x^*\|^2 - \lambda(2 - \lambda)\delta_{\min}^2
\]
Repeated application of the above inequality gives us

\[ \|x^q - x^*\|^2 \leq \|x^{q-1} - x^*\|^2 - \lambda(2 - \lambda)\delta_{\text{min}}^2 \]

\[ \leq \|x^{q-2} - x^*\|^2 - 2\lambda(2 - \lambda)\delta_{\text{min}}^2 \]

\[ \vdots \]

\[ \leq \|x^0 - x^*\|^2 - q\lambda(2 - \lambda)\delta_{\text{min}}^2 \]

Thus, we get

\[ 0 \leq \|x^0 - x^*\|^2 - q\lambda(2 - \lambda)\delta_{\text{min}}^2 \]

Since \(0 < \lambda(2 - \lambda) \leq 1\), \(q\) must be bounded. \(\square\)

**Theorem 3.4.2** Let \(C = \bigcap_{i=1}^{m} H^i\) be the feasible solution set for the system of \(m\) linear inequalities (3.1), and let \(\{x^q\}\) be the sequence of points obtained by the pseudo-relaxation method. If \(\lambda \in (0, 2)\), then \(\forall x^* \in C\)

\[ \|x^q - x^*\| \leq \rho^q\|x^0 - x^*\| \]

where \(0 < \rho < 1\).

**Proof** From Corollary 3.2.1, if \(\lambda \in (0, 2)\), then \(\forall x^* \in C\) and \(\forall p \in \{1, 2, \ldots, q\}\)

\[ \|x^p - x^*\| < \|x^{p-1} - x^*\| \]

Let \(\{H^r(p)\}\) be the sequence of halfspaces selected for projection after \(q\) successive operations of (3.7), where \(r : \{0, 1, \ldots, q\} \rightarrow \{1, 2, \ldots, m\}\). Then, the following strict inequality holds for all \(p\) in \(\{1, 2, \ldots, q\}\):

\[ \frac{d(x^p, H^r(p))}{\|x^0 - x^*\|} < \frac{d(x^p, H^r(p))}{\|x^p - x^*\|} \]
Note that since $x^* \in H^r(p)$,
\[
\frac{d(x^P, H^r(p))}{\|x^P - x^*\|} \leq 1
\]
Let $\delta_{min} = \min\{\delta^i \mid \text{for } i = 1, \ldots, m\}$. Since the pseudo-relaxation method performs
the relaxation procedure only if $d(x^P, H^r(p)) \geq \delta^r(p)$,
\[
d(x^P, H^r(p)) \geq \delta^r(p) \geq \delta_{min} > 0
\]
Thus,
\[
0 < \frac{\delta_{min}}{\|x^0 - x^*\|} \leq \frac{d(x^P, H^r(p))}{\|x^0 - x^*\|} < \frac{d(x^P, H^r(p))}{\|x^P - x^*\|} \leq 1 \quad (3.10)
\]
Let $\mu = \delta_{min}/\|x^0 - x^*\|$. It is easily seen from (3.10) that $0 < \mu < 1$. The
relationship in (3.10) can be summarized as follows:
\[
\mu \|x^P - x^*\| < d(x^P, H^r(p)) \quad (3.11)
\]
From Corollary 3.2.1 and Inequality (3.11), we get
\[
\|x^q - x^*\|^2 \leq (1 - \lambda(2 - \lambda)\mu^2)\|x^{q-1} - x^*\|^2 \quad (3.12)
\]
Let $\rho = \sqrt{1 - \lambda(2 - \lambda)\mu^2}$. Since $0 < \mu < 1$ and $0 < \lambda(2 - \lambda) \leq 1$, clearly $0 < \rho < 1$. Thus,
\[
\|x^q - x^*\| \leq \rho\|x^{q-1} - x^*\| \quad (3.13)
\]
It follows that
\[
\|x^q - x^*\| \leq \rho^q\|x^0 - x^*\|
\]
This completes the proof. \(\Box\)

We established a finite convergence of the pseudo-relaxation method in The­
orem 3.4.1 and a geometric convergence in Theorem 3.4.2. The pseudo-relaxation
method is an efficient technique to find a solution to a system of linear inequali-
ties if the solution set is full dimensional. Assuming the polyhedron $C$ to be full
dimensional, the pseudo-relaxation method works as follows:

**STEP 1** Define $H^i_\zeta$ to be the halfspace in $\mathbb{R}^n$ such that

$$H^i_\zeta = \{x \in \mathbb{R}^n | \langle a^i, x \rangle + b^i - \xi_i \geq 0\}$$

(3.14)

where $\xi_i > 0$. Let $d(H^i, H^i_\zeta)$ be the perpendicular distance between two hyper-
planes defined by the halfspaces $H^i$ and $H^i_\zeta$. Define $\delta_{\text{max}} = \max_i d(H^i, H^i_\zeta)$.

The solution set to the new system (3.14) is:

$$C_\zeta = \bigcap_{i=1}^m H^i_\zeta$$

If $C_\zeta \neq \emptyset$, then $C_\zeta \subset C$. Note that $C_\zeta \neq \emptyset$ as long as $C$ is full dimensional
and $\delta_{\text{max}}$ is chosen to be sufficiently small. In particular, if $C$ is a convex
polyhedral cone, $C_\zeta$ is not empty regardless of the choice of $\xi_i$.

**STEP 2** Apply the pseudo-relaxation method for solving the system \{ $H^i_\zeta$ \} using

$$\delta^i = d(H^i, H^i_\zeta).$$

**STEP 3** As proved in Theorem 3.4.1, pseudo-relaxation terminates at $x^\theta$, resulting
in $\forall i \ d(x^\theta, H^i_\zeta) < \delta^i$. Since $\delta^i = d(H^i, H^i_\zeta)$ and $C_\zeta \subset C$, $x^\theta \in C$.

An application of the pseudo-relaxation method for a two dimensional case is
illustrated and compared to that of the maximal distance relaxation method in Fig-
ure 3.5. In the illustration, both methods use under-relaxation with the same starting
point $x^0$. As shown in Figure 3.5, the maximal distance relaxation sequence goes
to the corner point in infinitely many steps; whereas the pseudo-relaxation sequence
quickly terminates.
3.5 Concluding Remarks

The relaxation method solves systems of linear inequalities using orthogonal projection. The method is particularly useful for solving large systems, because the amount of memory required is small and fixed regardless of the problem size. Furthermore, there are various minimization problems that can be reduced to a system of inequalities involving no minimization by the duality principle. For example, Herman [56] used the relaxation method for reconstructing objects from noisy X-ray
data. The problem involves solving a large sparse system of linear equations, typically $10^5$ equations with $10^4$ unknowns, with only about 1% non-zero coefficients. Herman solved the problem by converting it into a problem of solving a system of inequalities and demonstrated the reconstruction of a beating heart. The pseudo-relaxation method has two distinct advantages which are lacking in the maximal distance relaxation method: a finite convergence and suitability for a neural network implementation. Efficient learning algorithms for various neural network models can be derived based on the pseudo-relaxation method. We defer further discussion of its applications until the next chapter.
4. APPLICATIONS OF THE RELAXATION METHOD TO LEARNING IN NEURAL NETWORKS

4.1 Introduction

In neural networks, learning is a process of adaptively changing its weights to improve performance of the network. The quality of a learning algorithm is often measured by the following factors:

- **Learning speed**: What is the time required to learn a given set of training data until desired performance is achieved?

- **Parameter sensitivity**: How significantly does the performance of the learning algorithm vary with the choice of certain parameters associated with the algorithm?

- **Generalization**: How well does the network respond to unknown data after training?

The learning speed turns out to be a major source of difficulty in some applications. The currently available iterative learning algorithms based on the gradient descent technique are often prohibitively slow. The major cause of the slowness is their inability to choose appropriate learning parameters such as gain factor, which is
the constant of proportionality representing the learning rate $[100]$. The performance of a learning algorithm can vary significantly depending on the parameters. The existing methods for selecting the parameters are mostly based on trial and error, and choosing a good set of parameter values can prove to be very difficult. How well the network generalizes is largely dependent on the network topology, the training data size and the initial configuration of the network. We do not yet have a method to determine an adequate network topology and the minimal training data size for solving a given problem. Currently, generative learning algorithms $[61][12][35]$ are being investigated to address these issues, but a significant portion of their success still remains an empirical art.

This chapter presents new iterative learning algorithms for various neural network models: the Hopfield model $[65]$, Bidirectional Associative Memory (BAM) $[76]$, Dynamic Heteroassociative Neural Memory (DAM) $[54]$, and Radial Basis Function (RBF) Networks $[97][19][96]$. The learning problem in those models can be reduced to a problem of solving a system of linear inequalities. A similar mathematical formulation is possible for other well-known neural network models such as Temporal Associative Memory$[6]$, Brain-State-in-a-Box$[10][11]$, Cascade Correlation Architecture$[35]$, and Functional Link Net $[90]$. The relaxation method is used to solve the system of linear inequalities. The newly developed learning algorithms are quite different from the existing iterative algorithms. The relaxation technique makes a dynamically varying adjustment to the weights based on the training data being examined and on other considerations arising from the relaxation procedure. With this approach, it is not necessary to make only small adjustments to weights, which in turn significantly improves the training time. The relaxation technique is guaranteed
to find an appropriate set of weights, if such a set of weights exists. In other words, the technique fails only if the network is not capable of learning the desired mapping.

