Associative Memory as a Bayesian Building Block

Shaojuan Zhu
B.S., Electrical Engineering, Beijing Institute of Technology, 1996
M.S., Electrical Engineering, Beijing Institute of Technology, 1999

Presented to the Division of Biomedical Computer Science within
The Department of Science & Engineering
and the Oregon Health & Science University
School of Medicine
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
in
Electrical Engineering

August 2008

The dissertation “Associative Memory as a Bayesian Building Block” by Shaojuan Zhu has been examined and approved by the following Examination Committee:

_____

Dr. Dan Hammerstrom
Professor
Thesis Research Advisor

_____

Dr. Misha Pavel
Professor

_____

Dr. Todd K. Leen
Professor

_____

Dr. Xubo Song
Assistant Professor

Dedication

To my parents, Ziru Zhu and Guiying Zhao

Acknowledgment

I would like to thank my advisor, Dr. Dan Hammerstrom, for giving me the opportunity to work on my doctorate under his guidance. This thesis would not have been completed without his support, advice and encouragement. I benefit a lot from his rigorous scholarship, and my writing has improved greatly due to his detailed editing.

My sincere gratitude also goes to my committee members Dr. Misha Pavel, Dr. Xubo Song and Dr. Todd K. Leen for their time and effort in reviewing this work. Their valuable suggestions and comments are appreciated.

I am deeply and forever indebted to my parents for their unconditional love, support and encouragement throughout my entire life. I am also very grateful to my sisters Shaoqin and Shaoming.

Last but not least, I would like to express my gratitude to my husband Li Fang for his love and support.

Contents

Dedication iii
Acknowledgment iv
Abstract ix
Chapter 1 Introduction 1
1.1 Biological Implication 1
1.2 Associative Memory 2
1.3 Brief Review of Associative Memory 3
1.4 Thesis Organization 6
Chapter 2 Large Network Simulations 9
2.1 Aims 9
2.2 Palm Network 9
2.3 Simulation Tool - Csim 12
2.4 Simulation Results 14
2.4.1 On PC and Beowulf Cluster 14
2.4.2 On NASA Supercomputers 17
2.4.3 Discussion 18
2.5 Computational Complexity 19
Chapter 3 Voronoi Classification in Associative Memory 21
3.1 Application Issues 21
3.2 Communication Channel Analogy 23
3.3 Dendritic Sum Distribution in Palm Network 25
3.4 Bayesian Inference from a Voronoi Tessellation 27
3.5 Conditioned Bayesian Classifier in Binary Palm Network 28
3.5.1 Output Error Characteristics 28
3.5.2 The Attractor Basin 30

3.5.3 Bayesian Interpretation 33
3.5.4 Simulation Results 34
3.5.5 Fault Tolerance Experiments 38
3.6 Dynamic Learning in Palm Network 43
3.6.1 MAP Retrieval in Dynamic Learning 43
3.6.2 Weighted Voronoi Regions in Dynamic Learning 46
3.6.3 The Retrieval Performance in Dynamic Network 48
Chapter 4 Reinforcement Learning in Associative Memory 51
4.1 Introduction 51
4.2 Mappings in Brain 52
4.3 Reinforcement Learning (RL) 54
4.4 The RLAM System 56
4.4.1 The Model 56
4.4.2 Implementation – Grid-world Mapping 58
4.5 Complex Mappings 61
Chapter 5 Spiking Palm Model 65
5.1 Background 65
5.2 Description of Spiking Neuron Model 67
5.3 Implementation 68
5.3.1 Hebbian Learning 68
5.3.2 Network Retrieval Rrobustness 70
Chapter 6 Hierarchical Associative Networks 74
6.1 Scaling Problem 74
6.2 Implementation Cost 75
6.3 Brief Review of Some Hierarchical Memory Models 77
6.4 Bayesian Memory Based Hierarchical Network 81
6.4.1 Bayesian Memory 81
6.4.2 Bayesian Memory Implementation 87
6.4.3 Simulations and Results 89
Chapter 7 Summary and Conclusions 94
Bibliography 98

List of Figures

Figure 2.1 Retrieval performance versus the number of training vectors being stored 15
Figure 2.2 Retrieval performance versus the number of active nodes in the training vectors 16
Figure 2.3 The relationship between the MPI percentage and the vector size 17
Figure 3.1 The Decoder Model of Pattern Classification 23
Figure 3.2 The Association Memory as a Channel Decoder 23
Figure 3.3 The relationship between the output error rate $e_o/(2\alpha)$ and input bit-flip rate e_i 33
Figure 3.4 Attractor basin size e_{max} vs. number of stored training vectors M. 35
Figure 3.5 The attractor basin size e_{max} varies with the number of vectors stored *M* and the number of active nodes k, with network size *n*=1000..... 36
Figure 3.6 The attractor basin e_{max} varies with the network density ρ_l for different network size *n*. 37
Figure 3.7 Illustration of convergence between two vectors 38
Figure 3.8 Network performance for random addition and deletion of connections in weight matrix 40
Figure 3.9 Network performance for random fractional values in weight matrix 40
Figure 3.10 Network performance for random addition and deletion of connections, and each connection in the weight matrix is multiplied by a fractional value. 41
Figure 3.11 Performance comparison of a fault free Palm and the same network with Static faults. Three different kinds of Static faults are generated: random addition and deletion of connections, connection multiplication by random fractional values. 42
Figure 3.12 Performance comparison of a fault free Palm and the same network with Dynamic faults. Three different kinds of Dynamic faults are generated: random

addition and deletion of connections, connection multiplication by random fractional values.. 42
Figure 3.13 The relationship between the retrieval performance and the influence coefficient of each training vector. In this network, n=1000, k=10, M=700, and e_i=0.3. For each training vector, 300 testing vectors are used, and all the 300 test vectors have the same c (c=i/1000, i is the index of the training vector). 48
Figure 4.1 Retrieval performance versus total number of training steps 59
Figure 4.2 Different road maps in grid-world associate with different actions. A(1) is the simplest grid-world, and the goal (the red point) is at the center. A(2)-A(5) are pseudo gray-level images of weights corresponding to four output neurons, with 1 representing white and 0 representing black. A(2)-(5) illustrate the weight values that associate the states with the corresponding actions of going north, south, east and west respectively. B(1) is a grid-world that has barriers colored in green. B(2)-(5) correspond to the weights that associate the states with the actions of going north, south, east and west respectively. 60
Figure 5.1 The Hebbian learning between pre- and post-synapse. 68
Figure 5.2 Hebbian learning between input and output strings of spikes. 69
Figure 5.3 Piece-wise linear post-synaptic potential 70
Figure 5.4 Pulses representation of training, testing and output patterns. 72
Figure 6.1 (A) HTM structure, (B) Belief propagation computations within a node. 82
Figure 6.2 Bayesian Memory External Connections. V_{Lin} is the input vector from the lower level network, V_{Hout} is the output vector feeding to the higher level network, V_{Hin} is the feedback vector coming from the higher network, and V_{Lout} is the output vector feeding down to the lower level network. 83
Figure 6.3 A BM codebook. 84
Figure 6.4 A BM receives two inputs from below 86
Figure 6.5 A three-BM network 90
Figure 6.6 Performace of V_{Lin}, V_{Hout}, V_{Hin} and V_{Lout} for BM1 and BM2, and the performance of BM3 output 92

Abstract

Associative Memory as a Bayesian Building Block

Shaojuan Zhu

Doctor of Philosophy
Division of Biomedical Computer Science within
The Department of Science & Engineering
and the Oregon Health & Science University
School of Medicine

August 2008
Thesis Advisor: Dan Hammerstrom

The learning in neural systems is the formation of memory by changing the connections and their strengths in the circuits of the brain. Memory in the brain is fundamentally associative in that a new piece of information can be recalled if it is associated with previously acquired knowledge. The more associations that exist, the more meaningful the new information is, and the more efficiently the information can be stored and utilized later. Forming associative memory is one of the basic functions implemented by the brain, creating extensive mappings of the world throughout the brain. In addition to associative processing, there is growing evidence that neural systems represent probabilistic data and then performing inference over this data in a Bayesian manner.

The Palm network is one of the simple associative memories that is easy to implement, has reasonable behavior, is shown here to approximate Bayesian inference, and has some biological inspiration. This thesis discusses a computational model of associative memory based on the Palm network. The goal of the research is to investigate the computational characteristics of the model as a biologically plausible building block for a cognitive system with the specific goal of implementing Bayesian inference in a distributed manner.

We explore some of the computational capabilities that are provided by biological associative memory. We will first study the performance of the associative model in very large networks, and explore its performance with respect to stability, fault tolerance, and parallel processing of the network. We propose a theoretical interpretation of the basic operation of the model, where the Palm network is functionally equivalent to a Voronoi classification. During dynamic learning, where training vectors have different prior probabilities, weighted Voronoi regions are formed. We then show the capability of such networks to learn complex non-linear functions in the context of the reinforcement learning, where, through the interactive reaction with the outside world, the adaptive mappings of the outside world to the inside neuron activities are formed. As relatively large network is plausible for such a model, a spiking neuron representation of the network will be discussed to investigate the temporal learning of the network. Based on the Voronoi classification analysis, we propose a Bayesian Memory model which provides a maximum entropy data reduction from an input representation to an output representation. With a network of hierarchically connected Bayesian Memories, the scalability is significantly enhanced, and the resulting bidirectional information flow allows high level information to be incorporated in the decision process.

With all the characteristics discussed above, it is possible to build hierarchically connected associative memories that mimic a cognitive system. Although associative memory and Bayesian inference are not new concepts, to incorporate these ideas as the general basis for modular, scalable associative memory is not mentioned in the current literature. The main contribution of the thesis then is that we define the theoretical basis of Palm network as a reasonably applicable associative memory, and, based on the simple Vector Quantization concept, we propose the Bayesian Memory that incorporates all the necessary features of an associative memory, and, as a building block, provides a solution to build large associative networks that are in the scale of a cognitive system.

Chapter 1

Introduction

1.1 Biological Implication

Since the first electronic computer was invented in the late 1940s, computer size and speed have changed dramatically while the structure has changed little. The computer still uses the von Neumann stored-program methodology, and is entirely controlled by programmers. Computers can perform many rule-based tasks well, and can even defeat a human opponent at chess, but cognitive tasks that come easy to human beings such as recognizing a face, following a conversation, driving a car, or planning a series of motions, are beyond the capabilities of current machines, in spite of the phenomenal increases in computer speed and memory size. Computers still cannot automatically infer, predict or draw conclusions based on fuzzy, chaotic data. Obviously, it is not calculating speed alone that plays the key role in cognitive procedures. The underlying processing structures or mechanisms of actual cognition in human are far different from the computer system. Although a computer can be trained to learn thousands of millions of examples, it can not automatically generalize from past experiences in order to learn complex tasks such as driving a car.

Many scientists believe that association is one of the most basic functions underlying cognition. Recognition involves the retrieval of a stored abstract concept. A concept in the brain is not stored in the neurons themselves, rather, it consists of huge numbers of connections, or associations, among the neurons throughout the brain. Given some inputs, the responses in some cells may trigger responses in others which, in turn, may trigger more in others until these responses eventually lead to the recognition of the input stimuli. The recognition is formed via past experiences and learning. The learning

process establishes those connections between the given stimuli and the subsequent outcomes, so that later, when similar input is given, the corresponding outputs can be invoked by those established associations.

Neural association involves highly parallel interconnect among different regions in the brain. There are complex electrical and chemical processes involved in this process, and many of the mechanisms are still not understood. It is currently impossible to incorporate all the details into one model, and generally a detailed model is not necessary for implementing real applications. It is believed that associative memory fits the cortical network most admirably (Braitenberg and Schuz 1998; Johansson 2004). Associative memory is a computational model for the mechanisms in real neural circuits of associative neural processing. The study of these computational models provides a way to understand the brain by simulating nervous systems, and it also provides insight into designing a new generation of computing tools that are more powerful and intelligent. The goal of the thesis is to formalize the computational characteristics of generic associative memory as a basic cognitive function and apply such structures to applications. Generic associative memory is important in its own right, but also it is a useful model of the large, complex association areas found throughout the brain.

1.2 Associative Memory

Associative memory (AM) is a system that stores mappings from input representations to output representations, so that when the input is given, the output can be reliably retrieved. More importantly, when the input is incomplete or noisy, the AM is still able to return the output corresponding to the original input, which is a “best match” procedure, where the memory selects the input vector with the closest match, assuming some metric, to the given input, then returns the output vector for this closest matched input vector. The “exact match” association, as in the traditional content addressable memory (CAM), returns the stored value that corresponding to the exactly matched input.

There are two kinds of association operations: auto-association and hetero-association. In auto-association, the input and output spaces are identical and an input vector is associated with itself. Normally in auto-association the output is fed back to the

input until the memory converges, like an energy minimization model converges to some stable state. In hetero-association, an input vector is associated with a completely different output vector. Many cognitive procedures implement hetero-associative mapping, where one region of neurons can activate several groups of neurons in different cortical regions, and each of those regions may have different cognitive functions.

Best match associative memory is different from traditional content-addressable memory (CAM). A CAM holds a list of vectors which are distinguished by their addresses, when a particular vector is needed, the exact address of the vector must be provided. In best-match associative memory, vector retrieval is done by matching the contents of each location to a key. This key could represent a subset or a corrupted version of the desired vector. The memory then returns the vector that is "closest" to the key. Here, "closest" is based on some metric, such as Euclidian distance. Likewise, the metric can be conditioned so that some vectors are more likely than others, leading to Bayesian-like inference.

In the best match associative memory described in this thesis, the vectors are distributed represented and are stored in a distributed manner, so when more vectors are added, no extra memory is required (up to a point). In traditional memory, vectors are stored explicitly, with each set of bits occupying a unique location. In a distributed data representation, the attribute of a vector is represented by a few active neurons across all the neurons, and every active neuron corresponds to a few different vectors. Distributed representations have the advantage of fault tolerance (Field 1999) required by best match association, and make it possible of huge representation capacity.

1.3 Brief Review of Associative Memory

The investigation of associative memory began in the 1960s when the first associative memory model called *Die Lernmatrix* was introduced by Steinbuch (Steinbuch and Piske 1963). In Die Lernmatrix, both the inputs and outputs are binary vectors, the weights are adjusted via Hebbian learning, and binary values are formed. The performance of Die Lernmatrix was later studied by Willshaw (Willshaw, Buneman et al. 1969) and Palm (Palm 1980). The learning rule used in Lernmatrix is now called

"clipped" Hebbian learning rule, where the weight modification can simply implemented by the outer product of input and output vectors.