Recently, Wang et al. [113] have discussed a similar formulation of the learning problem in the BAM and suggested the use of a linear programming technique for learning. The objective of any typical linear programming technique is to find a feasible solution which is a corner point of a given convex set. In our formulation of the learning problem, the objective is to find a point not on the boundary but strictly inside the convex set. In the case of linear programming the solution set is "small", i.e., a set of measure zero, whereas in the case of our learning problem the solution set is "large", i.e., a full dimensional convex set which has a strictly positive measure [53]. The difference is indeed important in our context if it can be successfully exploited to find a fast learning algorithm that utilizes the latitude offered by the "large" solution set.

4.2 The Hopfield Model

An attempt to build brain-like memory systems was made by Hopfield [62] when he proposed a neural network as a theory of memory. A Hopfield network, as shown in Figure 4.1, is a single-layer, fully connected, symmetric, non-linear autocorrelator that stores and recalls binary (or bipolar) patterns. The Hopfield model offers many interesting features which are not found in conventional digital memory:

1. Distributed representation: Information is distributed over the connections between processing elements and memories are superimposed on the same media. Furthermore, information retrieval time is independent of the number of patterns stored in the network, which in turn contributes to a relatively fast
access time.

2. **Parallel and Distributed control:** The control is not governed by a central processing unit. Rather, each processing element makes its own decision based on the local information.

3. **Content-addressability:** The network can retrieve information based on a portion of content. It can also provide a close match for a noisy input.

4. **Fault tolerance:** With a few component failures, the network will still function properly.

![Figure 4.1: A Hopfield network](image)

#### 4.2.1 Network stability

Consider a Hopfield network with $N$ nodes. A state of the network is represented by a vector $s = (s_1, s_2, \ldots, s_N)$, where $s \in \{-1, +1\}^N$. Let $W = [W_{ij}]$ be the weight matrix, where $W_{ij}$ is the connection weight between the $i$-th and the $j$-th
node, and $\theta_i$ be the threshold (or bias) for the $i$-th node. Information stored in the network is retrieved by repeated application of the following state transition rule until the network stabilizes:

$$s_i \leftarrow \begin{cases} 
1 & \text{if } \sum_{j=1}^{N} W_{ij} s_j - \theta_i \geq 0 \\
-1 & \text{if } \sum_{j=1}^{N} W_{ij} s_j - \theta_i < 0 
\end{cases} \quad (4.1)$$

When all $s_i$ remain unchanged, the network is said to be stable. Applications of Hopfield networks seek to make the stable states correspond to solutions to the given problem. An associative memory application seeks to make the stable states correspond to the patterns being stored.

Each successive state of the network is computed from the current state by applying the update rule to a set $S$ of the nodes of the network. Different modes of operation are possible depending on the choices for set $S$ selected for each update. If only one node is selected at a time then the network is said to operate in a serial mode. If all the nodes are updated at the same time, i.e. $|S| = N$, then the network is said to operate in a fully parallel mode. All the other modes of update with $1 < |S| < N$ will be called parallel modes. The set $S$ can be chosen at random or according to some deterministic rule. The stability of the Hopfield model under various modes of operation has been investigated by several researchers [62][42][43][21]. Their results are summarized in the following theorems.

**Theorem 4.2.1 (Hopfield)** A Hopfield network with a serial mode of operation always converges to a stable state if $W$ is a symmetric matrix with the elements of the diagonal being nonnegative.
Theorem 4.2.2 (Goles et al.) A Hopfield network with a fully parallel mode of operation always converges to a stable state or to a cycle of length 2 in the state space if $W$ is a symmetric matrix.

Theorem 4.2.3 (Goles) A Hopfield network with a fully parallel mode of operation always converges to a cycle of length 4 in the state space if $W$ is a antisymmetric matrix with zero diagonal.

An energy function was introduced by Hopfield to facilitate the study of convergence and other properties of the network. Hopfield [62] and Goles et al. [42] have used the following discrete-time Lyapunov energy function:

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij}X_iX_j + \sum_{i=1}^{N} \theta_iX_i$$  \hspace{1cm} (4.2)

The energy function is a quadratic mapping from the state space to $\mathbb{R}$, the set of real numbers. Note that the energy function is uniquely defined by the set of weights and the threshold values. Conversely, an energy function uniquely defines weights and the threshold values. A stable state of a Hopfield network corresponds to a local minimum of the energy function.

The Hopfield model can be viewed as a neural network that performs a local search for a minimum of a quadratic optimization function defined over the state space. Thus, the Hopfield model can be applied to solve a class of optimization problems which can be represented by a quadratic function. Hopfield and Tank [65] have illustrated the use of the network for solving the traveling salesperson problem. Following Hopfield's work, several studies were done to investigate solutions of combinatorial optimization problems using neural networks.
4.2.2 Correlation encoding

Hopfield used first-order correlation encoding, a discrete approximation of Hebbian learning [55], for storing patterns in the network. Let $T = \{X^{(k)}\}_{k=1,...,P}$ be a set of training vectors, where each $X^{(k)} \in \{-1, +1\}^N$. The first-order correlations are stored in the weight matrix using the equation

$$W_{ij} = \begin{cases} \sum_{k=1}^{P} X_i^{(k)} X_j^{(k)} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{4.3}$$

Equation (4.3), however, provides a very low storage density. In addition, it does not guarantee the correct recall of all patterns unless they are encoded as orthogonal vectors [1]. The information capacity of the Hopfield model has been studied extensively by many researchers [8][1][9][114][110][81][78]. For random input patterns the information capacity is at most $0.147N$, with an allowance for a small margin of error in the output vector, where $N$ is the number of processing elements in the network. Without any error in the output vector, the capacity can be no more than $N/(4 \ln N)$ asymptotically as $N$ approaches infinity. Thus, the larger the number of processing elements, the worse is the utility factor, defined as the ratio of the capacity to the number of processing elements. Moreover, the information capacity is expected to be even smaller for correlated input patterns.

4.2.3 Formulation of learning

The purpose of learning is to find an appropriate weight vector $w = (W_{ij}, \theta_i)$ that allows the given set of vectors to be stored in the network as stable states. The
state transition rule (4.1) implies that every vector in $T$ is a stable state if and only if for $k = 1, \ldots, P$ and for $i = 1, \ldots, N$, 

$$\sum_{j=1}^{N} W_{ij} X_j^{(k)} - \theta_i \geq 0 \quad \text{if } X_i^{(k)} = 1$$

$$\sum_{j=1}^{N} W_{ij} X_j^{(k)} - \theta_i < 0 \quad \text{if } X_i^{(k)} = -1$$

(4.4)

The inequalities (4.4) provide a characterization of stable states. The characterization is the same for both the serial and the fully parallel mode of operation.

Lemma 4.2.1 Given $T = \{X^{(k)}\}_{k=1, \ldots, P}$, the system of linear inequalities (4.4) has a solution if and only if the following system of linear inequalities (4.5) has a solution

$$\left( \sum_{j=1}^{N} W_{ij} X_j^{(k)} - \theta_i \right) X_i^{(k)} > 0$$

(4.5)

for $k = 1, \ldots, P$ and for $i = 1, \ldots, N$.

Proof (if) If $w^*$ is a solution to the system (4.5), then trivially it is a solution to the system (4.4).

(only if) Let $w^* = (W_{ij}^*, \theta_i^*)$ be a solution to the system (4.4). Since the set $T$ is finite, $\exists \epsilon > 0$ such that for $k = 1, \ldots, P$ and for $i = 1, \ldots, N$,

$$\sum_{j=1}^{N} W_{ij}^* X_j^{(k)} - \theta_i^* + \epsilon > 0 \quad \text{if } X_i^{(k)} = 1$$

$$\sum_{j=1}^{N} W_{ij}^* X_j^{(k)} - \theta_i^* + \epsilon < 0 \quad \text{if } X_i^{(k)} = -1$$
Let \( w' = (W_{ij}^*, \theta_i^* - \epsilon) \). Then, clearly \( w' \) is a solution to the system (4.5).

In view of Lemma 4.2.1, \( T \) is storable if and only if \( \exists w \) that satisfies the system of linear inequalities (4.5). We have formulated learning as a problem of solving a system of strict inequalities. Later we will describe a learning technique which effectively exploits the formulation of learning described here.

Linear separability is another way to view the characterization of stable states by a system of inequalities. A partition \( S = P^+ \cup P^- \) of a set \( S \) in \( \mathbb{R}^N \) is called linearly separable if there exists a hyperplane in \( \mathbb{R}^N \) which separates \( P^+ \) and \( P^- \).

We define partitions \( \{P_i^+, P_i^-\} \) of the training set \( T \) for each \( i = 1, \ldots, N \), as follows:

\[
P_i^+ = \{x(k) \mid x(k) \in T \text{ and } X_i^{(k)} = 1\}
\]

\[
P_i^- = \{x(k) \mid x(k) \in T \text{ and } X_i^{(k)} = -1\}
\]

The condition (4.5) is true if and only if every partition \( \{P_i^+, P_i^-\} \) of \( T \) is linearly separable. Thus, all the vectors in \( T \) can be stored as stable states if and only if every partition \( \{P_i^+, P_i^-\} \) of \( T \) is linearly separable for \( i = 1, \ldots, N \).

4.2.4 An iterative learning algorithm

In the previous section, we formulated learning for Hopfield networks as a problem of solving the system of linear inequalities (4.5). Since the solution set \( C \) is a convex polyhedral cone with vertex at the origin, the pseudo-relaxation method can be applied. If the inequalities (4.5) hold, then the following system of linear inequalities holds for any \( \xi > 0 \):

\[
(\sum_{j=1}^{N} W_{ij} x_j^{(k)} - \theta_i) x_i^{(k)} - \xi \geq 0 
\] (4.6)
Applying the pseudo-relaxation method to the inequalities (4.6) with \( \delta = \xi / \sqrt{N} \) for all halfspaces, we get an iterative learning algorithm as described below (Learning Algorithm LA1). Note that application of the pseudo-relaxation method does not require any restrictions on the weights. Variations of the Hopfield model are possible with different conditions on the weight matrix [21]. However, in this thesis we limit our scope to the original Hopfield model with symmetric weights and zero diagonal.

**Learning Algorithm LA1**

Given a training set \( T = \{X^{(k)}\} \), where \( X^{(k)} = (X_1^{(k)}, \ldots, X_N^{(k)}) \), the following adaptation rule is applied for each \( X^{(k)} \):

\[
\begin{align*}
\Delta W_{ij} &= -\frac{\lambda}{N} [S_i^{(k)} - \xi X_i^{(k)}] X_j^{(k)} \quad \text{if } S_i^{(k)} X_i^{(k)} \leq 0 \\
\Delta \theta_i &= +\frac{\lambda}{N} [S_i^{(k)} - \xi X_i^{(k)}] \quad \text{if } S_i^{(k)} X_i^{(k)} \leq 0
\end{align*}
\]

(4.7)

where \( S_i^{(k)} = \sum_{j=1}^{N} W_{ij} X_j^{(k)} - \theta_i \), \( W_{ij} = W_{ji} \) and \( W_{ii} = 0 \).

In LA1, the weight vector \( w = (W_{ij}, \theta_i) \) and \( \xi \) correspond to \( x \) and \( b^j \) in (3.1), respectively. Since the vector corresponding to \( a \) in (3.1) is bipolar, we get \( ||a||^2 = N \). It can be easily shown that LA1 performs pseudo-relaxation with \( \delta = \xi / \sqrt{N} \).

Consider a halfspace \( H_\xi \) such that

\[ H_\xi : S_i^{(k)} X_i^{(k)} - \xi \geq 0 \]

Let \( w^q \) be the weight vector after \( q \) iterations. Suppose \( w^q \not\in H_\xi \). Then,

\[ d(w^q, H_\xi) = \frac{|S_i^{(k)} X_i^{(k)} - \xi|}{\sqrt{N}} \]
Since the learning rule is applied to \( w^q \) only if \( S_i^{(k)} X_i^{(k)} \leq 0 \), it does perform the relaxation procedure with respect to \( H_\xi \) only if \( d(w^q, H_\xi) \geq \frac{\xi}{\sqrt{N}} \).

LA1 uses the three parameters which need to be preset: the relaxation factor \( \lambda \), the initial vector \( w^0 \) and the constant \( \xi \). \( \xi \) affects the basins of attraction in the network. In general, the larger \( \xi \) is, the larger the basins of attraction are. The effects of other parameters on learning are further studied and presented in the next chapter. Based on our empirical results, LA1 performs well irrespective of the choice of \( w^0 \). Our empirical results are also valid for random choices of \( \xi \), because any pseudo-relaxation sequence can be simulated by a sequence starting with a different initial vector and a different \( \xi \). The justification is established in Lemma 4.2.2. Let \(|\{w^q\}|\) denote the length of the sequence \( \{w^q\} \).

**Lemma 4.2.2** Suppose \( \{w^q\} \) be a pseudo-relaxation sequence obtained by LA1 with \( \xi = \xi_1 \) to solve the system of linear inequalities (4.5). Then for any arbitrary choice of \( \xi = \xi_2 \), there exists a choice of the initial weight vector leading to a pseudo-relaxation sequence \( \{\hat{w}^q\} \) such that \(|\{w^q\}| = |\{\hat{w}^q\}|\).

**Proof** Let \( \xi_2 = \alpha \xi_1 \). Note that \( \alpha > 0 \). Let \( w^0 \) be the initial choice of weight vector which leads to the relaxation sequence \( \{w^q\} \) for \( \xi = \xi_1 \). Consider a new relaxation sequence \( \{\hat{w}^q\} \) obtained by choosing \( \xi = \xi_2 \) and \( \hat{w}^0 = \alpha w^0 \). We will show by induction on \( q \) that \( \hat{w}^q = \alpha w^q \) for all \( q \). Let \( w^q = (W_{ij}^q, \theta^q) \) and \( \hat{w}^q = (\hat{W}_{ij}^q, \hat{\theta}^q) \).
Define
\[ S_i^q = \sum_{j=1}^{N} W_{ij}^q x_j - \theta_i^q \]
\[ \hat{S}_i^q = \sum_{j=1}^{N} \hat{W}_{ij}^q x_j - \hat{\theta}_i^q \]

**Basis** \( \hat{w}^0 = \alpha w^0 \) by choice.

**Induction** Assume that \( \hat{w}^{q-1} = \alpha w^{q-1} \). By the induction hypothesis, \( S_i^{q-1} = \alpha S_i^{q-1} \). It follows that
\[ \hat{W}_{ij}^q = \hat{W}_{ij}^{q-1} - \frac{\lambda}{N} [S_i^{q-1} - \xi_2 x_i] x_j \]
\[ = \alpha W_{ij}^{q-1} - \frac{\lambda}{N} [\alpha S_i^{q-1} - \alpha \xi_1 x_i] x_j \]
\[ = \alpha\{W_{ij}^{q-1} - \frac{\lambda}{N} [S_i^{q-1} - \xi_1 x_i] x_j\} \]
\[ = \alpha W_{ij}^q \]

Similarly, one can show \( \hat{\theta}_i^q = \alpha \theta_i^q \). Since the system (4.5) is homogeneous, if \( w \) is a solution to the system, then so is \( cw \) for any \( c > 0 \). Thus, \( |\{w^q\}| = |\{\hat{w}^q\}|. \]

### 4.3 Bidirectional Associative Memory

Kosko [74][75][76], inspired by previous works [11][71][72][107], proposed a neural network model of bidirectional associative memory (BAM). Kosko’s model consists of two layers of nodes with feedback and symmetric synaptic connections between layers as shown in Figure 4.2.
The BAM behaves as a heteroassociative pattern matcher, storing and recalling pattern pairs. The pattern pairs are stored as bidirectionally stable states of the BAM. The BAM allows the retrieval of stored data associations from incomplete or noisy patterns. When presented with a noisy pattern, the recalling process of the BAM can correctly reconstruct the pattern through a sequence of successive updates until it arrives at a stable state. The feedback mechanism of the BAM helps to filter the noise as a pattern goes through successive updates.
4.3.1 BAM stability

Consider an $N \times M$ BAM with $N$ nodes in the first layer and $M$ nodes in the second layer. Let $W = [W_{ij}]$ be the weight matrix, where $W_{ij}$ is the connection weight between the $i$-th node in the first layer and the $j$-th node in the second layer. Let $\theta_{X_i}$ be the threshold (or bias) for the $i$-th node in the first layer and $\theta_{Y_j}$ be the threshold for the $j$-th node in the second layer. The BAM behaves as a heteroassociative content addressable memory, storing and recalling a set of vector pairs $T = \{(X^{(k)}, Y^{(k)})\}_{k=1,...,P}$, where $X^{(k)} \in \{-1, +1\}^N$ and $Y^{(k)} \in \{-1, +1\}^M$.

The recalling procedure of the BAM is nonlinear and employs inter-layer feedback. Given an initial vector $X$ (or $Y$), the recalling process in the BAM reverberates between its two layers until a stable state is reached in finitely many steps as shown in [75][76]. During the recalling process, nodes update their states using a nonlinear activation function. A node examines its net input of weighted signals from nodes in the other layer, and its state is changed to $+1$ if the net input is greater than the threshold and $-1$ if the net input is less than the threshold. BAM stability has been proven by Kosko [76] using a discrete-time Lyapunov energy function similar to (4.2).

$$E = -\sum_{i=1}^{N} \sum_{j=1}^{M} W_{ij} X_i Y_j + \sum_{i=1}^{N} \theta_{X_i} X_i + \sum_{j=1}^{M} \theta_{Y_j} Y_j$$

(4.8)

Kosko proved that the energy function decreases when the state of the BAM changes, and it remains constant when the BAM is in a stable state. Kosko also proved that unlike the Hopfield model, the BAM does not have oscillating states and always reaches a stable state for all $W$. 


4.3.2 BAM encoding

Kosko used the equation

\[ W_{ij} = \sum_{k=1}^{P} X_i^{(k)} Y_j^{(k)} \]  

(4.9)

to store a set of vector pairs \{((X^{(k)}, Y^{(k)}))_{k=1}^{P}\}. The correlation matrix \( W \) superimposes the information of several patterns on the same memory medium. However, unless the training vectors are orthogonal, the superimposition may introduce noise in the system and the recall of all training pairs is not guaranteed. Other learning schemes [106][112] have been proposed for improving the storage capacity of the BAM. One technique [106] is based on the orthogonal encoding of training vectors. Another technique [112] uses a multiple training concept to construct a generalization of the correlation matrix \( W \). While the multiple training technique can improve the storage capacity of the BAM, it does not guarantee the recall of all training pairs.

4.3.3 An iterative learning algorithm

The learning problem in the BAM is to find the weight and threshold values to store the training vectors in \( T \) as bidirectionally stable states. The state transition rule implies that vectors in \( T \) are stored as stable states if the following system of linear inequalities are satisfied for all \( k = 1, \ldots, P \).

\[ (\sum_{i=1}^{N} W_{ij} X_i^{(k)} - \theta Y_j^{(k)}) Y_j^{(k)} > 0 \quad \text{for } j = 1, \ldots, M \]  

(4.10)

\[ (\sum_{j=1}^{M} W_{ij} Y_j^{(k)} - \theta X_i^{(k)}) X_i^{(k)} > 0 \quad \text{for } i = 1, \ldots, N \]  

(4.11)
The inequalities determine a convex set in the weight space, and any point inside the convex set provides a solution to the learning problem, i.e., the coordinate values of such a point are the required weights.