The *Associative Net* was developed by Willshaw's group (Willshaw, Buneman et al. 1969; Buckingham and Willshaw 1992; Graham and Willshaw 1994) is later referred to in this thesis as the Willshaw model. It basically is the same as Die Lernmatrix. Willshaw proves that high information storage capacity can be achieved in an Associative Net when sparsely encoded binary vectors are used. In a later study, they showed that the incompletely connected Associative Net is functionally analogous to some cortical ability in that it is tolerant of connection failures and it demonstrates short-term memory (Henson and Willshaw 1995; Graham and Willshaw 1997b). Based on the original Associative Net, different threshold control techniques are proposed (Buckingham and Willshaw 1993; Graham and Willshaw 1995; Graham and Willshaw 1997a; Graham and Willshaw 1999) to improve the retrieval performance and maintain efficient usage of information in the network synapses.

The same asymptotic information storage capacity of the Willshaw model was later proved in a different way by Palm (Palm 1980), who models the associative memory as a communication channel. In the Palm model, the information stored in the network is defined as the knowledge gain obtained from the retrieved results of the network. In the Palm model, real-valued weights and iterative structure are also used (Palm, Schwenker et al. 1997), and it has been shown that the iterative retrieval strategy can improve retrieval performance (Schwenker, Sommer et al. 1996; Sommer and Palm 1998). In their later research, Palm and his group developed a parallel processing chip called Bacchus (Palm, Schwenker et al. 1997), and later a biologically plausible spiking model is used in simulations of the primary visual cortex (Knoblauch and Palm 2001a; Knoblauch and Palm 2001b). For the binary model where no iteration is used, the Palm model is the same as the original Willshaw model.

Another non-linear binary associative memory model is *brain-state-in-a-box* (BSB) proposed by Anderson et al. (Anderson, Silverstein et al. 1977; Anderson 1993). Positive feedback is used in BSB so that the attractors are saturated at the corners of the binary hypercube that spans the network's state space. Based on BSB, a more complex model has been developed where several parallel networks compose a "network of

networks" structure (Anderson and Sutton 1995). In the more recent "Ersatz Brain Project"(Anderson, Allopenna et al. 2007), the "network of networks" techniques are used in building parallel, brain like computers in both hardware and software.

The Hopfield network invented by John Hopfield in the 1980s has been a popular study of research (Hopfield 1982). In this network, an energy function is proposed in the computation of the recurrent networks with symmetric synaptic connections. Although the information capacity is relatively low, and there are spurious attractors, the Hopfield network built a bridge between neural networks and physics, and it provided a theoretical derivation for the stability in a dynamical network.

There is another well-known associative memory model called the *self-organizing map* (SOM) proposed by Kohonon (Kohonen 1982; Kohonen 1989). SOM creates a one- or higher- dimensional lattice that captures the topographic structures of the input space. SOM has some biological plausibility, for example, different sensory inputs are mapped onto different areas of the cerebral cortex in a topologically ordered manner.

The statistical characteristics of dynamical associative memory (DAM) are further discussed by Amari (Amari and Maginu 1988; Amari 1989). Amari claims that, in a dynamic or iterative associative memory, a strange attractor basin shape is formed where some input vectors are not attractors.

Recently, there have been renewed associative memory models. The *bidirectional associative memories* (BAM) (Kosko 1988) and *sparsely distributed memory* (SDM) (Kanerva 1988) are two other models. A good reference for those associative neural models is the edited book by Hassoun (Hassoun 1993). In this book, a general criterion for high-performance associative memory is introduced, and the dynamic associative neural memory (DAM) is defined as a network structure where the network output is fed back to the input.

Another important model is *Bayesian Confidence Propagating Neural Network* (BCPNN) (Lansner and Ekeberg 1989; Sandberg, Lansner et al. 1999; Sandberg, Lansner et al. 2000; Sandberg, Lansner et al. 2002). BCPNN has a Bayesian inference property, can avoid catastrophic forgetting, and can implement a "second match" capability. A hyper-column BCPNN (Sandberg, Lansner et al. 2000; Johansson, Sandberg et al. 2002; Sandberg, Lansner et al. 2002) resembles the hyper-column structure in cerebral cortex,

but it requires high-precision weight values. Based on their hyper-column structure, a patchy connected attractor network can result in a higher storage capacity (Johansson, Rehn et al. 2006).

Though a number of associative memory models have been defined, practical applications of these models are far less common. One successful application of an associative memory model is implemented by Cortronic networks (Hecht-Nielsen 1999; Hecht-Nielsen 2003b; Hecht-Nielsen 2003a). Cortronic networks have been used in English text recognition. The underlying model of the Cortronic network is explained by the thalamocortex theory proposed by Robert Hecht-Nielsen, which also incorporates aspects of the Palm model and Bayesian networks.

Another application of the ideas of cortical operation is done at Numenta (Hawkins and Blakeslee 2004; Hawkins and George 2006; George and Jaros 2007). Based on neuroscience principles, Jeff Hawkins proposes that cortex operates auto-associatively, learns sequences of patterns, captures invariants and organized hierarchically. The information flow within the hierarchical structure is based on Bayesian principles. Our hierarchical network is based on some of Numenta ideas.

1.4 Thesis Organization

The research goal of this thesis is to investigate the characteristics of associative memory as one of the building blocks for more complex cognitive machine learning systems. There are several associative models available for approximation of cognitive association. Palm model is one of them. We use the Palm model as a starting point in our research. It is simple, can be easily implemented, and is relatively straightforward to analyze. It has some biological plausibility in sparse encoding, fault tolerance and high memory storage capacity. Despite these characteristics, direct application of the model in real applications faces many practical difficulties, such as the need to use sparse representations and the high connection density. In particular, the possibility of the extending the Palm model to a large network in the scale of cortex is rarely discussed in previous literatures, in contrast, the huge number of neurons in cortex is one of the key elements in cognition.

In this thesis, we investigate several characteristics of associative memory that are essential for a building block in a complex cognitive system. A real cognitive system should have large storage capacity, be able to use the temporal information, have online learning capability, and most importantly, have the potential to scale up to large and hierarchical structure.

In the Palm model, the sparse representation implies that the network size could be large. However, as the network size grows, scaling issues have a significant impact on both the performance and the hardware implementation. In Chapter 2, we will investigate the network performance of very large networks. We have implemented and simulated large Palm networks on both serial and parallel processing platforms, since we believe that large networks are essential when modeling cognition and when using association networks in real applications.

In depth discussions on the Palm network capacity and performance improvement can be found in a number of research papers. However, how such a model is utilized in real world applications is less clear, and the theoretical interpretation of the behavior of the model is not specifically addressed in previous literatures. In Chapter 3, we investigate the functional capabilities of the Palm network, and propose a simple and useful theoretical model of the operation of the network, based on the fact that the computational characteristics of Palm network are similar to that of Vector Quantization, where, for each training vector, Voronoi regions are formed under certain conditions, and the Palm network can retrieve an output vector that has approximately the maximum posterior probability.

Cognitive ability is acquired by constantly learning. Therefore, the ability to perform online learning is very important to the ultimate usefulness of these models. Chapter 4 will investigate the reinforcement learning based associative memory that has the ability to learn complex non-linear data spaces.

Chapter 5 will discuss a spiking version of the Palm network. Temporal information is critical in cognition. Many researchers believe that the inter pulse interval encodes information in cortical neurons. Although not many differences exist in the shape of the post-synaptic potential in different neurons, the relative firing times may make big a difference in adjusting the efficacies between two neurons. The spiking

representation allows us to factor temporal information into network operation. Also, spiking representations can often lead to very simple implementations.

To solve the scaling issue of the large networks, Chapter 6 proposes a solution of building a hierarchical network of Bayesian Memories, where the maximum entropy codebook-like mappings are formed. With hierarchically connected Bayesian Memories, feedback information from high level network provides additional information for the low level networks so that they can converge to a stable state more easily.

And Chapter 7 summarizes the results of the thesis, and discusses some future research directions.

Chapter 2

Large Network Simulations

2.1 Aims

The objective of the large network simulation is to investigate the performance and stability of the model as the network size increases. Most studies of associative networks present simulation results with relatively small networks, and it is often implicitly assumed that these networks will scale to large sizes with similar results. Yet few models have been tested on very large networks. This is partially due to the limitation of computer resources when the work was performed. In this chapter, we will investigate the plausibility of large network simulations and parallel implementation. As a part of this exploration, we are concerned about several issues including performance, stability, computer resource utilization, and computational complexity.

2.2 Palm Network

Our simulations are based on the basic model due to Palm. The algorithm in the Palm model is basically the same with Willshaw model, except that the recurrent non-binary associative memory structure is also proposed in the Palm model. Palm (Palm 1980) derived the information storage capacity from an information theoretic prospective, and his result is consistent with the asymptotic results derived by Willshaw (Willshaw, Buneman et al. 1969).

The asymptotic capacity defined in the Willshaw model is $\ln 2$ bits/synapse (Willshaw, Buneman et al. 1969). This capacity is an efficiency measure of the storage capacity, and it is defined as the ratio of the total information in the output vectors to the

number of synapses in the network. If the associative memory is regarded as a communication channel, this capacity coincides with the information storage capacity (Palm 1980; Palm, Schwenker et al. 1997) as the bound of the information channel.

In the Palm model, the set of input vector to output vector mappings to be stored in the network are $\left\{\left(x^{\mu}, y^{\mu}\right), \mu = 1, 2, \cdots, M\right\}$. There are in total of M mappings, and both x^{μ} and y^{μ} are binary vectors with size of m and n respectively. x^{μ} and y^{μ} are sparsely encoded, with $\sum_{i=1}^{m} x_i = l$ ($l << m$) and $\sum_{j=1}^{n} y_j = k$ ($k << n$). l and k are the numbers of active (non-zero) nodes in the input and output vectors respectively.

For a binary network, in the training procedure each pair of the mapping (x^{μ}, y^{μ}) is presented to the network: the address vector x^{μ} at the input, and the corresponding content vector y^{μ} at the output. The "clipped" Hebbian learning rule is applied to store all the mappings in the network, and this binary weight matrix W can efficiently be implemented by:

$$W = \bigvee_{\mu=1}^{M} \left[y^{\mu} \cdot \left(x^{\mu}\right)^{T} \right] \tag{2.1}$$

where $\vee$ is the OR operation, and $\cdot$ is outer product between two vectors.

In the recall procedure, for a given key or address $\tilde{x}$, which may be the noisy version of pattern x, is applied to the input of the network. The output pattern retrieved by this network W is

$$\tilde{y} = f(W \cdot \tilde{x} - \theta) \tag{2.2}$$

Where $\cdot$ represents inner production between a matrix and a vector, θ is a global threshold, and $f()$ is the Heavyside function, where an output node will be 1 (active) if its dendritic sum $x_i = \sum_{j=1}^{m} w_{ij} \tilde{x}_j$ is greater than the threshold, otherwise it is 0.

To set the threshold θ, the "k winners take all (k-WTA) rule" is used, where k is the number of active nodes in an output vector. Therefore, the threshold θ is set so that only those nodes that have the k maximum dendritic sums are set to "1", the remaining nodes are set to "0". The k-WTA rule adds competitive (inhibitory) interaction among the

output neurons. The k-WTA threshold is very close to the minimum error threshold (Buckingham and Willshaw 1993). With binary vectors and weights, the dendritic sums are integers, and due to many numbers taking the same value, it is possible that the system cannot find exactly k maximum sums. In this case we use the next value of threshold that gives the maximum number of active nodes more than k.

The Palm model is an attractor model in that its state space creates an energy surface with minima ("attractor basins") occurring when the state is equal to a training vector. Both Palm (Palm 1980) and Willshaw (Willshaw, Buneman et al. 1969) have shown that when k is $O(\log n)$, the asymptotic pattern capacity of ln2 bits per synapse can be obtained. When the highest pattern capacity is achieved, one half of the synapses in the weight matrix are 1s and the other half are 0s. It is possible to store $M>>n$ patterns with very small retrieval error probability. Though the pattern capacity is a measure of the maximum information stored in a matrix, it is not guaranteed that all that information is accessible by retrieval. Iterative retrieval can improve the retrieval performance (Wennekers, Sommer et al. 1995; Schwenker, Sommer et al. 1996; Sommer and Palm 1998), and, in most cases, only a few iterations are needed for convergence. Other techniques using adjustable threshold can also improve the retrieval performance (Buckingham and Willshaw 1993; Graham and Willshaw 1995).

Sometimes it is confusing that different measures of capacity are used in different papers: some use the maximum number of patterns being stored as capacity, some use the maximum number of patterns per neuron as capacity, and some use the maximum information stored per synapse as capacity. One unifying measure called *information efficiency* can be used to compare the capacity in different models. Information efficiency is defined as ratio of the information capacity per synapse divided by the minimum number of bits required for each synapse (Sommer and Dayan 1998). Here, the *information capacity* is a measure of the channel capacity per synapse, and is defined as the information gain achievable by recall. It has been shown (Palm 1988; Palm and Sommer 1992) that the Palm model stores information efficiently in terms of storage capacity per physical bit. However, for maximum information efficiency, the network is ideally half full. In this case, the fault tolerance is diminished, reliable retrieval is not guaranteed, and the connectivity is $O(nm)$.

Palm and Sommer (Palm and Sommer 1992) prove that, for both the clipped Hebbian rule and the additive Hebbian rule, the upper bound of pattern capacity or storage capacity (in patterns per neuron) is twice as much as the upper bound of information capacity (in bits per neuron). For one-step auto-associative retrieval, the asymptotic information capacity is $(\ln 2)/2$ for the clipped Hebbian rule, and $1/(4\ln 2)$ for the additive Hebbian rule. However, both values can not be achieved with iterative retrieval starting from an incomplete or noisy version of the training vector. With incomplete or noisy input, the information capacity for practical iterative retrieval procedures will be lower, since many intermediate states map to the stored vector. It has also been shown that the binary Palm model has the highest information efficiency $(\ln 2)/4 = 0.17$ among the variations of Palm model and Hopfield model, with dense coding, linear/clipped learning, or a general nonlinear function for weight saturation (Sommer and Dayan 1998).

2.3 Simulation Tool - Csim

For some of the work described, we developed the Csim simulator, which performs complex simulations and also allows execution on a parallel machine. It is programmed in C++. The basic structure of Csim is the Pathway object, which is a template for all other derived objects, and operates as a communication region with internal data storage. Pathway objects contain one or more vector arrays. The entire simulation is vector oriented.