In the system of inequalities (4.10) and (4.11), $W_{ij}$, $\theta X_i$ and $\theta Y_j$ are unknowns and the set of feasible solutions is a convex polyhedral cone with vertex at the origin, not including the boundary points. Thus, if (4.10) and (4.11) have a feasible solution, then for any positive $\xi$ the following system of linear inequalities has a solution:

\begin{align*}
&\left( \sum_{i=1}^{N} W_{ij}X_i^{(k)} - \theta Y_j \right)Y_j^{(k)} - \xi \geq 0 \text{ for } j = 1, \ldots, M \quad (4.12) \\
&\left( \sum_{j=1}^{M} W_{ij}Y_j^{(k)} - \theta X_i \right)X_i^{(k)} - \xi \geq 0 \text{ for } i = 1, \ldots, N \quad (4.13)
\end{align*}

The new learning algorithm for the BAM is described as a pseudo-relaxation procedure applied to the system of inequalities defined by (4.12) and (4.13). Starting with arbitrary initial values for weights and thresholds, the learning algorithm determines $w = (W_{ij}, \theta X_i, \theta Y_j)$, a $(NM + N + M)$-dimensional weight vector, which will satisfy the system of inequalities (4.10) and (4.11). The complete description of the learning algorithm is given below:

**Learning Algorithm LA2**

For each pair $(X^{(k)}, Y^{(k)})$, the vector $w$ is modified using the following adaptation rules:
For the nodes in the first layer,

\[
\Delta W_{ij} = -\frac{\lambda}{1 + M} [S_{X_i}^{(k)} - \xi X_i^{(k)}] Y_j^{(k)} \quad \text{if } S_{X_i}^{(k)} X_i^{(k)} \leq 0
\]

\[
\Delta \theta X_i = +\frac{\lambda}{1 + M} [S_{X_i}^{(k)} - \xi X_i^{(k)}] \quad \text{if } S_{X_i}^{(k)} X_i^{(k)} \leq 0
\]

(4.14)

and for the nodes in the second layer,

\[
\Delta W_{ij} = -\frac{\lambda}{1 + N} [S_{Y_j}^{(k)} - \xi Y_j^{(k)}] X_i^{(k)} \quad \text{if } S_{Y_j}^{(k)} Y_j^{(k)} \leq 0
\]

\[
\Delta \theta Y_j = +\frac{\lambda}{1 + N} [S_{Y_j}^{(k)} - \xi Y_j^{(k)}] \quad \text{if } S_{Y_j}^{(k)} Y_j^{(k)} \leq 0
\]

(4.15)

where

\[
S_{X_i}^{(k)} = \sum_{j=1}^{M} W_{ij} Y_j^{(k)} - \theta X_i
\]

\[
S_{Y_j}^{(k)} = \sum_{i=1}^{N} W_{ij} X_i^{(k)} - \theta Y_j
\]

The learning algorithm is easily derived from the relaxation procedure (3.7). Note that \( w \) corresponds to \( x \) and \( \xi \) corresponds to \( b_i \) when applying the relaxation procedure (3.7). The norm \( ||a|| \) in (3.7) becomes a constant as shown in the algorithm, since \( X \) and \( Y \) are bipolar vectors.

Consider a halfspace \( H_\xi \) such that

\[
H_\xi : S_{Y_j}^{(k)} Y_j^{(k)} - \xi \geq 0
\]

where

\[
S_{Y_j}^{(k)} = \sum_{i=1}^{N} W_{ij} X_i^{(k)} - \theta Y_j
\]
Let $w^q$ be the weight vector after $q$ iterations. Suppose $w^q \not\in H_\xi$ and $w^{q+1}$ is obtained by the learning rule. The algorithm performs relaxation procedure with respect to $H_\xi$ only if $S_{Y_j}^{(k)} Y_j^{(k)} \leq 0$. Note that

$$d(w^q, H_\xi) = \frac{|S_{Y_j}^{(k)} Y_j^{(k)} - \xi|}{\sqrt{N+1}}$$

$$\geq \frac{\xi}{\sqrt{N+1}} \quad \text{if } S_{Y_j}^{(k)} Y_j^{(k)} \leq 0$$

Thus, LA2 performs the relaxation procedure only if $d(w^q, H_\xi) \geq \frac{\xi}{\sqrt{N+1}}$. Similarly, for a halfspace $H_\xi$ defined by (4.13) the relaxation procedure is performed only if $d(w^q, H_\xi) \geq \frac{\xi}{\sqrt{M+1}}$. Thus, the proposed algorithm is a special case of the pseudo-relaxation method.

The relaxation factor $\lambda$, the initial vector $w^0$ and the constant $\xi$ are the parameters which need to be set for an application of LA2. We will describe empirical results which show that LA2 performs well irrespective of the choice of $w^0$. Lemma 4.3.1 is proven to show that our empirical results are also valid for random choices of $\xi$.

**Lemma 4.3.1** Suppose $\{w^q\}$ be a pseudo-relaxation sequence obtained by LA2 with $\xi = \xi_1$ to solve the system of linear inequalities (4.10) and (4.11). Then for any arbitrary choice of $\xi = \xi_2$, there exists a choice of the initial weight vector leading to a pseudo-relaxation sequence $\{\hat{w}^q\}$ such that $|\{w^q\}| = |\{\hat{w}^q\}|$.

**Proof** Let $\xi_2 = \alpha \xi_1$. Note that $\alpha > 0$. Let $w^0$ be the initial choice of weight vector which leads to the relaxation sequence $\{w^q\}$ for $\xi = \xi_1$. Consider a new relaxation sequence $\{\hat{w}^q\}$ obtained by choosing $\xi = \xi_2$ and $\hat{w}^0 = \alpha w^0$. We will show by
induction on $q$ that $\hat{w}^q = \alpha w^q$ for all $q$. Let

$$w^q = (W_{11}^q, \ldots, W_{NM}^q, \theta_{X_1}^q, \ldots, \theta_{X_N}^q, \theta_{Y_1}^q, \ldots, \theta_{Y_M}^q)$$

and

$$\hat{w}^q = (\hat{W}_{11}^q, \ldots, \hat{W}_{NM}^q, \hat{\theta}_{X_1}^q, \ldots, \hat{\theta}_{X_N}^q, \hat{\theta}_{Y_1}^q, \ldots, \hat{\theta}_{Y_M}^q)$$

**Basis** $\hat{w}^0 = \alpha w^0$ by choice.

**Induction** Assume that $\hat{w}^{q-1} = \alpha w^{q-1}$. Without loss of generality, we assume that the adaptation rule for the nodes in the first layer is applied. Define

$$S_{X_i}^q = \sum_{j=1}^{M} W_{ij}^q Y_j - \theta_{X_i}^q$$

$$\hat{S}_{X_i}^q = \sum_{j=1}^{M} \hat{W}_{ij}^q Y_j - \hat{\theta}_{X_i}^q$$

By the induction hypothesis, $S_{X_i}^{q-1} = \alpha S_{X_i}^{q-1}$. It follows that $S_{X_i}^{q-1} X_i \leq 0$ if and only if $S_{X_i}^{q-1} X_i \leq 0$. If $S_{X_i}^{q-1} X_i \leq 0$, then

$$\hat{W}_{ij}^q = \hat{W}_{ij}^{q-1} - \frac{\lambda}{1 + M} [S_{X_i}^{q-1} - \xi_1 X_i] Y_j$$

$$= \alpha W_{ij}^{q-1} - \frac{\lambda}{1 + M} [\alpha S_{X_i}^{q-1} - \alpha \xi_1 X_i] Y_j$$

$$= \alpha(W_{ij}^{q-1} - \frac{\lambda}{1 + M} [S_{X_i}^{q-1} - \xi_1 X_i] Y_j)$$

$$= \alpha W_{ij}^q$$
Similarly, $\dot{\theta}_{X_i}^q = \alpha \theta_{X_i}^q$. Since both $\theta_{Y_j}$ and $\dot{\theta}_{Y_j}$ remain unchanged, by the induction hypothesis,

$$\dot{\theta}_{Y_j}^q = \theta_{Y_j}^{q-1} = \alpha \theta_{Y_j}^{q-1} = \alpha \theta_{Y_j}^q,$$

Thus, $\dot{w}^q = \alpha w^q$. We have previously shown that if LA2 terminates then $w^q$ satisfies the inequalities (4.10) and (4.11). Since the system is homogeneous, if $w$ is a solution to the system, then so is $\alpha w$ for any $\alpha > 0$. Thus, $|\{w^q\}| = |\{\dot{w}^q\}|$. 

4.4 Dynamic Heteroassociative Neural Memory

Dynamic heteroassociative neural memory or, for short, dynamic associative memory (DAM) is a two-layer, feedback, heteroassociative pattern matcher. The DAM was proposed by Hassoun [54] and is a direct extension of the bidirectional dynamic correlation-type associative memories [62][76]. The DAM architecture, shown in Figure 4.3, is essentially same as the BAM architecture except that the connections are unidirectional.

The DAM uses an iterative learning algorithm based on the Ho-Kashyap algorithm [59]. The Ho-Kashyap algorithm is a technique to solve systems of inequalities. Unlike the relaxation method which directly solves each inequality one by one using orthogonal projection, the Ho-Kashyap method solves the system using an optimization technique.

4.4.1 The Ho-Kashyap algorithm

Consider an $N$-$M$ DAM with $N$ nodes in the first layer and $M$ nodes in the second layer. Let $W_{ij}^q$ be the connection weight from the $i$-th node in the first layer...
to the $j$-th node in the second layer, and $W_{ji}^*$ be the connection weight from the $j$-th node in the second layer to the $i$-th node in the second layer. Note that $W_{ij} 
eq W_{ji}^*$.

Let $\theta_{X_i}$ be the threshold (or bias) for the $i$-th node in the first layer and $\theta_{Y_j}$ be the threshold for the $j$-th node in the second layer. Let $T = \{(X^{(k)}, Y^{(k)})\}_{k=1}^P$ be a training set, where $X^{(k)} \in \{-1, +1\}^N$ and $Y^{(k)} \in \{-1, +1\}^M$.

The DAM stores vector pairs in $T$ if the following system of linear inequalities are satisfied for all $k = 1, \ldots, P$.

$$\left( \sum_{i=1}^N W_{ij} X_i^{(k)} - \theta_{Y_j} \right) Y_j^{(k)} > 0 \quad \text{for } j = 1, \ldots, M$$ (4.16)
If the system of inequalities (4.16) and (4.17) has a solution, then there exist $rX_i$ and $rY_j$ such that

$$
(\sum_{i=1}^{N} W_{ij} Y_j^{(k)} - \theta Y_j^{(k)}) = rY_j > 0 \text{ for } j = 1, \ldots, M
$$

(4.18)

$$
(\sum_{j=1}^{M} W_{ji} Y_j^{(k)} - \theta Y_j^{(k)}) = rX_i > 0 \text{ for } i = 1, \ldots, N
$$

(4.19)

In order to find a solution to the system, the Ho-Kashyap algorithm repeats the following steps:

- Assuming that $rY_j$ is fixed, $W_{ij}$ is determined such that it minimizes the sum of squares

$$
\sum_{k=1}^{P} \left[ (\sum_{i=1}^{N} W_{ij} Y_j^{(k)} - \theta Y_j^{(k)}) - rY_j \right]^2
$$

(4.20)

using the Moore-Penrose pseudo-inverse. Similarly, for a fixed $rX_i$, $W_{ji}^*$ is determined such that it minimizes the sum of squares

$$
\sum_{k=1}^{P} \left[ (\sum_{j=1}^{M} W_{ji}^* Y_j^{(k)} - \theta Y_j^{(k)}) - rX_i \right]^2
$$

(4.21)

- Assuming that $W_{ij}$ is fixed, $rY_j$ is changed in the direction of the gradient of Equation (4.20), subject to the condition $rY_j > 0$. $rX_i$ is similarly determined.

Details of the Ho-Kashyap algorithm is as follows:

1. Select $rY_j > 0$ and $rX_i > 0$
2. Assuming that \( rY_j \) and \( rX_i \) are fixed, minimize Equation (4.20) and Equation (4.21) using the Moore-Penrose pseudo-inverse technique.

3. Calculate the error \( eY_j \) and \( eX_i \):

\[
eY_j = \sum_{k=1}^{P} \left[ (\sum_{i=1}^{N} W_{ij} X_i^{(k)} - \theta Y_j^{(k)}) - rY_j \right] \tag{4.22}
\]

\[
eX_i = \sum_{k=1}^{P} \left[ (\sum_{j=1}^{M} W_{ji}^\pi Y_j^{(k)} - \theta X_i^{(k)}) - rX_i \right] \tag{4.23}
\]

4. If \( eY_j = 0 \) and \( eX_i = 0 \) then stop. Otherwise, go to the next step.

5. Change \( rY_j \) in the direction of the gradient of Equation (4.20), and change \( rX_i \) similarly:

\[
\Delta rY_j = \eta (eY_j + |eY_j|) \tag{4.24}
\]

\[
\Delta rX_i = \eta (eX_i + |eX_i|) \tag{4.25}
\]

where \( 0 < \eta < 1 \).

6. Go to Step 2.

The advantage of the Ho-Kashyap algorithm is that it always converges to a solution in a finite number of steps. The algorithm also helps to terminate early when there is no solution. If either \( eY_j \) or \( eX_i \) is negative, then one can conclude that the training samples are not linearly separable, and thus there is no solution. The disadvantage of this method is that it has some practical difficulties in its implementation. Since it involves the computation of the pseudo-inverse, its calculation would be very
expensive if the number of nodes and the number of training samples become large. This situation is very likely in practice. For example, the samples could be actual images of human faces, where one has to deal with huge matrices.

4.4.2 An iterative learning algorithm

An iterative algorithm described in this section is almost identical to the algorithm in Section 4.3. The only difference between BAM and DAM is that in the former network the connection weights are bidirectional \((W_{ij} = W_{ji})\), whereas in the latter case the connection weights are unidirectional \((W_{ij} \neq W_{ji}^*)\).

Learning Algorithm LA3

For each pair \((X^{(k)}, Y^{(k)})\), the vector \(w = (W_{ij}, \theta_{X_i}, W_{ji}^*, \theta_{Y_j})\) is modified using the following adaptation rules:

For the nodes in the first layer,

\[
\Delta W_{ji}^* = -\frac{\lambda}{1 + M} [S_{X_i}^{(k)} - \xi X_i^{(k)}] Y_j^{(k)} \quad \text{if } S_{X_i}^{(k)} X_i^{(k)} \leq 0
\]

\[
\Delta \theta_{X_i} = +\frac{\lambda}{1 + M} [S_{X_i}^{(k)} - \xi X_i^{(k)}] \quad \text{if } S_{X_i}^{(k)} X_i^{(k)} \leq 0
\]

and for the nodes in the second layer,

\[
\Delta W_{ij} = -\frac{\lambda}{1 + N} [S_{Y_j}^{(k)} - \xi Y_j^{(k)}] X_i^{(k)} \quad \text{if } S_{Y_j}^{(k)} Y_j^{(k)} \leq 0
\]

\[
\Delta \theta_{Y_j} = +\frac{\lambda}{1 + N} [S_{Y_j}^{(k)} - \xi Y_j^{(k)}] \quad \text{if } S_{Y_j}^{(k)} Y_j^{(k)} \leq 0
\]
where

\[ S_{X_i}^{(k)} = \sum_{j=1}^{M} W_{ji}^* Y_j^{(k)} - \theta X_i \]

\[ S_{Y_j}^{(k)} = \sum_{i=1}^{N} W_{ij} X_i^{(k)} - \theta Y_j \]

LA3 has several distinctive advantages over the Ho-Kashyap algorithm:

- LA3 is computationally feasible even for a larger problem, whereas the performance of the Ho-Kashyap algorithm may be limited by the problem size because each node may have to solve the pseudo-inverse of a large matrix.

- Using LA3, the DAM can be trained in an adaptive manner by examining each sample one by one. The Ho-Kashyap algorithm requires the simultaneous use of all training samples to compute the pseudo-inverse.

- LA3 is well suited for parallel and distributed processing, especially for neural-network implementation. The Ho-Kashyap algorithm involves the computation of the pseudo-inverse, whose parallel implementation is not immediate.

### 4.5 Radial Basis Function Networks

In the past few years, the Radial Basis Function (RBF) method has been investigated by Micchelli [83] and Powell [97] as a possible solution to the real multivariable interpolation problem. Recently, the RBF technique has been used in neural networks by many researchers [19][85][96] as an alternative tool for interpolating multivariate functions.
4.5.1 Multivariable interpolation using RBFs

Consider a set of $P$ distinct data points $T = \{(x^{(k)}, y^{(k)}) | k = 1, 2, \ldots, P\}$, where $x^{(k)} \in \mathbb{R}^N$ represents an input vector and $y^{(k)} \in \mathbb{R}$ is the corresponding output. The real multivariable interpolation problem can be stated as follows. Given $T$, find a function $F : \mathbb{R}^N \rightarrow \mathbb{R}$ which satisfies the interpolation conditions:

$$F(x^{(k)}) = y^{(k)} \quad \text{for } k = 1, \ldots, P \tag{4.28}$$

The RBF approach consists of constructing a linear function space which depends on the positions of the known data points. Consider a function $F$ of the following form:

$$F(x) = \sum_{i=1}^{M} w_i \phi(||x - c_i||) \tag{4.29}$$

where $\phi(||x - c_i||)$ is a radial basis function whose center is $c_i$, and $|| \cdot ||$ denotes a norm which is normally Euclidean. $w_i$ is a constant. The centers of the basis functions are normally taken to be sample data points, i.e., $c_i = x^{(i)}$.

Assume that the centers of the basis functions are known. The interpolation conditions (Equation (4.28)) gives the following system of linear equations for the coefficients $\{w_i\}$,

$$y = Aw \tag{4.30}$$

where $A$ is a $P \times M$ matrix such that

$$A_{ij} = \phi(||x^{(i)} - c_j||) \tag{4.31}$$

$y = (y^{(1)}, y^{(2)}, \ldots, y^{(P)})$ and $w = (w_1, w_2, \ldots, w_M)$. Thus, the interpolating problem is reduced to a problem solving a set of linear equations. There are two possible cases depending on $A$. 
• Case 1: If $P = M$, $A$ is a square matrix. Then, there always exists a solution if $A$ is non-singular. Micchelli [83] proved that if $c_i = X^{(i)}$ and the data points are different, then, for all positive integers $M$ and $N$ and for a large class of functions $\phi$, $A$ is non-singular.

• Case 2: If $P > M$, the system is overdetermined. In this case, the problem becomes one of linear optimization. The Moore-Penrose pseudo-inverse [86][95] and other methods [26][14][15] can be used to minimize the sum of squares,

$$\|Aw - y\|^2$$

(4.32)

4.5.2 RBF networks

A RBF network consists of a two-layer feedforward network with a single hidden layer. The RBF network architecture is shown in Figure 4.4. The input layer consists of $N$ nodes that accept $N$-dimensional vector $x$. The hidden layer is composed of a set of RBF nodes, each of which is associated with the center of the basis function. On receiving $x$ from the input nodes, each hidden node $i$ computes how close $x$ is to the center $c_i$ by calculating $\|x - c_i\|$, and produces $\phi(\|x - c_i\|)$ as an output. The hidden layer is fully connected to the output layer. The output layer consists of linear nodes so that the response of each output node is a linear function of its net input which may include a bias term. The output nodes could be non-linear, and as long as they have an invertible nonlinearity, the methods described in the previous section are applicable by a suitable modification of Equation (4.28).