The various objects required in a simulation are derived from the Pathway. These derived objects are either communication objects that perform scatter and gather for parallel processor implementation, or operators that operate over the vector inputs, such as an inner product or k-WTA on data obtained from the input pathways to the object. A simulator is created by instantiating these derived types and connecting them together. This collection of objects is under the control of the Timing and Control (TC) module, which initiates the parameters, sets connections between the different Pathways, runs the simulation, and deletes the objects after simulation. Csim has an interactive interface, which is mainly used for debugging, but is normally executed in batch mode. Data output

can be generated for analysis and graphical representation using Matlab. Csim is batch oriented and compiles and executes on Windows (Visual C++), Linux (g++), and SGI Unix.

Csim can execute on a single processor or on an arbitrary number of multiple processors. A special set of pathway objects are available which use MPI (Snir, Otto et al. 1996) to subdivide the problem and enable inter-processor communication. MPI, the Message Passing Interface, is a standard for message communication among multiple processors. In multiple-processor mode, the pathways are partitioned automatically by the simulator. For more complex simulation, we could also partition the system among the pathways.

For the experiments performed for this work, the Csim simulator was configured to operate in three main phases: 1) training, 2) testing and 3) computing statistics. In the experiments discussed here, artificial training and test vectors are used. However, Csim can also take data from real-world applications.

In the training phase, a set of training vectors is generated, and the weight matrix is formed after feeding all the training vectors. If the multi-process version is used, the training vector is generated in the root process, then the root process broadcasts the vector to all other processes. In Csim, each processor has one process, which is the common configuration in most MPI run-time environments.

The test vectors are noisy versions of the training vectors, and are generated by randomly flipping some bits in the training vector, with 1 becoming 0 and vice versa. For simplicity and in keeping with the basic Palm model, the number of 1s in the test vector is kept constant *k*. The number of bits that are flipped in the test vector is calculated according to the given noise probability (bit-flip probability). In multi-processor mode, the root process creates each test vector and broadcasts it to the other processes, which calculate an inner product of the test vector with the weight sub-matrix for the nodes on that process. The k-WTA (for the global *k*) is executed in parallel on all processes to reduce the data amount communicated between each process (at most *k* * number of processes). The root process receives each process's *k*-element vector and does another k-WTA to get the final *k* global winners. A number of approximations are possible such that each process can reduce the number of elements in its k-WTA, but by using k we

guarantee that the parallel version and the sequential version generate the exact same results which may not be necessary in a real application.

Statistics for each memory output (and the corresponding test input) are collected to evaluate the performance of the network. The Euclidian distance between the training vectors and the output vectors, and the distance between the training vectors and the test vectors are used to determine the information "gain" of the network in terms of reduced noise at the output, that is, Gain = $1- D_{train\_out} / D_{train\_test}$, where D_{train_out} and D_{train_test} are Euclidian distances.

2.4 Simulation Results

2.4.1 On PC and Beowulf Cluster

For most of our experiments we used a Linux based Beowulf cluster that consists eight 1GHz Pentium III processors, each with 512MB memory. The processors communication via a 100BaseT based router. This router is on a separate subnet, so that the only real traffic on the router is from the cluster. The parallel version of Csim can be executed on this cluster. Both C and C++ calls for MPI are available in the cluster.

A network of 64K neurons was trained on 100K training vectors, with each vector having 200 active nodes. When 100 test vectors, with 10% bit-flip probability, are given as the input, the corresponding training vectors are recalled perfectly. In such a simulation, the cluster required 6 minutes 20 seconds (11.2% is used for inter-processor communication) for training, and 53 seconds (1.3% for inter-processor communication) for retrieval. For the same 64K node network, the performance corresponding to different numbers of training vectors is shown in Figure 2.1. The number of active nodes in the training vectors is always 200, and 100 test vectors with 10% bit-flip probability are used.

The *performance* (or Information Gain) of these networks, which is a measure of information gain due to recall, is defined as:

$$\text{Performance} = 1- D_{train\_out} / D_{train\_test}$$

where D_{train_out} represents the average distance between the training vectors and the output vectors, and D_{train_test} represents the average distance between the training vectors and the

test vectors. When the Performance is 1, all the training vectors are retrieved perfectly; when it is negative, the network does not provide any information gain, and in fact, loses information.

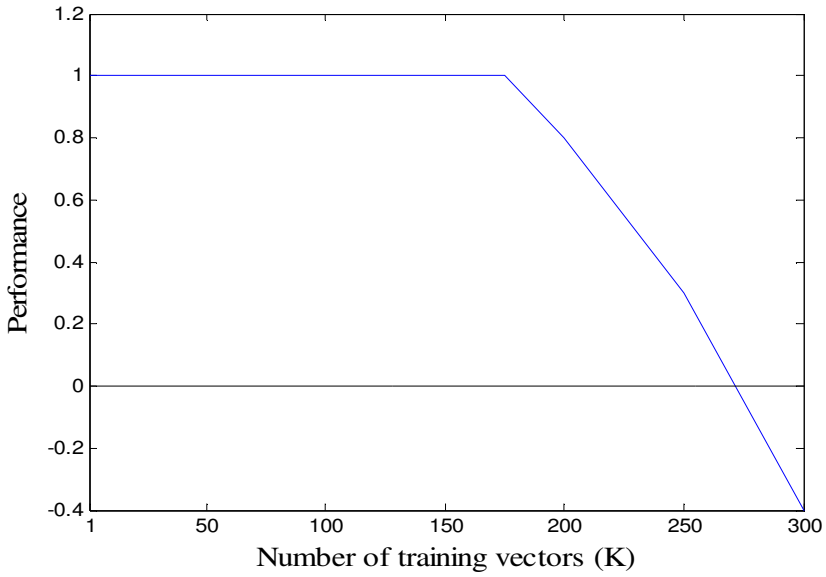

Figure 2.1 Retrieval performance versus the number of training vectors being stored

In Figure 2.1, for 250K training vectors, even though not all the outputs are perfectly recalled, the information gain is positive. When more training vectors are stored, the network becomes overloaded, eventually there is too much interference and the performance drops rapidly. For 45K training vectors, when the bit-flip error rate approaches 50% (not shown in Figure 2.1), the retrieval is still perfect.

The same 64K node network with 45K training vectors was tested by 100 test vectors with 10% error probability. Here the active number of nodes in the training vectors is varied, and the performance is shown in Figure 2.2. When there are 500 active nodes in each training vector, the retrieved vectors are not perfect, but, on average, there is still positive information gain. When the number of active nodes is extended to be larger than 575, the network is too dense to effectively recall the output.

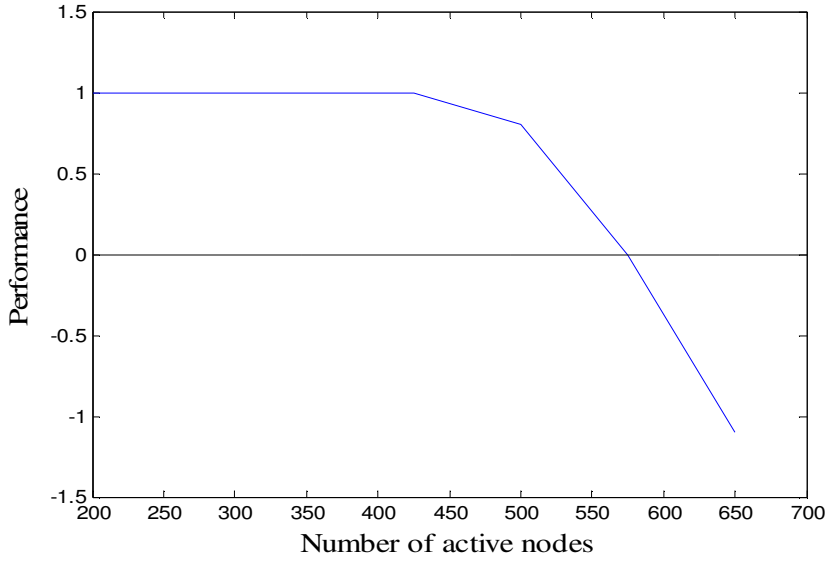

Figure 2.2 Retrieval performance versus the number of active nodes in the training vectors

The simulation results in Figure 2.1 and Figure 2.2 show that, the network density is an important parameter in determining the network performance. There is a more detailed discussion on how the density influences performance in Chapter 3.

To determine the efficiency of the MPI system, both the time spent in MPI routines and the total simulation time were collected. Most of the MPI time is spent on waiting for information to be communicated. In the simulation, the number of training vectors is set fixed as 45K, and each has 200 active nodes, but the vector size of training vectors for each network varies. The number of test vectors is fixed at 100. With the same amount of active nodes in each vector, the larger the vector size is, the sparser the network. However, for different sized networks the total amount of data that are communicated among processes is roughly the same. As the network size gets larger, for roughly the same amount of data is communicated, and although the MPI waiting time increases, the time spent computing increases more quickly, so the percentage of the total MPI time actually decreases. Figure 2.3 shows that the larger the network, the less percentage of time is spent on inter-processor communication. This implies that the parallel computation can speed up the computation speed, and can improve the computation efficiency in large network.

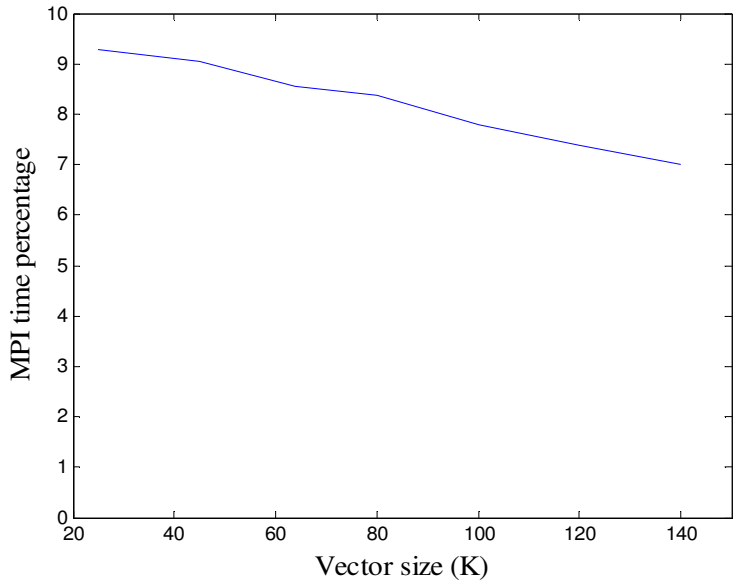

Figure 2.3 The relationship between the MPI percentage and the vector size

The largest network simulated on the Beowulf cluster had 128K neurons (the vector length) and 100K training vectors, each with 200 active nodes. The training procedure required 6 minutes, of which 11.2% is spent in MPI intercommunication. To retrieve 100 test vectors with 10% error probability, 1 minute and 55 seconds were required, only 0.5% of the time was in MPI intercommunication.

The simulations run on the cluster indicate that: the larger the network and the sparser the vector, the more vectors can be stored and the better retrieval performance will be, though as we shall now see as networks get larger than this, performance actually degrades. This also leads to more efficient parallel processing.

2.4.2 On NASA Supercomputers

Initial simulations were done on the Beowulf clusters, and larger simulations were done on NASA's SGI 2000 Origin Supercomputers at the Ames Research Center. The two we used were, Steger, with 256 processors and 64GB memory in total, and Lomax with 512 processors and 192GB memory. Both machines are based on the 250 MHz MIPS 12K processor connected via high-speed inter-processor communication (1.2GB/node). The Origin 2000 is a shared-memory "NUMA" (Non-Uniform Memory

Access) architecture. Due to long access queues, most simulations were of 256K neurons, however, we did occasional simulations of 512K nodes.

As expected, the behavior of the larger simulations could be extrapolated from the smaller. For example, for a network of 256K nodes and 180K training vectors with 40 active nodes, when presented with test vectors with 20% probability of error, all training vectors were recovered perfectly. The execution time (64 processors) is 4 min 7 secs, and 13% of that time was spent in MPI calls – a large number considering the interconnect bandwidth of the Origin 2000.

Interestingly the largest simulations (with the network size of 512K) were not as robust as we thought they would be. They were more sensitive to the number of active nodes and the total number of training vectors. Part of the reason could be quantization effects that occur during the global k-WTA operation. Using limited precision and binary inputs and weights, many nodes will have the same value after the inner product operation. In this case it is impossible to use the adaptive threshold deterministically to get exactly K winners. Winners can be chosen randomly, but that does not increase information gain.

2.4.3 Discussion

The simulation results from both the cluster and the super computers demonstrate that the larger the network size and the sparser the network, the better the retrieval performance is. However, this is not to say that, for any noise level of a test vector, the network can always return a best matched input, even for a large network size. The maximum noise level allowed in a network to guarantee a convergent recall is represented by attractor basin surrounding the training vector. Theoretical analysis of attractor basins will be presented in Chapter 3, where the relationships among the network size, the number of vectors, the noise level and the retrieval performance are investigated.

Larger networks have the potential to store more vectors, but it is possible for the network become too large, where implementation becomes more difficult and expensive and results start to deteriorate. Even with parallel processing, as the number of vectors being stored gets larger, more time is spent on waiting for communication. Scaling does

not necessarily set a hard limit on performance if a parallel model is used, but, for implementation and computation efficiency, large network size will obviously limit the use of the model in a real application. In addition, as shown in (O'Kane and Treves 1992b), the traditional association models are inadequate as models of cortex, the number of patterns that could be stored and retrieved would not scale with the size of network. In Chapter 6, a modular association model will be proposed to address the scaling issue.

The Palm model is a fully connected network. In cortical systems, according to Braitenberg (Braitenberg and Schuz 1998), there are two kinds of connections: metric and ametric. The metric connections are very dense connections to a node's local neighborhood. The ametric connections are much sparser, random, point-to-point connections to densely connected groups. Though the exact neural circuitry is still not known, a hierarchically connected network is potentially a more powerful model for complex applications, and evidentially, it is a preferable structure in many biological systems. A hierarchical network structure will be discussed in Chapter 6. It is hypothesized that large groups of modules may have different learning and connectivity patterns, where each module specializes on certain types of functionality, creating a larger, more complex system.

2.5 Computational Complexity

With appropriate parameter settings, the Palm-Willshaw model is robust to noise and can return an output that is the best match to the given input. This mapping function is very useful and can be used in a number of applications. For example, it is similar to the nearest neighbor rule, where the given input is always mapped to the closest training vector. Here we can compare the computational complexity of the Palm model with the brute force approach. The brute force method of matching is essentially a lookup-table searching approach, and the memory formed is referred to as the Hamming associative memory (Hassoun and Watta 1996).

The Palm network settings are: the vector size is set as n, and there are k 1s in each vector, with k in the order of log n. The total number of vectors to be stored is m.