The choice of a function that can be used as a RBF was studied by Micchelli [83]. Micchelli has proposed sufficient conditions for RBFs and any of the following functions can be used:
Figure 4.4: RBF network architecture
- Gaussians $\phi(r) = e^{-(r/c)^2}$
- Hardy's multiquadrics $\phi(r) = \sqrt{r^2 + c^2}$
- Inverse multiquadrics $\phi(r) = \frac{1}{\sqrt{r^2 + c^2}}$
- Thin plate splines $\phi(r) = r^2 \log r$
- Cubic $\phi(r) = r^3$
- Linear $\phi(r) = r$

The centers of radial basis functions need not be sample data points, and the number of RBF nodes can be smaller than the number of samples. Clustering algorithms [85][101][88] have been proposed to determine the number of RBF nodes and their centers. The RBF approach has been further generalized to have adjustable centers and to use a different distance metric [96].

4.5.3 Non-exact interpolation using RBFs

If the number of sample data points becomes larger, one needs a large number of RBF nodes in the hidden layer to satisfy the interpolating conditions given by Equation (4.28). In practice, increasing the RBF nodes may not be feasible due to hardware restrictions. Moreover, the sample data points could be noisy. Under these circumstances, which may often occur in real-time applications, it would be more reasonable to try to find a solution within a certain error tolerance range rather than to satisfy the exact conditions. Let $e^{(k)}$ be the error tolerance for $(X^{(k)}, Y^{(k)})$. Then,
Equation (4.28) can be restated as follows:

$$|F(X^{(k)}) - Y^{(k)}| \leq \epsilon^k \text{ for } k = 1, \ldots, P$$  \hspace{1cm} (4.33)

Equation (4.33) gives rise to $2P$ inequalities. The system of inequalities can be solved using the relaxation method.

Let $T = \{(X^{(k)}, Y^{(k)}) | k = 1, 2, \ldots, P\}$ be a set of $P$ sample data points, where $X^{(k)} = (X_{1}^{(k)}, X_{2}^{(k)}, \ldots, X_{N}^{(k)})$ in $\mathbb{R}^N$ represents an input vector and $Y^{(k)}$ is the corresponding output. Let $F$ be a function of the form in Equation (4.29).

Introducing the bias $w_0$ to the output node, Equation (4.33) can be restated:

$$|\sum_{i=0}^{M} w_i \phi(||X^{(k)} - c_i||) - Y^{(k)}| \leq \epsilon^k \text{ for } k = 1, \ldots, P$$ \hspace{1cm} (4.34)

where $w_0$ is the bias with $\phi(||x - c_0||) = -1$. Let

$$I_i^k = \phi(||X^{(k)} - c_i||)$$

and

$$O^k = \sum_{i=0}^{M} w_i \phi(||X^{(k)} - c_i||) = \sum_{i=0}^{M} w_i I_i^k$$ \hspace{1cm} (4.36)

The relaxation algorithm is as follows:

**Learning Algorithm LA4**

Given a training set $T = \{(X^{k}, Y^{k})\}$, where $X^{k} \in \mathbb{R}^N$ and $Y^{k} \in \mathbb{R}$, the following adaptation rule is applied for each $(X^{k}, Y^{k})$:

$$\Delta w_i = \begin{cases} -\lambda \frac{O^k - Y^k - \epsilon^k}{\sum_{i=0}^{M} (I_i^k)^2} I_i^k & \text{if } O^k - Y^k > \epsilon^k \\ -\lambda \frac{O^k - Y^k + \epsilon^k}{\sum_{i=0}^{M} (I_i^k)^2} I_i^k & \text{if } O^k - Y^k < -\epsilon^k \end{cases}$$ \hspace{1cm} (4.37)
Note that $\lambda$ is the relaxation factor that must be selected between 0 and 2 for guaranteed convergence. The choice of $\epsilon^k$ is critical in real applications. If $\epsilon^k$ is too small, the system of inequalities (4.34) may be inconsistent, resulting in no solution. On the other hand, if it is too large, the network might poorly interpolate the sample data points, resulting in poor generalization. Although in practice the estimation of $\epsilon^k$ is somewhat *ad hoc*, it is desirable to reflect our past experience along with knowledge of the method of data collection in its choice as much as possible.

After a feasible solution to the system of inequalities (4.34) is found, the solution may be further optimized using the equation:

$$\Delta w_i = -\eta(O^k - Y^k)I_i^k$$

(4.38)

where $\eta$ is a positive constant and usually a small $\eta$ is preferable. Equation (4.38) is often called the delta rule and it minimizes the sum of squared errors $\sum_{k=1}^{P} (O^k - Y^k)^2$ using the steepest descent technique. Since Equation (4.38) also solves the system of inequalities (4.33), it could be used in place of Equation (4.37). However, Equation (4.38) is much slower than Equation (4.37), even if both equations have a common geometric interpretation. Both the methods utilize orthogonal projections, i.e., $w$ is projected to a new point along the direction of the normal to the hyperplane defined by a sample point. The critical difference lies in the amount of distance by which each method moves. Equation (4.37) moves the point by the amount proportional to the distance to the hyperplane, whereas Equation (4.38) moves proportional to the residual. The residual could be large depending on the direction of the hyperplane even though the point is near the hyperplane, or possibly near the solution set. A large residual can cause a serious "over-shooting" towards a solution and have
a tendency to zig-zag about the true direction to a minimum. On the other hand, Equation (4.37) guarantees that the new point is always closer to a solution than the previous one.

4.6 Concluding Remarks

New iterative learning algorithms based on the relaxation method have been proposed for various neural network models. The learning problem in the feedback networks including the Hopfield model, Bidirectional Associative Memory and Dynamic Heteroassociative Memory involves finding a set of weight values to store a set of training vectors as stable states in the time-varying system. The state transition rule of the network leads to a system of linear inequalities. The inequalities determine a convex set in the weight space, and any point inside the convex set provides a solution to the learning problem. The relaxation technique provides several benefits which are not found in correlation-type algorithms. Since the technique always finds a solution, if one exists, it exploits the maximum capacity of a network and guarantees recalls of all the training vectors. The technique is also independent of the network topology, which helps a graceful recovery from a few connection failures by finding an alternative representation without those connections. For the failed connection, its weight can be regarded as a constant, rather than a variable to be determined. Mathematically, the system will have a lower degree of freedom with connection failures.

The relaxation technique is also successfully used for deriving an iterative learning algorithm for RBF networks in the situation where the set of sample points is large and possibly noisy. The same technique is easily applicable to several other models.
including *Temporal Associative Memory* [6], *Brain-State-in-a-Box* [10], *Cascade Correlation Architecture* [35], and *Functional Link Net* [90]. One important property of the relaxation technique is that it utilizes a dynamic learning rate. The learning rate is a varying value determining the degree of weight adjustment. Each of the relaxation algorithms presented in this chapter determines its learning rate proportional to the distance to the decision hyperplane rather than simply employing a fixed learning rate. There are some algorithms where the fixed learning rate is crucial for guaranteed convergence [99][20][111][36][40]. The other advantage of the relaxation technique is its suitability for a neural network implementation. Each algorithm changes weights in an adaptive manner by examining one sample after another until the desired performance is achieved.
5. SIMULATION STUDY

5.1 Introduction

This chapter presents the results of several simulation experiments done using relaxation algorithms. In this simulation study, we address the following performance considerations: learning speed, parameter sensitivity and scalability. The learning speed is a metric to measure the time that a learning algorithm takes to achieve the desired performance of a network. We use the term epoch to denote a learning iteration. Each training sample is presented once during an epoch, and the learning speed is measured in number of epochs. Parameter sensitivity is an important issue in learning algorithms. The learning speed not only depends on the nature of training data, but also depends on appropriate choice of learning parameters. Depending on learning parameters, significant variations in performance of the same learning algorithm are possible. The purpose of the study of parameter sensitivity is to provide an in-depth analysis of a learning algorithm and to suggest a guide line for subsequent users of the algorithm. Finally, an algorithm is said to be scalable if its learning speed remains fairly constant when the number of training samples and the number of processing nodes are both increased in the same proportion. The scalability of an algorithm is an important advantage for large applications. Unfortunately, the number of training samples is not the only factor that affects the learning speed of an
algorithm. Since the learning speed also depends on how difficult a given problem is and various other factors, any discussion of the scalability would be premature unless there are common characteristics among the sample data sets. For this reason, the scalability test was conducted only on randomly generated data.

5.2 Performance of Associative Memories

5.2.1 Random patterns

The information capacity of Hopfield-style networks has been extensively studied using random patterns. Analytic techniques for estimating the capacity relying on a probabilistic notion [8][1][9][114][110][81][78] show that the capacity using the first-order correlation encoding is very limited. The correlation encoding scheme does not guarantee storage of training patterns unless they are represented by orthogonal vectors. For instance, Hopfield [62], based on his simulation study, observed that for randomly generated patterns the capacity is well below $0.15N$, where $N$ is the number of nodes. Moreover, the capacity deteriorates significantly if one tries to store more patterns or if the patterns are correlated, sharing many portions in common. This deterioration occurs because the noise term in correlation encodings becomes increasingly dominant and causes more failures in storing of patterns. The memory capacity of other models shows similar behavior when the first-order correlation encoding is used. The true maximum capacity of the Hopfield model for storing random patterns is $2N$ [28][110] and it can be much larger for correlated patterns [124][39].

Correlation encodings are not powerful enough to fully exploit the maximum capacity. Relaxation algorithms store as many patterns as they can up to the maximum capacity and fail only if the network is not capable of storing the given patterns. One
experiment was performed to test the capacity of the Hopfield model using different random patterns. The results of this experiment are summarized in Table 5.1. The statistics was computed over 100 trials of the experiment using the relaxation algorithm LA1. The parameters were kept fixed at $\lambda = 1.9$ and $\xi = 0.1$, and the initial weights were chosen randomly between $-1$ and $1$. As shown in Table 5.1, storing up to $N$ random patterns is achieved fairly quickly, requiring an average of 7 epochs, where $N$ is the number of nodes.

Table 5.1: Storing random patterns in Hopfield networks

<table>
<thead>
<tr>
<th>type</th>
<th>number of patterns</th>
<th>number of nodes</th>
<th>learning epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>min</td>
</tr>
<tr>
<td>random</td>
<td>50</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td>random</td>
<td>100</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>random</td>
<td>150</td>
<td>150</td>
<td>6</td>
</tr>
<tr>
<td>random</td>
<td>200</td>
<td>200</td>
<td>6</td>
</tr>
<tr>
<td>random</td>
<td>250</td>
<td>250</td>
<td>6</td>
</tr>
<tr>
<td>random</td>
<td>300</td>
<td>300</td>
<td>7</td>
</tr>
</tbody>
</table>

In another experiment, the capacity of the BAM was tested. The result of the experiment is compared with the results of other coding schemes, as reported in [112], including a multiple training strategy [112]. As shown in Table 5.2, in all cases, the other schemes could store only a fraction of the total number of training patterns, whereas the relaxation algorithm LA2 successfully stored all the patterns. Moreover, for the relaxation algorithm, the average number of learning epochs was small. The last two columns of Table 5.2 are based on 200 trials of the experiment using the relaxation algorithm. Parameters were $\lambda = 1.9$ and $\xi = 0.1$. The initial weights were chosen randomly between $-1$ and $1$. 


Table 5.2: Storing random patterns in BAMs

<table>
<thead>
<tr>
<th>number of nodes</th>
<th>number of training patterns</th>
<th>number of stored patterns</th>
<th>learning epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>correlation encoding</td>
<td>multiple training</td>
</tr>
<tr>
<td>100-100</td>
<td>50</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>145-145</td>
<td>50</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>200-200</td>
<td>100</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>225-225</td>
<td>100</td>
<td>14</td>
<td>20</td>
</tr>
</tbody>
</table>

5.2.2 Correlated patterns: CGA character font

We also studied a practical example using correlated patterns drawn from the IBM PC CGA (Color Graphics Adapter) character font in Figure 5.1. Each character is defined within a $7 \times 7$ array of pixels. Character representation in the network is straightforward, i.e., nodes have state 1 for black pixels and −1 for white pixels.

This experiment illustrates the significant increase in information capacity for correlated patterns using the relaxation technique in place of correlation encoding. Figure 5.2 shows the results of a comparison between the two methods for a 49-node Hopfield network using 10 digits as training patterns. The network trained by the relaxation algorithm LAI not only recalls all 10 digits but also performs well in the presence of noise. On the other hand, the first-order correlation encoding scheme is capable of recalling only 20% of the input patterns, even when they are noise-free. In fact, out of 10 digits only the digits ‘1’ and ‘4’ are successfully stored by the correlation encoding scheme.

Correlation encoding provides limited storage capacity. Moreover, the storage capacity drops sharply as the number of patterns is increased. If training patterns
Figure 5.1: The sample training set of 93 patterns
Figure 5.2: Recognition capability in the presence of noise when 10 digits are stored in a 49-node Hopfield network are correlated, correlation encoding deteriorates even further. In the case of capital letters, the patterns share many pixels in common showing correlations. The correlation encoding scheme performs poorly with the set of 26 capital letters as shown in Figure 5.3.

In another experiment, all pairs of capital and corresponding small letters were presented to a BAM with 49 nodes per layer. The relaxation algorithm LA2 stored all 26 associations in the BAM. The first-order correlation encoding scheme could not store any of the 26 pattern pairs when all of them were presented to the BAM. Since correlation encoding completely failed with 26 pattern pairs, we did another experiment using only the 5 vowel pairs. In this case, correlation encoding stored only the (E,e) pair. The quality of recall from noisy patterns was checked in the presence
Figure 5.3: Recognition capability in the presence of noise when 26 capital letters are stored in a 49-node Hopfield network of random noise. As shown in Figure 5.4, the network trained using the relaxation algorithm can not only store more patterns, but also performs well in recalling noisy patterns. An example of a recall from a noisy pattern is illustrated in Figure 5.5. As indicated by our experiments, the inability of the correlation encoding scheme to store the complete set of training patterns is a major drawback in real applications.

5.3 Sensitivity to Learning Parameters

5.3.1 Initial weights

The objective of the experiments was to study the sensitivity of relaxation algorithms to the initial values of weights. Several different sets of patterns, ranging
Figure 5.4: Recognition capability in the presence of noise when 5 vowel pairs are stored in a 49-49 BAM from the set of 10 digits to the set of 93 character font consisting of digits, letters and special symbols, were used to train a 49-node Hopfield network. For each set of patterns, 500 trials were run with randomly generated initial weights ranging from −1 to 1. For each trial, we recorded the number of epochs necessary to store all the patterns in the given set. The average number of learning epochs was computed over 500 trials. The results of this experiment are shown in Table 5.3.

During this experiment the parameters $\lambda$ and $\xi$ were kept fixed at $\lambda = 1.8$ and $\xi = 0.1$. As shown in Table 5.3, the number of learning epochs remained low for all trials for each set of patterns. The same experiment was performed by selecting each weight between −500 and 500, and almost identical results were obtained (see Table 5.4). For example, in both experiments, storing 93 character sets required an
average of 11 epochs. Our simulation results confirm a mathematical property of the relaxation method: since the method makes a dynamically varying adjustment proportional to the distance from the current vector to the halfspace which contains the solution set, no matter how far the initial weight vector is from the solution set, the method quickly moves it near the solution set after a few iterations. This is quite different from other iterative learning algorithms [99][20][111][36][40], where the weight adjustments are static. In these algorithms, if the initial weight vector is far away from the solution set, it certainly take more steps to move the weight vector near the solution set because the fixed adjustment is too small.

5.3.2 Relaxation factor $\lambda$

To measure the effects of $\lambda$, a 49-node Hopfield network was trained for the same character sets used in Section 5.3.1 by selecting $\lambda$ from 0 to 2 with a step size of 0.01. In order to carry out a fair comparison, each trial started with the same initial
Table 5.3: Effect of the initial choice of weights on learning: initial values are randomly chosen between $-1$ and $1$

<table>
<thead>
<tr>
<th>type</th>
<th>number of patterns</th>
<th>number of nodes</th>
<th>learning epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>min</td>
</tr>
<tr>
<td>digit</td>
<td>10</td>
<td>49</td>
<td>3</td>
</tr>
<tr>
<td>upper case</td>
<td>26</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>lower case</td>
<td>26</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>special</td>
<td>31</td>
<td>49</td>
<td>5</td>
</tr>
<tr>
<td>all of them</td>
<td>93</td>
<td>49</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 5.4: Effect of the initial choice of weights on learning: initial values are randomly chosen between $-500$ and $500$

<table>
<thead>
<tr>
<th>type</th>
<th>number of patterns</th>
<th>number of nodes</th>
<th>learning epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>min</td>
</tr>
<tr>
<td>digit</td>
<td>10</td>
<td>49</td>
<td>3</td>
</tr>
<tr>
<td>upper case</td>
<td>26</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>lower case</td>
<td>26</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>special</td>
<td>31</td>
<td>49</td>
<td>5</td>
</tr>
<tr>
<td>all of them</td>
<td>93</td>
<td>49</td>
<td>9</td>
</tr>
</tbody>
</table>

configuration (all weights were initialized to zero). Table 5.5 summarizes the effect of varying $\lambda$, where $\lambda \in (1, 2)$. As indicated by the results in Table 5.5, the effect of $\lambda$ is small. For the case of 93 patterns, the detailed results are plotted in Figure 5.6.

In another experiment, $\lambda$ was also varied in the interval $(0, 2)$ in steps of 0.01 with several experiments using different values for the size of the BAM and the number of pairs. The results for a 100-100 BAM with 100 randomly chosen training pairs and for 26 letter pairs are reported in Figure 5.7. As shown in Figure 5.7, for the relaxation
factor $\lambda$ in the range 1.4 to 2.0, the algorithm LA2 uses the least number of learning epochs and is almost insensitive to the choice of $\lambda$. Our experiments consistently indicated that when inputs are binary, the performance of relaxation algorithms was almost insensitive to the choice of $\lambda$ within a certain range, normally between 1 and 2, and also provided the best performance in that range.

### 5.4 Delta Rule vs Relaxation Method

Delta rule is an iterative learning algorithm for a single-layer network using a linear activation function at each nodes. The delta rule minimizes the sum of squares of the error over all the training samples using the steepest descent technique. An iterative learning algorithm for the Hopfield networks described in [20] can be considered as the delta rule adapted to the feedback networks, though the algorithm is essentially an extension of the fixed-increment perceptron learning algorithm [99][84].
In general, the algorithm can be written as:

\[
\begin{align*}
\Delta W_{ij} &= -\eta[A_i^{(k)} - X_i^{(k)}]X_j^{(k)} \\
\Delta \theta_i &= +\eta[A_i^{(k)} - X_i^{(k)}]
\end{align*}
\] (5.1)

where \( \eta \) is the gain factor, and \( A_i^{(k)} \) is the actual output of the \( i \)-th node after the presentation of the \( k \)-th training pattern. It is easily seen that the algorithm described in [20] is a special case of (5.1) where \( \eta = 0.5 \).