In the Palm model, to compute the best match for a given input, only k columns in the weight matrix are needed, with each in size of n. Therefore, to get the dendritic sum

vector, O(kn) operations are needed. To implement a k-WTA, a sort of O($n$log$n$) operations is required. For the 1-WTA used in the hypercolumn structure of BCPNN, O(n) operation is needed, which is computationally more efficient in this sense. The total operation needed to retrieve a best match vector is O(kn)+O($n$log$n$), that is, O($n$log$n$).

In Hamming associative memory, to do an exhaustive search of a lookup table, the given input vector has to compare its Hamming distances to all the stored vectors. The Hamming distance calculation is similar to inner-product operation, and is in O(n). There are m vectors, so the total calculations for Hamming distance is in O(mn). After all the distances are available, a maximum operation is needed to choose the vector with the minimum Hamming distance, or the maximum inner production. This maximization needs O(m) operations. Therefore, the total number of operations in the brute force method is O(mn)+O(m), that is, O(mn).

Comparing the brute forced method with the Palm model, we can see that when m>log$n$, the operation in Palm network is more efficient than brute force. When m is small, a look-up table is an easier way to implement a search, and the storage required (O(mn)) is less than that of the Palm network (O(n^2)). In the applications where the number of vectors to be stored is much larger than the network size (m>n), building a Palm network is computationally more efficient in both storage and retrieval via the distributed storage technique. Moreover, fault tolerance can be expected in Palm model if distribution of representation is used in the vectors, and if we build a system that uses distributed representations, retaining the distributed representation throughout the system requires that we use computational methods such as Palm's that can operate on such representations. Fault tolerance allows input noise in the sense that many input patterns which are "close" to the training vector in some metric (here Hamming distance) can be mapped to the content vector belonging to that address. In addition, it has been shown that the Palm network has fault tolerance to errors in internal storage, such as connection failure, which will be discussed in Chapter 3. It also has been shown in (Graham and Willshaw 1997a) that, when a small number of connections in the network are missing, network performance experiences only minor degradation, still doing a reasonably efficient retrieval operation. Such tolerance is especially beneficial to hardware implementation.

Chapter 3

Voronoi Classification in Associative Memory

3.1 Application Issues

The simulation results in Chapter 2 demonstrate the large storage capacity and fault tolerance of the Palm model. It has been shown (Palm 1988; Palm and Sommer 1992) that the Palm model has the highest information efficiency among several models. However, for maximum information efficiency, the network is ideally half full. In this case, the fault tolerance is diminished, and reliable retrieval is not guaranteed. The high information capacity has little use if there is significant cross-talk among training vectors.

Many techniques have been proposed to improve the performance of the original Willshaw associative net, and to relax the requirement of sparse coding in the training vectors. The threshold-control technique developed by Willshaw's group (Buckingham and Willshaw 1993; Graham and Willshaw 1995) can improve the retrieval performance when the input vector has a relatively high error rate. Based on different threshold control techniques, a partially connected network is also feasible. Both analytical results (Buckingham and Willshaw 1993) and numerical calculations (Graham and Willshaw 1997a; Bosch and Kurfess 1998) show that a partially connected network has reasonably good information capacity and information efficiency. Iterative retrieval techniques (Gibson and Robinson 1992; Schwenker, Sommer et al. 1996; Sommer and Palm 1998), based on attractor or energy minimization dynamics, can also improve the network performance. Bayesian inference theory (Sommer and Dayan 1998) provides a theoretical foundation for most of the above techniques in performance improvement and resistance

to connection failures, though the paper does not specifically address the issue of convergence of these vectors.

Although a number of associative memory models have been proposed over the years, most of these discussions have focused on network capacity, information storage and retrieval improvement. The theoretical interpretation of associative memory behavior of these associative models, when applied to real world applications, is not generally addressed. It is important to know how such networks behave in real applications based on massive quantities of noisy data. How much information can be stored is less important than how robust the model is in performing the expected function in a specific application. For example, in our Enhanced Vision System (EVS) for aircraft landing guidance (Luk, Gao et al. 2004), the training vectors are not random and are actually grouped close to each other in one region of the state space, which reduces the memory capacity and the retrieval accuracy. Given some number and distribution of vectors, how well can the network retrieve those vectors from noisy inputs? And how many defects in the implementation can the network tolerate and still be useful? Previous work on associative memories have not really addressed these issues nor provided any quantitative measure of how much noise the model can tolerate and still provide acceptable retrieval results. Also in real applications the definition of "good enough" retrieval often changes dynamically. All of these issues can be summarized as, what is the relationship between the number of vectors stored and the maximum allowed error rate? Even though iterative retrieval can provide better retrieval performance, the quantitative measure is still not discussed in the literature. A typical question is: when a network is operating with sufficient information efficiency, will the network demonstrate good performance on any noisy input?

In this chapter, we try to answer those questions. We define a Bayesian model of the operation of the Palm associative memory. We will discuss the computational characteristics of the Palm network, especially the general functional analogy of the Palm model with Voronoi classification. And we will provide a theoretical interpretation of the behavior of the operation of the Palm associative memory.

3.2 Communication Channel Analogy

A communication channel, see Figure 3.1, is a useful model of pattern classification. Basic association can then be modeled as a simple channel decoder shown in Figure 3.2. Under certain assumptions, it is possible to use both information theory and Bayesian decision theory to model the association process (MacKay 1991; MacKay 2003), where a pattern to be recognized is a noisy version of a learned pattern, just as a received message is a noisy version of the original message.

In this discussion, a test vector contains some genuine active bits that are the same as the original training vector, and some spurious active bits that are caused by noise. For simplicity, the total number of active bits in a test vector is set to be the same as that in a training vector.

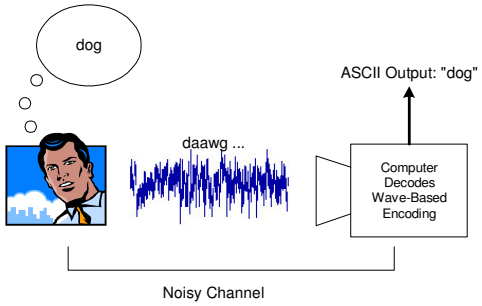

Figure 3.1 The Decoder Model of Pattern Classification

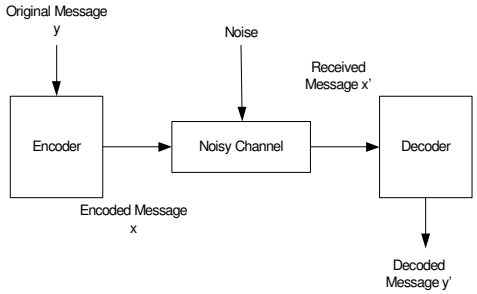

Figure 3.2 The Association Memory as a Channel Decoder

In the communication channel analogy, messages are generated with probability *P(y)*, ignoring higher order probabilities between messages, that is, the probabilities of successive messages are independent. Generated messages are encoded into a message *x* (a bitstream) that is transmitted over a noisy channel, a received message (another bit stream, the transmitted message with errors) *x'* is received and input to the decoder. The decoder has, via many examples, obtained an estimate of the probabilities *P(y)* and *P(x'|y)*. It then uses these estimates to determine the most likely *y*, given that it received *x'*:

$$P(y \mid x') = \frac{P(x' \mid y) \cdot P(y)}{\sum_x P(x' \mid x) \cdot P(x)} \tag{3.1}$$

Using this communication channel concept, we will describe the Bayesian inference characteristics of a Voronoi tessellation of the state space. In a Voronoi tessellation, the feature space is partitioned into regions consisting of all vectors closer, according to some metric defined over the space, to a given training vector than to other training vectors, and all vectors in such a region are labeled by the category of that training vector (Duda, Hart et al. 2001). In this chapter, a recursive auto-associative network is assumed, and we will show the functional analogy of the Palm network and Voronoi tessellation, and subsequently the Bayesian classification characteristics.

As presented in previous chapter, the Palm model uses a sparse data representation, and has large information capacity and fault tolerance (Graham and Willshaw 1994; Sommer, Wennekers et al. 1998; Knoblauch and Palm 2001a; Hecht-Nielsen 2003b). In this chapter, we analyze the operation characteristics of Palm network, and consider the Palm association model as a stand-alone module, though in real systems it will most likely be a single building block of a larger system such as cortical areas, where a Palm memory represents a cortical column with dense local connectivity, and the columns are, in turn, loosely connected with each other. Hierarchical laterally connected columns form even more complex and higher-order information processing units, which will be discussed in Chapter 6.

3.3 Dendritic Sum Distribution in Palm Network

The retrieval characteristics of the Palm network have been well studied earlier in this dissertation and in the literatures (Willshaw, Buneman et al. 1969; Willshaw and Dayan 1990; Buckingham and Willshaw 1992; Graham and Willshaw 1995; Palm and Sommer 1995; Sommer and Dayan 1998). The theoretical results match well with the simulation results (Buckingham and Willshaw 1992). Based on theoretical analysis, several retrieval techniques have been proposed to improve the retrieval performance, information capacity and efficiency (Buckingham and Willshaw 1993; Graham and Willshaw 1995; Graham and Willshaw 1997a; Sommer and Palm 1998). Detailed network analysis can be found in (Willshaw, Buneman et al. 1969; Palm 1980; Buckingham and Willshaw 1992). To facilitate the further discussion of Bayesian characteristics in the following sections, here we summarize the analysis.

Suppose a binary auto-associative Palm network has a size (vector dimension) of n, there are M vectors being stored, and for each vector, k nodes are active. The denditic sum d_i of an output node is $d_i = \sum_{j=1}^{n} w_{ij} \cdot x_j$, x_j is the input activity. The distribution of the d_i is determined by k and the number of connections in the network. The output node can be either active (1) or silent (0) depending on the threshold. Via the k-WTA rule, the threshold is dynamic and will be different for different input vectors.

If a node in the training vector is 0, then the corresponding node in the output vector is supposed to be 0 (has low potential), and its dendritic sum is mainly contributed by the connections that other training vectors create. The silent node's dendritic sum d_l results from noise in the test vector. Assuming k active nodes in a test vector, the d_l approximate a binomial distribution:

$$P(d_l = x) = \binom{k}{x} \cdot \rho_l^{\,x} \cdot (1-\rho_l)^{k-x} \tag{3.2}$$

with

$$\rho_l = 1-(1-\alpha^2)^M \tag{3.3}$$

where $\alpha = k/n$. ρ_l is the probability that a connection in a network is 1 in at least one training vector. ρ_l can also be regarded as the *network density*, representing the sparseness or the memory load of a network.

If an output node is supposed to be active (1), or, have high potential, the contributions to its dendritic sum, d_h, not only come from the true connections for that training vector, but also from other connections that other training vectors create. This sum also approximates a binomial distribution:

$$P(d_h = x) = \binom{k}{x} \cdot \rho_h{}^x \cdot (1-\rho_h)^{k-x} \tag{3.4}$$

with

$$\rho_h = 1 - e_i + e_i \cdot \rho_l \tag{3.5}$$

where e_i is the error rate in the active nodes of a test vector. So the input error rate in the whole vector base is $2\alpha e_i$. With Equations (3.2) and (3.4), the distribution of the real dendritic sum of an output node is:

$$P(d = x) = (1-\alpha) \cdot P(d_l = x) + \alpha \cdot P(d_h = x) \tag{3.6}$$

Both Equation (3.2) and (3.4) can each be approximated by a Gaussian distribution, therefore, Equation (3.6) can be approximated by:

$$P(d = x) = \frac{1-\alpha}{\sqrt{2\pi\sigma_l{}^2}} \exp[\frac{-(x-m_l)^2}{2\sigma_l{}^2}] + \frac{\alpha}{\sqrt{2\pi\sigma_h{}^2}} \exp[\frac{-(x-m_h)^2}{2\sigma_h{}^2}] \tag{3.7}$$

with means and variances as:

$$m_l = k \cdot \rho_l, \qquad \sigma_l{}^2 = k \cdot \rho_l \cdot (1-\rho_l) \tag{3.8}$$

$$m_h = k \cdot \rho_h \qquad \sigma_h{}^2 = k \cdot \rho_h \cdot (1-\rho_h) \tag{3.9}$$

From Equations (3.7)-(3.9), it can be seen that the distribution of the dendritic sums is determined by: the density ρ_l, the input error rate e_i, and the number of active nodes k. Accordingly, the network retrieval performance relies on these three parameters. Threshold setting techniques that utilize these parameters to improve the retrieval performance are discussed in (Buckingham and Willshaw 1993; Graham and Willshaw 1995).

3.4 Bayesian Inference from a Voronoi Tessellation

Returning to the communication channel concept in Equation (3.1), if all the original vectors are equally likely, i.e., the prior probabilities are equal, then given a received vector *x'*, the original vector *y* that has the largest conditional probability $P(x'|y)$ is the most likely vector that was sent in a Bayesian sense. It is assumed that, in the sparse encodings used for the transmitted messages, the bits are independent of each other. If Hamming distance is used as the distance metric, and the bit-flip (error) probability in a binary symmetric channel is strictly less than 0.5, then the conditional probability $P(x'|y)$ is a monotonically decreasing function of the Hamming distance between *x'* and *y*. Therefore, the training vector *y* that has the minimum Hamming distance to *x'* has the largest conditional probability $P(x'|y)$, and is the most likely vector that was sent.

The vectors that constitute the transmitted messages create a *Voronoi tessellation* over the vector space. All the vectors in one Voronoi region have the smallest Hamming distance to that transmitted vector's region. For binary vectors with iid error per bit, given a received vector *x'*, the transmitted vector *x* that forms the Voronoi region where *x'* lie, is the most likely vector that was sent in a Bayesian sense. It is then a simple mapping from *x* back to the original message *y*. Therefore, a Voronoi tessellation of the input vector space can be used to perform Bayesian inference with equal priors.

Here we assume that the transmitted messages were all equally likely. We can also define weighted Voronoi regions which address the situation where the prior probabilities are unequal.

Definition: let *S* denote a set of attractor vectors such that each vector v_i is assigned a positive and finite influence $w(i)$. In this case, the distance of an arbitrary vector v_x from v_i is given by $D(v_x, v_i)/w(i)$, where *D* denotes a distance measure. The *weighted Voronoi region* for *S* is a subdivision of the input space such that each vector v_i in *S* is associated with a region consisting of all vectors v_x in the region for which v_i is the nearest vector using a weighted distance metric. ■

In a weighted Voronoi region, all the vectors have the smallest weighted distance to the training vector that is in the same region. Via Equation (3.1), if the influence factor

w in the above definition is monotonic to the prior probability $P(y)$, and the distance $D(v_{x'}, v_y)$ is proportional to the inverse of $P(x'|y)$, then the weighted distance $D(v_{x'}, v_y)/w$ is a measure of posterior probability of y given x'. The smaller the weighted distance, the more likely y is. Therefore, the weighted Voronoi region gives the most likely vector in MAP (maximum a posteriori) sense.

If one considers the set of messages that can be transmitted over the channel as the training vectors for a Palm network, and the received messages as the test vectors, we will show in the following sections that the binary Palm network approximately partitions the input space into Voronoi regions and the dynamically learned Palm model approximates a weighted Voronoi tessellation.

3.5 Conditioned Bayesian Classifier in Binary Palm Network

3.5.1 Output Error Characteristics

If an input vector is "closer" (has smaller Hamming distance) to one training vector than to the other training vectors, then the Palm network will always output a vector that is "closer" to that training vector than to other training vectors.

Proof: Suppose an input vector v can be regarded as the noisy version of training vectors v_1 or v_2 with error rates e_1 and e_2, where e_1 and e_2 are the error rates in the active nodes of v corresponding to v_1 and v_2 respectively. Let $e_1 < e_2$. Given the threshold θ and Equation (3.6), the number of active nodes k in the output vector consists of two parts: the number of genuine (correct) active nodes k_g and the number of spurious (error) active nodes k_s:

$$k = k_s + k_g = (n-k)\int_{\theta}^{k} P(d_l = x)dx + k\int_{\theta}^{k} P(d_h = x)dx \tag{3.10}$$

Via Equation (3.7), k_s can be approximated by:

$$k_s = \int_{\theta}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_l} \exp[-\frac{(x-m_l)^2}{2\pi\sigma_l^2}] = Q(\frac{\theta - m_l}{\sigma_l}) \tag{3.11}$$

Assume the number of active nodes in the output vector is equal to the number of active nodes in the training vector, then the number of spurious 0s equals the number of spurious 1s. For explanatory purposes, we only discuss the spurious 1s here, but the derivation is similar. The total error bits in a whole vector would be twice the number of spurious active nodes.

Corresponding to v_1 and v_2, the components of k are denoted by k_{s1}, k_{g1}, k_{s2}, and k_{g2}. Since the actual number of active nodes k in an output vector is determined by the network, k is fixed no matter whether the input is from v_1 or v_2. The network density is determined by training vectors, and we have $\rho_{l1} = \rho_{l2}$. Therefore, according to Equation (3.5), we get $\rho_{h1} > \rho_{h2}$. Given Equation (3.9), we get $m_{h1} > m_{h2}$ and $\sigma_{h1} < \sigma_{h2}$. According to (Buckingham and Willshaw 1993), if the minimum-error threshold is applied, then the relationship between the two thresholds is: $\theta_1 > \theta_2$. By Equation (3.11), we get $k_{s1} < k_{s2}$. So the output vector has fewer spurious active nodes corresponding to v_1 than to v_2, thus the output vector is closer to v_1 than v_2 in Hamming distance. ■

We see then that the Palm network effectively partitions the input vector space into Voronoi regions, with each centered at the training vector. A Voronoi region of a vector V_i is the union of all vectors V_x to which it is the closest one, $V_i = \left\{ V_x : \left\| V_x - V_i \right\| < \left\| V_x - V_j \right\|, i \neq j \right\}$, where $\|\cdot\|$ is a distance measure. A Voronoi tessellation can be created based on several different metrics, for the research discussed here, we use Euclidean distance for real valued vectors and Hamming distance for binary valued vectors. The Voronoi regions that binary Palm network creates are approximate in the sense that some training vectors may not appropriately represent the vector space such that there may be some spurious regions. If an input vector is in one Voronoi region, the retrieved output vector is within the same region as the input. Although the output vector may contain more noise than the input, it is still closer to the training vector in that region than to other training vectors.

The retrieved output vector is within the same Voronoi region as the input vector. However, not all the output vectors that are within a Voronoi region can be mapped to the original training vector, only those vectors that are within the attractor basin of the

training vector will converge to it. The attractor basins do not line up exactly with the Voronoi regions. Voronoi region distinguishes the nearest training vector, but can not ensure the convergence to it. Attractor basin is a region that ensures the convergence for any test vector within it. Some vectors that are beyond the attractor basin can still be mapped in the same Voronoi region, but since they can not converge to the original training vector, they can be mapped to a vector that has more noise, or, in other words, they can be mapped even further away the training vector than the test vector. This is very common when the minimum-error threshold can not distinguish exactly k active nodes in the retrieved output vector, then the output vector may have more noise than the test vector. Detailed analysis of the attractor basin is discussed in the next section.

Based on these considerations, we can see that, although the output vector is closer to the original training vector than to other training vectors, it may not necessarily be closer to the original training vector than the input vector, that is, it may contain enough noise to put it further away from the training vector, though a true Bayesian classifier would classify it incorrectly. The output error is an increasing function of input error, which will be discussed in the following section.

3.5.2 The Attractor Basin

Although the Palm network will return an output vector that is closer to one training vector than to other training vectors, it is not necessarily true that the output is convergent, that is, where successive operations of the network, feeding the network output back to the input results in output vectors that are not closer to the desired training vector. The Voronoi regions that a Palm network forms may not be the convergent region (an attractor basin with a single local minimum) where successive operations of the network (using output fed back to the input) result in vectors ever closer to the center of the region. For certain noisy input vectors, it is possible for the network to generate an output vector that has more noise than the input, that is, the output vector is even further away from the original training vector than the input vector.

For convergence to work, the vectors should be within the attractor basin of the training vector. The best match property of Palm is conditioned on this attractor basin

constraint. The attractor basin is defined as the maximum allowed input error rate e_{max} that can ensure a convergent network output. If an input vector is outside the basin, the retrieved vector can not be guaranteed to be the original vector. We next discuss how to find the attractor basins of a Palm network. In our formulation of this model, we assume that every test input vector is a noisy version of one of the training vectors. A condition for convergent performance is that the input vector be within the attractor basin of its source training vector.

The Hamming distance between the retrieved vector and the nearest training vector is the sum of the number of the false (spurious) active nodes and the number of the false silent nodes. The output error rate e_o then is:

$$e_o = (1-\alpha)\cdot P(d_l \geq \theta) + \alpha[1 - P(d_h \geq \theta)] \tag{3.12}$$

Given Equation (3.7), e_0 can also be approximated by:

$$e_o = (1-\alpha)\cdot Q(\frac{\theta - m_l}{\sigma_l}) + \alpha[1 - Q(\frac{\theta - m_h}{\sigma_h})] \tag{3.13}$$

where $Q(\cdot)$ is the error integral function of the standard Gaussian distribution.

According to Buckingham and Willshaw (Buckingham and Willshaw 1993), the k-WTA threshold is very close to the minimum-error threshold when e_i is small (less than 0.5). The threshold that gives the minimum expected error is the one that satisfies:

$$(1-\alpha)\cdot\binom{k}{\theta}\cdot\rho_l^{\theta}\cdot(1-\rho_l)^{k-\theta} = \alpha\cdot\binom{k}{\theta}\cdot\rho_h^{\theta}\cdot(1-\rho_h)^{k-\theta} \tag{3.14}$$

Since the *k*-WTA threshold is close to the minimum error threshold, we use the minimum error threshold to represent the *k*-WTA threshold. Solving Equation (3.14), we have the threshold:

$$\theta = \frac{\log\frac{k}{n-k} + k[\log(1-\rho_h) - \log(1-\rho_l)]}{\log[\rho_l(1-\rho_h)] - \log[\rho_h(1-\rho_l)]} \tag{3.15}$$

Consequently, once the network parameters are known, equations (3.13) and (3.15) can be used to compute the error rate e_o for the entire output vector. Since e_o is the error rate in the whole vector, while e_i is the error rate for the active nodes of an input vector, the normalized value of $e_o/(2\alpha)$ is used to make it comparable with e_i. To ensure convergent performance of the network, $e_o/(2\alpha)$ is generally required to be less than e_i, that is,

$$e_o / (2\alpha) < e_i \tag{3.16}$$

Accordingly, the attractor basin e_{max} for any particular Palm memory should satisfy the equal condition in equation (3.16), and can be computed via equations (3.13), (3.15) and (3.16). For any test vector within the attractor basin, iterative retrieval will result eventually in the original training vector that is the center of the basin.

To obtain an analytic expression for e_{max} by solving equations (3.13), (3.15) and (3.16) is complicated, but it can be illustrated graphically. The relationship between the input error rate e_i and output error rate e_o obtained from the equations is illustrated in Figure 3.3, where, $n=100K$ and $k=17$. As shown in Figure 3.3, the output error rate is an increasing function of the input error rate. The intersection of the dotted line and the colored curves specifies the maximum allowed input error rates (the values of the attractor basins) corresponding to different network densities ρ_l. If the output error rate lies below the dotted line in Figure 3.3, the corresponding input vector is in that vector's attractor basin, and the output error rate of a recurrent network approaches zero – which ensures convergent behavior.

As shown in Equation(3.15), the threshold θ is determined by the network parameters (n, k and M) and the input error rate e_i. Since the network density ρ_l varies with different combinations of network parameters (n, k and M), ρ_l can be regarded as a basic indicator of the network configuration. Therefore, the threshold depends on the density ρ_l and input error rate e_i.

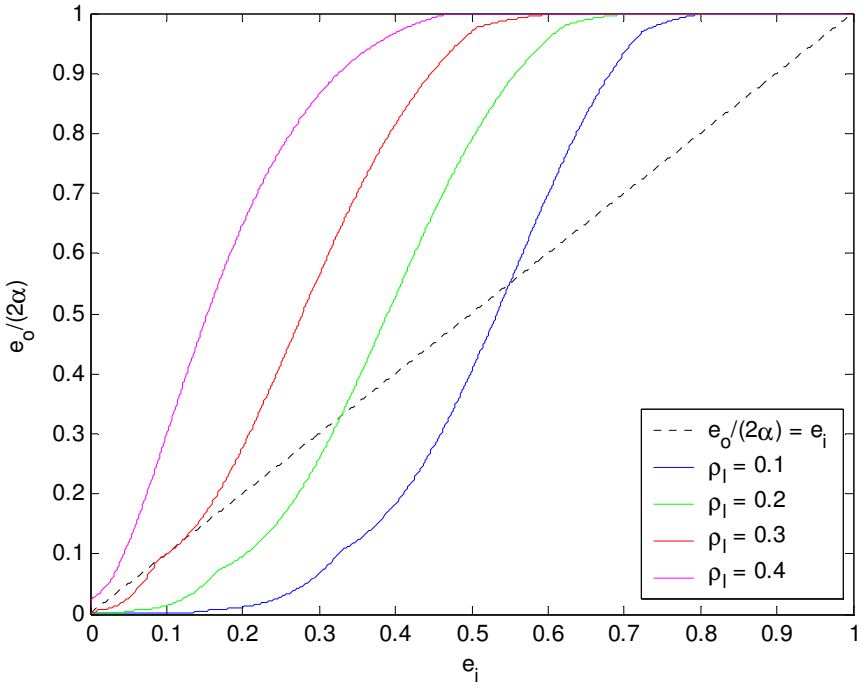

Figure 3.3 The relationship between the output error rate $e_o/(2\alpha)$ and input bit-flip rate e_i

3.5.3 Bayesian Interpretation

Summer and Dayan (Sommer and Dayan 1998) show the Bayesian retrieval characteristics of the Palm associative memory. However, that paper does not specify the conditions under which the network can actually retrieve the original training vector. In addition, asynchronous update of the state of the nodes in the network is assumed to ensure an appropriate Lyapunov function, which, in some applications of the Palm network, may not be satisfied. Although the Lyapunov approximation method explains the MAP inference from the posterior probability, the convergence issue is not specifically discussed there.

Here we provide a simpler interpretation based on the Voronoi model. As shown in the last section, under the convergence conditions, the Palm network will generate the original training vector that has the minimum Hamming distance to the input vector. In this way, the Palm network approximates a nearest neighbor classifier by dividing the space according to a Voronoi tessellation. Therefore, the Palm network can be regarded as a minimum Hamming distance classifier.

Returning to the communication channel paradigm, if one assumes a simple binary symmetric channel with independent, identically distributed errors, and all input vectors are of fixed length and with equal prior probabilities, then the minimum Hamming distance decoder is the maximum-likelihood decoder (Haykin 2001). Therefore, the training vector with the minimum Hamming distance from the input vector is the most-likely vector to have been transmitted over the channel in a Bayesian sense. Since the Palm network approximates a minimum Hamming distance classifier, the Palm associative memory will approximate a Bayesian decoder of the vectors received over the channel. Under the condition that all the output vectors can converge to the original training vectors, the Palm network works as a Bayesian classifier.

3.5.4 Simulation Results

Once the network parameters are given, attractor basin size $e_{\max}$ can be easily computed numerically by solving equations (3.13), (3.15) and (3.16). In simulations, the maximum input error rate that ensures the average output error rate equal to the input error rate is used as the attractor basin size. Figure 3.4 shows one example of $e_{\max}$ derived by both the theoretical computation and the simulation. In this example, vector size $n = 1000$, number of active nodes $k = 10$, and the number of test vectors is 100. As shown in Figure 3.4, the attractor basin shrinks as the number of vectors stored increases. The curves are not smooth, this is most likely due to the fact that the basin is not sensitive to minor changes on the number of vectors being stored.

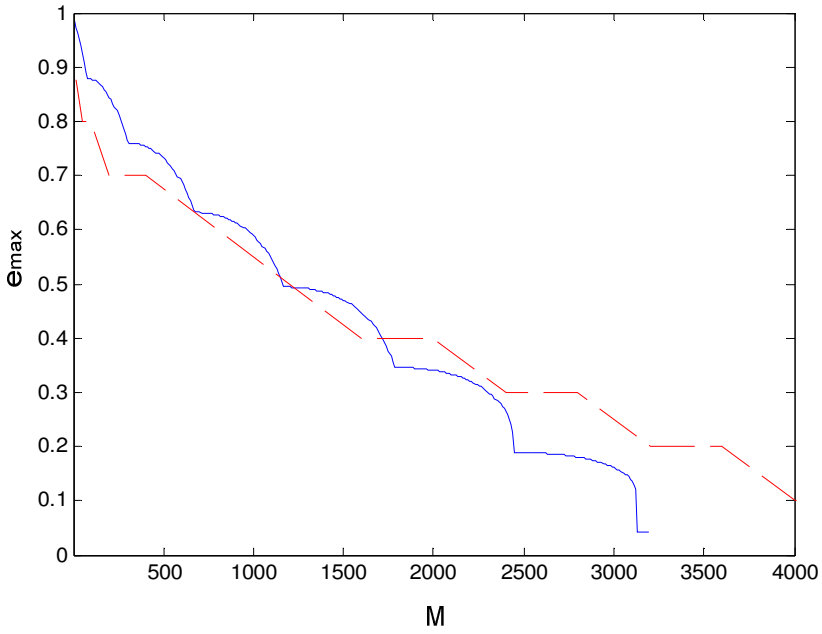

Figure 3.4 Attractor basin size e_{max} vs. number of stored training vectors M.
In this network, n=1000, k=10. The blue curve is the theoretical values for attractor basin size, and the red curve is obtained by simulation.