The delta rule can also be applied to two-layer feedback networks such as BAM and DAM. In our empirical study, the following delta rule is considered for training...
Figure 5.7: Effect of varying $\lambda$ on the learning speed in BAMs

the BAM:

For the nodes in the first layer,

$$\Delta W_{ij} = -\eta[A^{(k)}_{X^i} - X^{(k)}_i]Y^{(k)}_j$$

$$\Delta \theta X^i = +\eta[A^{(k)}_{X^i} - X^{(k)}_i]$$

and for the nodes in the second layer,

$$\Delta W_{ij} = -\eta[A^{(k)}_{Y^j} - Y^{(k)}_j]X^{(k)}_i$$

$$\Delta \theta Y^j = +\eta[A^{(k)}_{Y^j} - Y^{(k)}_j]$$
where $A_{X_i}^{(k)}$ is the actual output of the $i$-th node in the first layer and $A_{Y_j}^{(k)}$ is the actual output of the $j$-th node in the second layer after the presentation of the $k$-th training pair.

First, we carried out an experiment on the learning speed by storing 150 random patterns in a 100-node Hopfield network. The relaxation algorithm LAI did significantly better than the delta rule. LAI required 25 epochs whereas the delta rule terminated after 1577 epochs for the same training set. Each training pattern is presented once in a learning epoch. The learning curves for the relaxation algorithm and the delta rule are shown in Figure 5.8 and Figure 5.9, respectively.

The learning curve plots the number of epochs and the number of parallel weight
Figure 5.9: Learning curve for the delta rule when a 100-node network is trained on 150 random patterns.

updates per epoch. Note that the maximum number of parallel weight updates per epoch is at most equal to the total number of patterns. The number of parallel weight updates drops sharply after some iterations for the relaxation algorithm. The delta rule shows an erratic behavior where the number of weight updates keeps oscillating. Our experiments show that the relaxation algorithm requires not only a very small number of epochs but also significantly fewer weight updates per epoch compared to the delta rule.

In the next two experiments, the two algorithms were compared in terms of their sensitivity to initial weights and learning parameters. The objective of the first set of experiments was to study the sensitivity to the initial values of weights while keeping the number of training pairs small. A 49-49 BAM was chosen to store 5 vowel pairs.
Table 5.6: Sensitivity of learning to initial choice of weights: delta rule vs relaxation algorithm

<table>
<thead>
<tr>
<th>method</th>
<th>learning epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg.</td>
</tr>
<tr>
<td>relaxation algorithm</td>
<td>4.70</td>
</tr>
<tr>
<td>delta rule</td>
<td>174.83</td>
</tr>
</tbody>
</table>

The results are shown in Table 5.6. The relaxation algorithm LA2 required 4.7 epochs on the average with a standard deviation of 0.53. The delta rule required an average of 174.8 epochs with a standard deviation of 28.4. The statistics are based on 200 trials of the experiment with randomly chosen weights between —500 and 500. Parameters were $\lambda = 1.9$, $\xi = 0.1$ and $\eta = 0.5$.

The second set of experiments was designed to study the sensitivity to learning parameters: $\lambda$ for the relaxation algorithm, and $\eta$ for the delta rule. The relaxation factor $\lambda$ was varied in the interval $(0,2)$ in steps of 0.01. For the delta rule, the gain factor $\eta$ was varied in the interval $(0,1)$ with the same step size. The initial weights were chosen randomly between $-1$ and $1$ and the same initial weights were used in each trial for both methods. The results for a 100-100 BAM with 100 randomly chosen training pairs are reported in Figure 5.10 and Figure 5.11. For the relaxation factor $\lambda$ in the range 1.7 to 1.9, the relaxation algorithm uses the least number of learning epochs and is almost insensitive to the choice of $\lambda$. The performance curve for the delta rule displays wide swings depending on the choice of the gain factor $\eta$, as shown in Figure 5.11.

The discrepancy in the learning speed is mainly because of the different learning rate that they employ. Let $\mathbf{W} \in \mathbb{R}^N$ be a weight vector. We can treat $\mathbf{W}$ as a point in
Figure 5.10: Effect of varying relaxation factor $\lambda$ on the learning speed of the relaxation algorithm

an $N$-dimensional space. Starting from an arbitrary point $W^0$, both algorithms use the orthogonal projection method to find a point $W$ on the right side of all decision hyperplanes. The delta rule moves the point in the direction of the normal to a hyperplane by the amount proportional to the residual. Note that the residual is a fixed constant in the delta rules described in this section. It implies that the distance by which the point moves is fixed. At the early learning stage, the fixed amount could be too small, resulting in a slow convergence to a solution. However, when the point is near the solution, the fixed amount could become too large, resulting in overshooting the solution. The updates made by relaxation algorithms are not static. The distance by which a point moves varies dynamically by the amount proportional
Figure 5.11: Effect of varying gain factor $\eta$ on learning speed of the delta rule

to the distance between the point and the hyperplane. Thus, relaxation algorithms
move the point near the solution set not only faster but also more accurately. The
difference between the two updating rules is illustrated in Figure 5.12 for a two-
dimensional case.

5.5 Scalability

Relaxation algorithms exhibit high scalability on binary random patterns. As
shown in Table 5.1, the number of learning epochs remains fairly constant as long as
the number of patterns and the number of nodes are increased in the same proportion.
The scalability of relaxation algorithms become even clearer when it is compared with
the performance of the delta rule. To test the scalability of each algorithm, $N$ random
The update $\mathbf{W}^D$ using the delta rule and the update $\mathbf{W}^R$ using the relaxation method are shown for two initial positions of the point $\mathbf{W}$. The distance between $\mathbf{W}^D$ and $\mathbf{W}$ is fixed, whereas the distance between $\mathbf{W}^R$ and $\mathbf{W}$ varies depending on the position of $\mathbf{W}$.

Patterns were tried for each $N$-$N$ BAM as $N$ increased. Initial weights were randomly chosen between $-1$ and $1$ for both methods. Parameters were $\lambda = 1.9$, $\xi = 0.1$ and $\eta = 0.5$. The results of this experiment are presented in Table 5.7 and Figure 5.13. The statistics in Table 5.7 are based on 100 trials of the experiment. As shown in Figure 5.13, The relaxation algorithm significantly outperforms the delta rule in speed and also shows higher scalability.

5.6 Concluding Remarks

Relaxation algorithms overcome an important practical difficulty commonly faced by iterative learning algorithms. Iterative algorithms depend on parameters, and often the difficulty lies in the selection of "good" values for fast learning. Relaxation
Table 5.7: Scalability of learning speed, measured in number of epochs

<table>
<thead>
<tr>
<th>nodes</th>
<th>training patterns</th>
<th>relaxation</th>
<th>delta rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>avg.</td>
<td>std. dev.</td>
</tr>
<tr>
<td>50-50</td>
<td>50</td>
<td>18.05</td>
<td>2.35</td>
</tr>
<tr>
<td>100-100</td>
<td>100</td>
<td>20.51</td>
<td>1.59</td>
</tr>
<tr>
<td>150-150</td>
<td>150</td>
<td>20.54</td>
<td>1.42</td>
</tr>
<tr>
<td>200-200</td>
<td>200</td>
<td>21.54</td>
<td>1.27</td>
</tr>
</tbody>
</table>

algorithms are highly insensitive to the initial choice of weights and to the choice of parameters within a certain range. They also exhibit high scalability on binary random patterns. The number of learning epochs remains fairly constant when the number of pairs and the number of nodes are both increased in the same proportion. The scalability of an algorithm is clearly an important advantage for large applications. Finally, the relaxation method is significantly faster than the delta rule, especially for a large number of training pairs of patterns. However, since the relaxation method does not terminate when there is no feasible solution, establishing a termination criterion is an interesting problem. Our experiments indicate that the number of weight updates drops sharply after a few iterations and relaxation algorithms quickly converge to a solution. When a solution does not exist, the number of weight updates does not drop sharply, but starts oscillating. This information can hopefully be used to establish a general termination criterion when no solution exists.
Figure 5.13: Scalability of the learning speed
6. CONCLUSION

In this thesis, a new mathematical approach for deriving learning algorithms for various neural network models is presented. The mathematical approach is based on the relaxation method for solving systems of linear inequalities. The newly developed learning algorithms are fast and guaranteed to converge to a solution in a finite number of steps. The convergence rate follows a geometric progression. The new algorithms are highly insensitive to choice of parameters and the initial set of weights. Rigorous mathematical foundations for the new algorithms and their simulation studies are provided.

Artificial neural networks provide some major benefits, such as the ability to take incomplete or noisy data and produce approximate results. Massive parallelism and highly distributed representations give neural networks a high degree of fault tolerance and graceful degradation from processor failures. Since information stored in the networks is distributed over a large number of interconnections, there exists a built-in redundancy to withstand component failures without crashing. Another important benefit is the ability to retrieve information instantaneously based on content and to make a reasonable guess if there is no exact match for the requested information.

How closely must the internal representations and processes of a man-made
machine resemble those of humans for it to learn and recall like the human brain does? Unfortunately, we know little about how exactly our own biological memories learn and recall, and what distinguishes learning from recall. Moreover, the current neural-network learning algorithms are neither very novel nor powerful. This research work has been done to develop a better learning technique for the existing neural network models and to establish a new and effective learning paradigm.
BIBLIOGRAPHY