In the binary Palm model, we have shown that the network performance is determined by the density ρ_l , the input error rate e_i, and the number of active nodes k. The size of the attractor basin is the maximum input error rate that ensures convergent performance, so the sizes of the attractor basins depend on the density ρ_l and the number of active nodes k. The attractor basins shrink as the network density increases, which is an expected result as is shown in Figure 3.5 and Figure 3.6.

Figure 3.5 demonstrates the relationship between the size of the attractor basin and the network parameters: the number of vectors stored, M, and the number of active nodes, k. In Figure 3.5, the network size n = 1000. As can see, the attractor basin shrinks as network density increases. When the network size is unchanged, the network density is determined by the number of active nodes in each vector, and the total number of vectors being stored.

As can be seen in Figure 3.5, the curves fluctuate. This variation is due to the quantization effect of the k-WTA operation on the output vector. The depth and width of each fluctuation vary with the number of active nodes, k. The smaller the number of

active nodes, the more sensitive the attractor basins are to the number of vectors stored, and the deeper the fluctuation. Larger *k* results in smaller attractor basins.

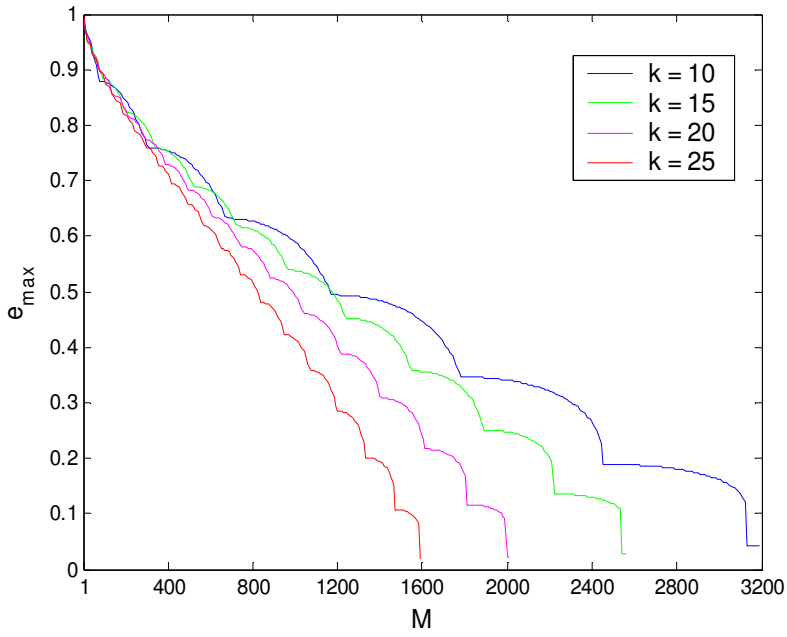

Figure 3.5 The attractor basin size e_{max} varies with the number of vectors stored *M* and the number of active nodes k, with network size *n*=1000.

However, if *k* is too large, even when only a small number of vectors are stored, the attractor basins still end up being very small and the interference among the training vectors becomes significant. With large *k*, the vectors get closer to each other so that the Voronoi regions shrink and the size of each attractor basin approaches zero.

Both the number of vectors stored, *M,* and the number of active nodes, *k,* determine the network density, ρ_l, which, as we saw above, determines the network performance. Various combinations of *M* and *k* can have the same ρ_l, so it is useful to see how the single parameter ρ_l influences the attractor basin e_{max}.

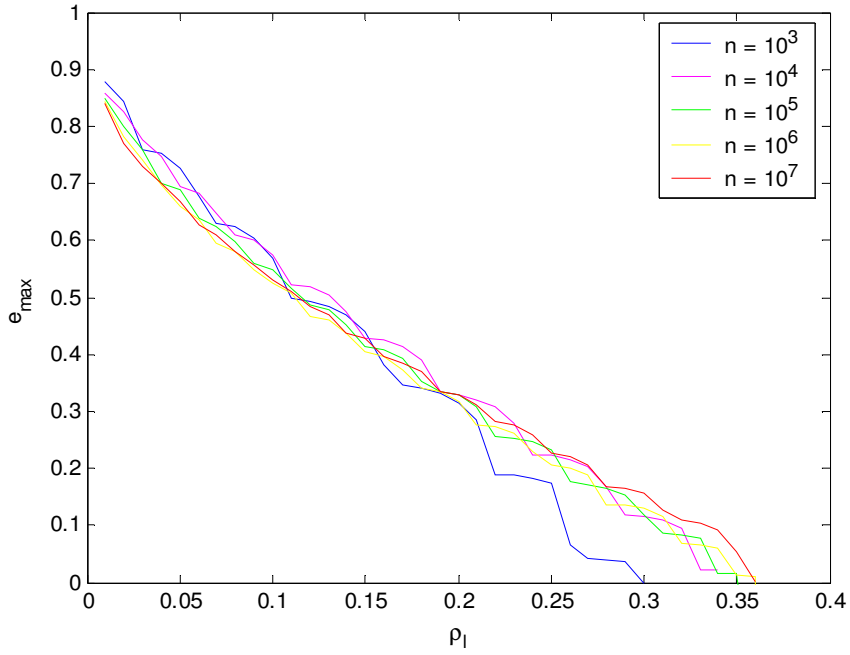

Figure 3.6 The attractor basin e_{max} varies with the network density ρ_l for different network size n.

Figure 3.6 illustrates the relationship between the attractor basin e_{max} and the network density ρ_l. It is interesting to note that the attractor basin does not vary much with the network size n. In the simulations, k is a number that is greater or equal to $log\ n$. When $k = \log n$, we get the maximum values of e_{max}. As shown in Figure 3.6, the achievable maximum ρ_l that can ensure a convergent performance is 0.35, which is consistent with the theoretical asymptotic storage capacity of ln2/2 for an auto-associative network (Palm, Schwenker et al. 1997).

Figure 3.7 illustrates the characteristics of the Voronoi regions. In the simulations, the network size n=1000, the number of vectors stored M=100, each with k=10 active nodes. With such network settings, the network density is small, and the attractor basin can reach to the Voronoi boundary. To see this, we choose a test vector that is formed by the components of two training vectors (V1 and V2). We randomly swap the active nodes of two vectors, and still keep the number of active nodes the same in the test vector. With k=10, the Hamming distance between the two training vectors is 20. By swapping the active nodes of the two training vectors, the test vectors lie on a line between the two training vectors, with the sum of each test vector's Hamming distance to V1 and the

Hamming distance to V2 equals 20. Interestingly, any test vectors that are closer (have less Hamming distance) to V1 will converge to V1, and any test vectors that are closer to V2 will converge to V2. When the test vector is equal distance to V1 and V2, the network will not converge to either.

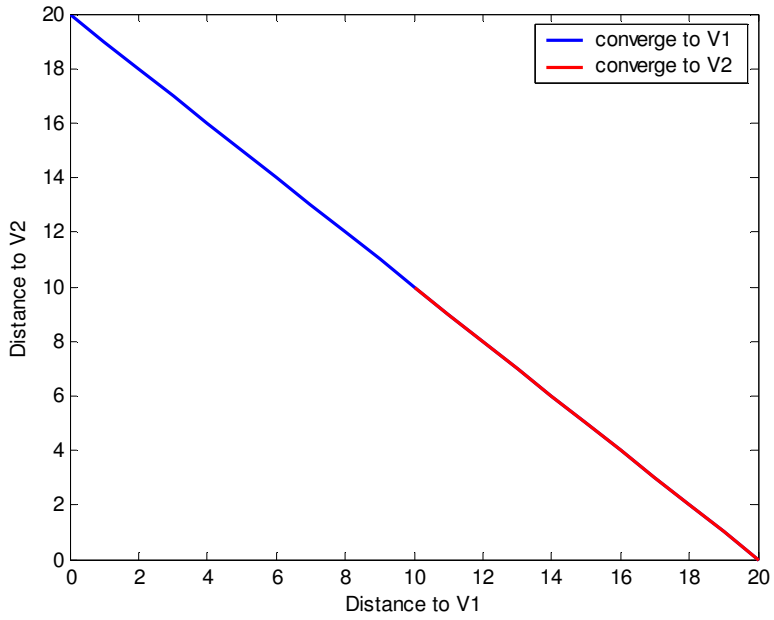

Figure 3.7 Illustration of convergence between two vectors

In binary learning, if all the training vectors are equally likely, the Voronoi boundaries are equal distant to each vector. However, the attractor basin of each vector may not necessarily cover the Voronoi region defined by that vector. Therefore, when an input vector does not belong to any attractor basin, the retrieved output vector might be an arbitrary vector, and convergent retrieval is not guaranteed.

3.5.5 Fault Tolerance Experiments

In Chapter 2, we showed the capacity of Palm model for large networks, and we simulated the retrieval performance for noisy inputs. The simulations demonstrated that the network is able to remove noise from the input vector, which partially demonstrates our Bayesian Classifier theory. In this section, we describe several simulations that tested

the fault tolerance of Palm network, which demonstrates the Voronoi characteristics regarding network fault tolerance.

We test three different fault models in the weight matrix: Static, Dynamic, and a hybrid of Static and Dynamic. For the Static Fault the weight matrix is modified once and then used for all the test vectors. For the Dynamic Fault, each time the matrix is used, a different set of faults are added to the original matrix, but the faults are not accumulated over time. For each configuration of fault, three different methods can be used to modify the weight matrix: 1) randomly add/delete connections; 2) multiply the connections by a random fractional number; and 3) a hybrid combination of the two methods.

We do several experiments on the combinations of different configurations and methods. All simulations are based on the original batch-training model. The parameters used here are: vector_size n = 1000, number of active nodes k = 10, number of training vectors M = 250, number of testing vectors = 250, error rate (bit-flip probability) in the test vector = 10%. The network performance is still defined as:

Performance = 1- D(trn, out) / D(trn,tst),

where D(trn, out) is the average Euclidean distance between training vectors and output vectors, and D(trn,tst) is the average Euclidean distance between training vectors and test vectors.

Experiment 1: Randomly add and delete connections in the binary weight matrix by the probability of P_add_delete. In this configuration, some binary connections are disabled and some connections are reestablished, and the total percentage of disabled or reestablished connection is P_add_delete. The performance of this experiment is shown in Figure 3.8. As shown in the figure, the binary Palm model is tolerant of a modest degree of connection failure and random reconnection, and the network can still achieve a fairly good performance. This random addition and deletion of connections is different from the patchy connectivity (Johansson, Rehn et al. 2006) network where a random patchy (block) masking matrix is applied to the original binary weight matrix. In the patchy connectivity network, all connections that are not within the random block are disabled, but no new random connections are reestablished. While in our experiment here, both random addition and deletion of connections are applied, and the addition and deletion are at neuron level instead of block of neurons level. The effect of Static and the

Dynamic faults to the retrieval performance are reasonably comparable in this configuration.

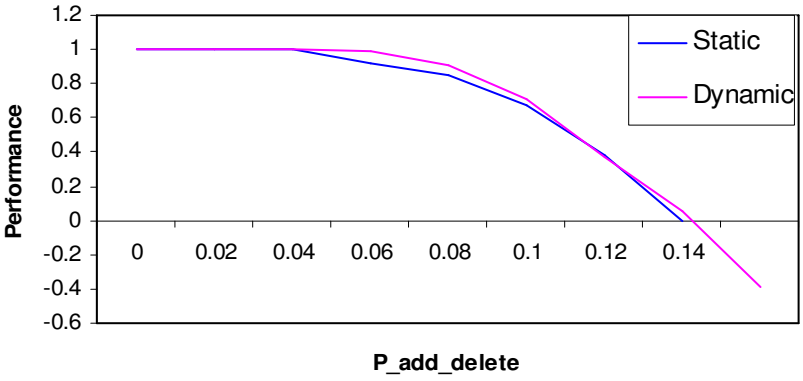

Figure 3.8 Network performance for random addition and deletion of connections in weight matrix

Experiment 2: Some weights are multiplied by random fractional values. In this configuration, all the 1s in the Palm network are multiplied by a random value. No addition or deletion of connections is considered here. The performance of this experiment is shown in Figure 3.9, where the performance in both the Static and the Dynamic configurations are close to the original Palm, which has a Performance of 1. There are not much difference in the performance for the Static and the Dynamic faults.

Figure 3.9 Network performance for random fractional values in weight matrix

Experiment 3: Combine the configurations for Experiment 1 and 2 together, and the performance is shown in Figure 3.10. In this configuration, the dynamically changed weight matrix results in better performance than the statically changed weight matrix in the network.

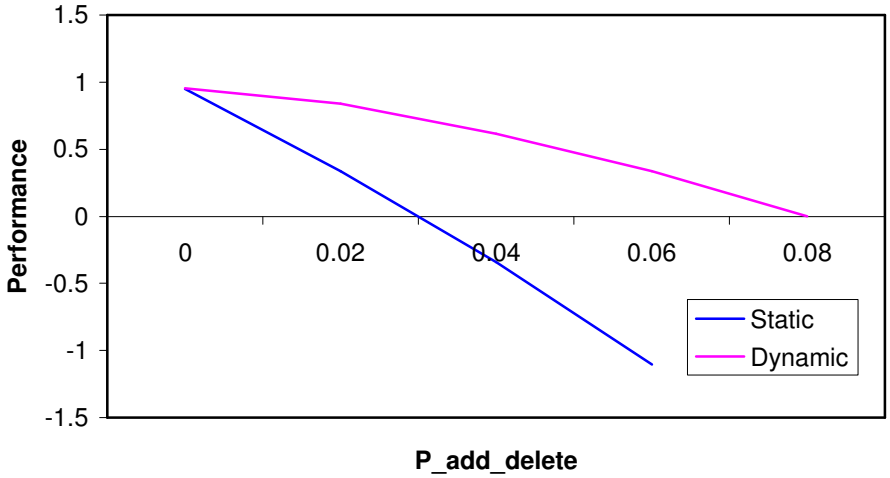

Figure 3.10 Network performance for random addition and deletion of connections, and each connection in the weight matrix is multiplied by a fractional value.

For better comparison, the performance of the network with Static faults via three different connection modifications is compared with that of the original Palm network, as shown in Figure 3.11. The performance of the network with Dynamic faults via three different connection modifications is also compared with that of the original Palm network, as shown in Figure 3.12. For both Static and Dynamic faults, multiplying each connection by a random factional value will not lose much performance compared to the original Palm model. Randomly adding and deleting some connections in the weight matrix causes the performance to deteriorate, especially when the probability of addition and deletion is relatively large. A combination of the two methods produces the worst performance even with a very small probability of addition and deletion. In cortex, for example, gradual changes in synaptic connection are common, since the chemical concentration around a synapse may not always be constant, but random addition or deletion of a synaptic connection is probably much less common.

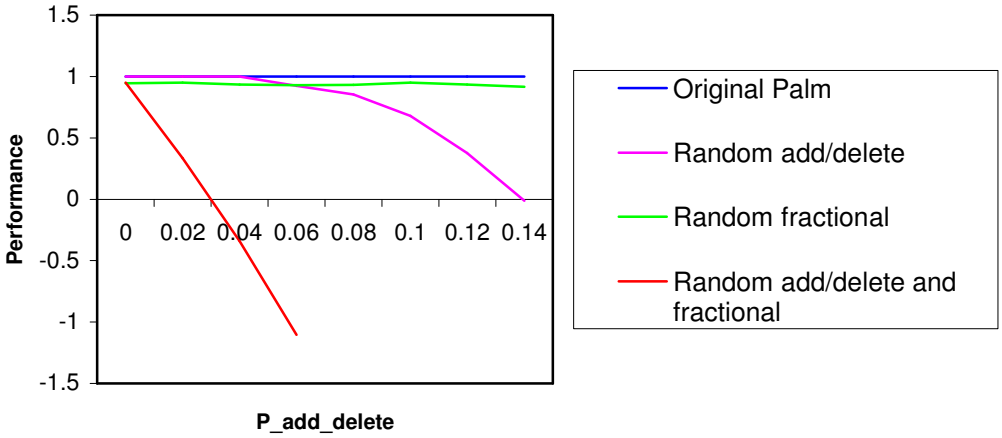

Figure 3.11 Performance comparison of a fault free Palm and the same network with Static faults. Three different kinds of Static faults are generated: random addition and deletion of connections, connection multiplication by random fractional values.

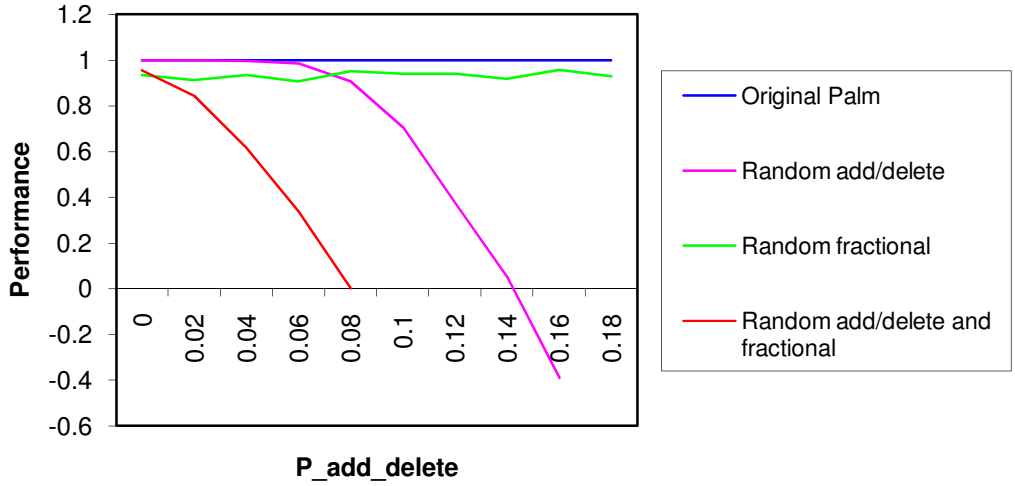

Figure 3.12 Performance comparison of a fault free Palm and the same network with Dynamic faults. Three different kinds of Dynamic faults are generated: random addition and deletion of connections, connection multiplication by random fractional values.

Our experiments show that the Palm network is fault tolerant to several different types of faults. For relatively small faults in the weight matrix, the network continues to perform Bayesian classification, by retrieving the best training vector that is closest to the given input.

3.6 Dynamic Learning in Palm Network

In the binary Palm model, training and retrieval are separate processes. If too many training vectors are stored, or if they are too close to one another, the attractor basin for each training vector becomes so small that correct retrieval becomes difficult. To overcome the problems, a dynamic learning model is proposed.

3.6.1 MAP Retrieval in Dynamic Learning

The most common usage of association memory, as we have assumed in the previous chapters, is that a set of coefficients, the weight matrix, is generated once from a set of training vectors. In addition the training vectors are assumed to be equally likely, that is, their prior probabilities are all equal. In many real world applications this is generally not the case, but, nevertheless, the use of such a "naïve" prior is common. Likewise, in situations where the associative network is used as a module in a larger self-organizing system, it may be impossible to provide the "training" vectors ahead of time.

In addition to being able to handle non-uniform priors, in some cases we would like the associative memory to be able to learn incrementally during operation, since it may not always be possible to obtain all the relevant vectors in advance. Also if the fundamental probabilities are non-stationary, though changing slowly, dynamic learning allows the memory to adapt to these changes.

Dynamic learning implies that new vectors should start with small attractor basins that increase gradually over time. Likewise, old vectors may fall out of disuse and their attractor basins should decrease, possibly eventually disappear. In the binary Palm network, if too many training vectors are stored, they can be too close to each other such that the attractor basin for each training vector becomes too constrained for correct retrieval. Dynamic learning also allows us to deal with this situation more flexibly. Dynamic learning implies that the retrieval procedure can take place before learning is complete. Learning and operation are concurrent. The weight matrix elements are adjusted incrementally. In this mode, the Voronoi regions are affected by the learned mappings, and boundaries are reshaped via the learning of new training vectors.

In the situation where two training vectors are close to each other but have different prior probabilities, dynamic learning would allow the vector with larger prior to have a larger Voronoi region, absorbing part of the attractor basin of a neighboring vector with a smaller prior. Therefore, the prior probability of each training vector affects the size of the Voronoi region in that a weighted Voronoi region is formed, which will be discussed in the next section.

In the dynamic learning model of auto-associative memory, each training vector is associated with an *influence coefficient c* that directly depends on the prior probability, *P(y)*, of that vector. Generally the influence coefficient *c* should be a monotonic function of *P(x)* to approximate the prior probability of *x*. The amount of space each vector takes is relative to its neighbors, since the total Voronoi space is fixed, and training vector tessellations divide that space into a number of regions. This is reasonable since the prior probabilities divide up the real line between 0 and 1.

The Voronoi tessellation implemented in an associative memory is similar to Vector Quantization (VQ) (Gersho and Gray 1992). In VQ, there are a fixed number of codebook vectors that are moved around to best represent a given set of input vectors. The lengths of the codebook vectors can change, and a dynamic learning algorithm is needed to find the set of codebook vectors avoiding one vector occupying all the space. In an associative memory, all the training vectors have the same length, each associated with different coefficient. As the training proceeds, the size of Voronoi region where each training vector occupies also changes. Once the training is done, weighted Voronoi regions are formed. Associative memory is a reasonably efficient approximation to VQ by mapping the input space into Voronoi tessellation.

During training, the weight matrix *W* can be formed such that $W(t+1) = F[W(t) + x \cdot x^T]$, where $F(\cdot)$ is normally a non-linear "compressing" function such as the sigmoid function. For the simplicity of analysis, assume $F(\cdot)$ is a linear function, then the weight adaptation for each training vector x^m is approximated by $\Delta w_{ij}^m \propto c^m \cdot x_i^m \cdot x_j^m$, so the weight computation is

$$w_{ij} \propto \sum_m \Delta w_{ij}^m \tag{3.17}$$

The non-linear function, $F(\cdot)$, is generally monotonic, it is still reasonable to approximate the weight computation with a linear combination of weight adaptation for each training vector. Our objective here is to show that the Palm network can perform incremental learning and that unequal vector weights create a weighted Voronoi region. Though not shown here, it is possible to have far more complex weight computations. One example is to use time-varying averages similar to a Kalman filter as is used in the BCPNN algorithm (Sandberg, Lansner et al. 2000).

When a test vector $\tilde{x}$, which is a noisy version of one of the training vectors, is presented to the network, an ideal associative memory should return a training vector x that has the maximum posterior probability $P(x|W,\tilde{x})$. Under the condition that the network W is independent of $\tilde{x}$ given x, the posterior probability $P(x|W,\tilde{x})$ can be written as:

$$P(x|W,\tilde{x}) = \frac{P(x,W,\tilde{x})}{P(W,\tilde{x})} = \frac{P(W|x)P(\tilde{x}|x)P(x)}{P(W,\tilde{x})} \tag{3.18}$$

Taking the logarithm on both sides, we get

$$\log P(x|W,\tilde{x}) \propto \log P(W|x) + \log P(\tilde{x}|x) + \log P(x) \tag{3.19}$$

During retrieval, we want the vector x that maximizes $\log P(x|W,\tilde{x})$. We can enforce the neuronal activity update in the gradient ascending direction of $\log P(x|W,\tilde{x})$,

$$\frac{d}{dx}\log P(x|W,\tilde{x}) = \frac{d}{dx}\log P(W|x) + \frac{d}{dx}\log P(\tilde{x}|x) + \frac{d}{dx}\log P(x) \tag{3.20}$$

Letting $P(x^m) \propto c^m$, and assuming that $P(\tilde{x}|x)$ depends only on the input error rate, that is, $P(\tilde{x}|x^m) \propto e_i$ for all the m, then $\frac{d}{dx}\log P(W|x)$ becomes the central quantity in Equation (3.20). If we further assume that the elements in the weight matrix W are conditionally independent of each other, then $P(W|x)$ can be written as:

$$P(W|x) = \prod_{i,j} P(w_{ij}|x_i, x_j) \tag{3.21}$$

Using equation (3.17) and the central limit theorem, $P(w_{ij})$ can be approximated by a Gaussian distribution:

$$P(w_{ij}) \simeq G(w_{ij}; \mu_w, \sigma_w) \tag{3.22}$$

with $\mu_w \simeq \alpha^2$ and $\sigma_w \simeq \alpha^2(1-\alpha^2)$, where α is the percentage of the active nodes in a training vector. Therefore, the probability of *W,* given *x* is:

$$P(w_{ij}\,|\,x_i, x_j) \simeq G(w_{ij}; \mu_w + cx_i x_j, \sigma_w) \tag{3.23}$$

Taking the derivative of both sides of equation (3.21), we have:

$$\frac{\partial}{\partial x_i}\log P(W\,|\,x) = \sum_j \frac{\partial}{\partial x_i}\log P(w_{ij}\,|\,x_i, x_j) \tag{3.24}$$

From equation (3.23), we get:

$$\frac{\partial}{\partial x_i}\log P(w_{ij}\,|\,x_i, x_j) = \frac{2}{\sigma_w^2}[c\cdot(w_{ij}-\mu_w)x_j - c\cdot x_i \cdot x_j] \tag{3.25}$$

As derived by Lengyel and Dayan (Lengyel and Dayan 2004), the optimum neuronal activity updating function $A(x_i, x_j)$ is:

$$A(x_i, x_j) = c\cdot(w_{ij} - \mu_w)\cdot x_j \tag{3.26}$$

$A(x_i, x_j)$ is approximately optimal when retrieving the information stored in the weight matrix *W*. Then the central quantity of the gradient ascent in equation (3.20) becomes:

$$\frac{\partial}{\partial x_i}\log P(W\,|\,x) = \frac{2}{\sigma_w^2}[\sum_j A(x_i, x_j) - \varepsilon] \tag{3.27}$$

where ε is a constant.

Equations (3.26) and (3.27) show that the weighted sum of inputs leads to gradient ascent of the posterior probability $P(x|W,\tilde{x})$, and that the retrieval update defined by $A(x_i, x_j)$ is a procedure for finding the logarithm of the maximum posterior probability, $\log P(x|W,\tilde{x})$. Therefore, the output vector retrieved by a dynamically learned associative memory approximates the most likely training vector in MAP sense.

3.6.2 Weighted Voronoi Regions in Dynamic Learning

Since the weight matrix elements are adjusted incrementally, dynamic learning implies that the retrieval procedure can take place during learning. In this mode of operation, the Voronoi regions are affected by the learned mappings, with the boundaries being reshaped by the learning of new training vectors.

The retrieval procedure returns a vector that has approximately the maximum posterior probability $P(x|W,\tilde{x})$. In equation (3.18), where *P*(*x*) is proportional to *c*, $P(x) \propto c$.

For $P(\tilde{x}|x)$, the smaller the input error rate e_i, the more likely that $\tilde{x}$ is generated by x. $P(W|x)$ is a monotonic increasing function of the coefficient c. So the posterior probability $P(x|W,\tilde{x})$ is influenced by its coefficient c. We now show that dynamic learning will generate weighted Voronoi regions in the input space.

In equation (3.18), $P(\tilde{x}|x)$ is a measure of the distance $d(\tilde{x},x)$ between x and $\tilde{x}$. The smaller the input error rate e_i, that is, the smaller the distance $d(\tilde{x},x)$, then the larger the likelihood $P(\tilde{x}|x)$. Suppose two training vectors x_1 and x_2 have different coefficients c_1 and c_2 respectively, with $c_1 > c_2$. Let $d(\tilde{x},x_1) = d(\tilde{x},x_2)$, thus $P(\tilde{x}|x_1) = P(\tilde{x}|x_2)$, that is, $\tilde{x}$ is equally likely to have been sent as x_1 or x_2. $d(\tilde{x},x)/[P(W|x)P(x)]$ can be regarded as the weighted distance of $\tilde{x}$ from x. Since $c_1 > c_2$, we then have:

$$d(\tilde{x},x_1)/[P(W|x_1)P(x_1)] \le d(\tilde{x},x_2)/[P(W|x_2)P(x_2)],$$

or equivalently:

$$P(W|x_1)P(\tilde{x}|x_1)P(x_1) \ge P(W|x_2)P(\tilde{x}|x_2)P(x_2).$$

Therefore, the weighted distance of $\tilde{x}$ from x_1 is smaller than that from x_2, so $\tilde{x}$ is within x_1's weighted Voronoi region. By equation (3.18), we can infer $P(x_1|W,\tilde{x}) \ge P(x_2|W,\tilde{x})$. Consequently, the network will output the training vector that has the minimum weighted distance. As a result, a dynamic-learning Palm network approximately partitions the input space into weighted Voronoi regions, with each influenced by the corresponding influence coefficient. Since the coefficient can varies along learning, the value of the coefficient for each training vector can be a complicated function of time. The convergence characteristics and the attractor basin of the weighted Voronoi regions are not discussed here, which could be a direction for future research. The influence coefficient is a value that represents how dominant a training vector is in the network, so that the vector that has a relatively high influence coefficient has better error tolerance and is less likely to be forgotten; while another vector having a low influence coefficient will be more easily forgotten.

3.6.3 The Retrieval Performance in Dynamic Network

In dynamic learning, when a vector is presented to the network more frequently, its influence coefficient, *c,* will increase, thus its corresponding Voronoi region will grow, which is what one would expected from a Bayesian perspective. The retrieval performance is thus affected by *c*, as is the relative size of the Voronoi regions. If infinite learning time and stationary priors are assumed, the influence coefficients are also stationary, and the sizes of the Voronoi regions are uniquely determined by the influence coefficients, with larger *c* corresponding to larger Voronoi regions and relatively better error resistance.

Figure 3.13 illustrates the relationship between the retrieval performance and the coefficient *c*. The retrieval performance is defined as the ratio of the number of accurately retrieved test vectors to the total number of test vectors. Due to the non-linearity of the network, the retrieval performance is non-linear with respect to the influence coefficient.

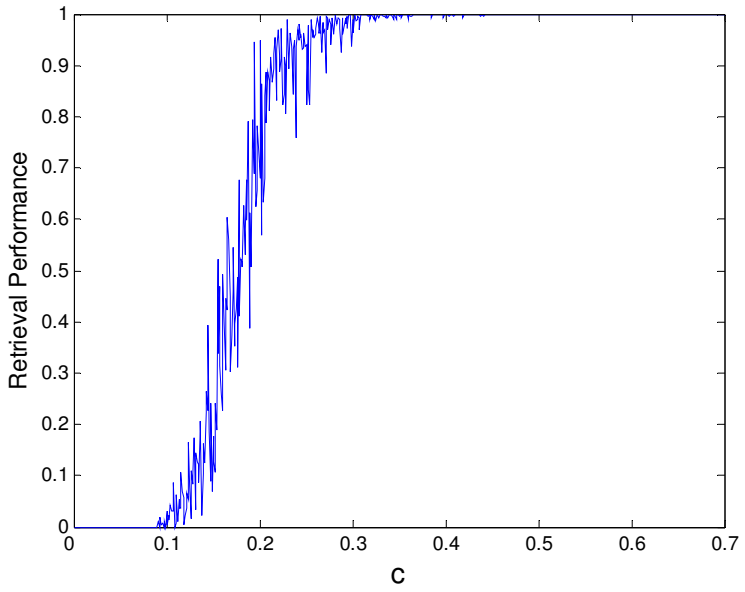

Figure 3.13 The relationship between the retrieval performance and the influence coefficient of each training vector. In this network, n=1000, k=10, M=700, and e_i=0.3. For each training vector, 300 testing vectors are used, and all the 300 test vectors have the same c (c=i/1000, i is the index of the training vector).

A larger influence coefficient corresponds to better retrieval performance. A vector that has a large influence coefficient is less likely to be forgotten than low

coefficient vectors. It is interesting to look at forgetting and unlearning. The less a stored vector is used, the more likely it will be forgotten. On the other hand, a stored vector that is used frequently will be reinforced. To unlearn a frequently used habit thus requires significant effort to reverse or disable the structural changes. The dynamic learning model suggests a common idea: repetition is essential for most types of learning.

In cortex, short-term memory may involve an increase in the efficiency of synapses for only a short period of time. If the influence coefficient can be modeled by some neurotransmitter modules to affect the synaptic efficacies, the dynamic learning associative network can be modeled as a short-term memory. Short-term memory can be transferred to long-term memory by repeating the information. For example, sleep is considered a necessary factor for establishing well-organized long-term memories. Probably during sleep, the important knowledge and events are rehearsed (Hecht-Nielsen 2003b), therefore the high-influence coefficient, corresponding to the importance of an event, may also be modeled to simulate long-term memory.

There are many types of memory in the human brain, and different neural activities associate with different functional phenomena. Dynamic learning associative memory is a simple model, and this single model by itself cannot simulate all the various kinds of memory. Still, the general learning characteristics of the model provides helpful material in modeling a more complex system and makes it more useful in a wider ranged of application.

In the next chapter, we implemented reinforcement learning in a Palm network (Zhu and Hammerstrom 2003), where the state-action mappings are learned incrementally. Reinforcement was a demonstration of the use of dynamic learning in a Palm network. Our results show that when a particular state mapping is fed to the network more frequently, its effect on the input space increases. However, an even more important result is that the complex mappings of fairly arbitrary state-action pairs can be learned via dynamic learning defined over a simple associative memory model, and that the weighted Vornoi regions are formed accordingly.

In conclusion, we've discussed the functional analogy of a simple associative memory network to Voronoi classification, and subsequently to Bayesian inference. It was shown that when the input vector is within the attractor basin of a training vector, the

associative network can return the most likely training vector in a Bayesian sense. Furthermore we introduced the concept of weighted Voronoi regions and then defined a dynamic learning version of the network that approximates weighted Voronoi classification. In addition, this dynamic learning capability approximates the influence coefficient of each training vector and allows the representation of prior probabilities in the Bayesian inference process.

The Bayesian inference approximation provides a straightforward and useful theoretical foundation for the operation of simple associative memory, such as the Palm model, and provides an analytical basis for the application of Palm network to real world problems. Based on these results, a Palm memory was used in an Enhanced Vision System (EVS) (Luk, Gao et al. 2004) used for pilot landing guidance. In this system the Palm model retrieved the original runway image from a database using small amounts of input data from a range of visual sensors, where association is done in feature vector space. The Palm model is an efficient way to do "most likely" Bayesian inference over our vector space.

We believe that distributed representations used by neural-like models provide a way to significantly reduce, or more precisely spread, the inference calculation over a large number of simple processors. And they may be an important component of cortical computation.

However, associative memory models such as Palm's have significant limitations as potential models of cerebral cortex, one of the most serious is the model's high connectivity. Consequently more complex structures are required. Research is already proceeding in this direction (Hecht-Nielsen 2003b; Johansson 2004; George and Hawkins 2005). One promising approach is to construct hierarchically connected associative networks that distribute the inference calculation to maximize inference efficiency, and capture higher order information from the application space. With the foundation presented developed in this chapter, the Chapter 6 will discuss the hierarchically connected network that is capable of utilizing high-order feedback from the above network and scaling to a large network size.

Chapter 4

Reinforcement Learning in Associative Memory

4.1 Introduction

So far we have shown a number of useful characteristics of a simple associative model. It is robust when the network is not overloaded, and it scales reasonably well. In dynamic learning, each training vector has different influence on the retrieval performance and generalization ability. For a biologically inspired system, incremental learning via interaction with a larger environment is very important. Reinforcement Learning (RL) is a type of learning that fundamentally requires the ability to learn incrementally during network operation. It has been used by many systems that learn via direct interaction with their external environment. One reason for its wide-spread use, as a useful model of biological learning, is that it operates via trial-and-error, there is no supervisory feedback provided to the system, only a single success or fail reward signal is available. In many real environments the only feedback available is binary, pass/fail, good/bad, success/failure, etc.

In this chapter, we show a Palm model applied to RL, demonstrating that a simple associative network is capable of dynamically learning complex, non-linear functions with good retrieval performance and generalization. Then using these results we discuss its potential as a model of specific areas of the brain such as the lateral intra-parietal (LIP) area.

RL is often modeled with a separate "actor" and "critic." The critic learns to predict the value of certain actions under certain conditions (system and world state), and the actor learns to perform certain actions based on the predicted values. It has been

recently proposed by neuroeconomics (Glimcher 2003), that the decisions animals make are also based on some kind of "critic" map formed in our nervous system that connects the sensation and the action. In this chapter, we investigate the "critic" and "actor" relationship in a dynamically learning associative memory. We propose a reinforcement learning based Associative Memory (RLAM) system, give its application in a simple grid-world simulation, and discuss the computational power of this model.

4.2 Mappings in Brain

Neural anatomy shows that sensory information is almost always topographically represented in the brain. For example, visual information perceived by the retina is mapped topographically (retinotopically) to primary visual cortex, as is somatosensory input. It is also believed that motor control results from a topographic-mapping of the motor areas in cortex to the muscles that control the movement (Glimcher, Dorris et al. 2005). The connection between sensation and action is formed through a decision-making procedure where topographic parallel sensory information is transformed to the corresponding movement information.

Recently, researchers have begun to model certain decision processes by using theories from economic game theory. This area, called neuroeconomics (Glimcher 2003), leverages research on neuroscience, economics and psychology, and studies how neural circuits make decisions, categorizing risks and rewards, and the interaction with other animals.

A map-like topographic structure in the brain transforms parallel sensory information into a relative magnitude map of expected reward, which, in turn, is mapped to the corresponding action. These various mappings are performed by associative memory like processing, consequently, associative memory appears to be one of the most important functions in many cognitive processes, and in various forms it appears to be a fundamental computational capability for a wide variety of neural computation. In a visual saccade, for example, the sensory information obtained by the eye is transformed into movement information, which controls the eyeball to perform the saccade. There are many complex circuitries involved in this operation, some of them can be approximated

by associative mappings. These mappings are learned incrementally by previous experiences, and it is believed that LIP participates in the decision making process. It is shown (Glimcher 2003) that the neurons in LIP carry information about the probability that a movement would yield a reward, and they also encode the value of each movement. In the sensorimotor process or the decision making process, the nervous system is trying to maximize the reward or fitness of the organism.

The posterior parietal cortex is believed to participate in sensorimotor signal generation (Goldberg, Colby et al. 1990; Buneo, Jarvis et al. 2002). In reaching for an object, or doing a saccade to a visual target, the brain must transform the sensory coordinates of the stimulus into motor coordinates. Early motion planning appears to be made in the posterior parietal cortex. Based on different assumptions on how the sensorimotor transformation is performed, people build different models to simulate the functions in the parietal cortex. The model by Zipser and Andersen (Zipser and Andersen 1988; Braitenberg and Schuz 1998) and the later studies by their group (Goodman and Andersen 1990; Mazzoni, Andersen et al. 1991; Andersen 1995) decompose the sensorimoter transformation into a series of sub-transformations, where the visual signal is mapped into several intermediate references, such as head-centered and body-centered coordinates. Pouget and Sejnowski (Pouget and Sejnowski 1997) propose another sensorimotor transformation model in which the sensory inputs are encoded in a representation suitable for generating motor commands. The representation is created from a set of basis functions that can be used to generate receptive fields in either retinotopic or head-centered coordinates via linear transformations. Their simulations show that the spatial representation model of parietal cortex can explain the hemi-neglect syndrome that the patient fails to respond to stimuli presented to the side opposite to a brain lesion.

The Basal Ganglia are sets of cortical nuclei that are involved in movement decisions and control. Cell death in the basal ganglia causes a variety movement disorders, such as Parkinson's disease and Huntington's disease. The Basal Ganglia play important roles in the contextual analysis of the environment and the use of that information for movement (Houk 1995). Some of its functions include: sensory-motor associative learning, reinforcement learning, procedural learning, planning, etc. The basal

ganglia play a major role in selecting different sensory-motor pathways according to different contexts. A modular neural-network model by Baldassarre (Baldassarre 2002) is proposed to simulate the role that the basal ganglia play in learning and motor selection with different goals. The methods that this model uses are reinforcement learning based actor-critic methods, and are similar to the methods that our RLAM system uses, except that there are multiple goals in their system.

4.3 Reinforcement Learning (RL)

RL is an online learning rule that interacts dynamically with an environment. Many biological neural systems perform some kind of reinforcement like learning. There are two components in an RL system: the agent and the environment. The learning takes place via a series of steps. At each step, the agent performs an action, if the action is successful, the agent gets a reward, otherwise no reward is given. The agent's goal is to maximize reward. After each step, the agent updates its selection policy according to the reward, and so incrementally improves task performance. After sufficient exploration of the state space, the agent generally finds a policy for that task that returns a maximum reward.

Q learning is one of the most important methods in RL. In Q learning, there are two basic elements: the actor and the critic. The *critic* estimates the value of the current action-state pair, and the *actor* selects the action based on input from the critic. The system is generally provided with a reward signal that is normally binary. From this signal the critic learns to estimate the value of action-state pairs. The goal of Q-learning is to develop a policy that generates certain actions in certain states that maximizes the reward. In Q-learning, $Q(s,a)$, the value for the state s and action a pair, represents the agent's current knowledge about expected long term reward for taking action a in state S. For each state, there may be several Q-values corresponding to each action, the agent will generally take the action that has the maximum Q-value ("exploitation"). The goal of learning is to reach a long-term maximum reward from the environment. The Q-value update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.1}$$

Q-learning is one of the most popular forms of RL. However, it converge very slowly to a good policy. Q-learning essentially creates up a look-up table that, for any state, the possible actions and their relative values are listed by the table. Since it is difficult to explore all possible actions from all possible states (even once, let alone several times), we would like to implement the critic in a way that leverages reasonable generalization of the values of known states and actions to unknown state and/or actions, which a table cannot do. Generalization allows for the efficient storage of learned information and broad utilization of the knowledge between "similar" states to actions.

In the RL algorithm an optimal policy can be learned eventually and a "good" policy learned quickly, where a *policy* is the mapping from states to the best possible action for each state. The knowledge of past experiences actually exists as reward values for each state. The learning procedure essentially establishes the mappings from the states to the optimal values for those states.

It is possible to build "localized" association models that are very similar to a look-up table. However, for the task that has a very large state space or action space, traversing every state and action is not possible. A more general structure is needed and preferably one that allows reasonable generalization so that the agent can make efficient use of its experiences. The RLAM system generalizes very well and will be discussed later in this chapter.

One of the important strengths of neural models is their ability to generalize, which is due, in part, to the use of distributed representations. The nodes in the network can be trained to respond to a certain set of states, and the networks can be trained to store a variety of mappings, including: state $\rightarrow$ policy, state $\rightarrow$ value functions, state+action $\rightarrow$ reward, etc. We will use the Q-learning technique in our dynamic learning Palm network, which implements the two mappings: $S \rightarrow a$ and $S \rightarrow Q$. Both networks are one-layer association networks with no hidden layers, and k-WTA is used in $S \rightarrow a$. We will see later that the k-WTA step enables a one-layer RLAM system to quickly learn complex mappings.

4.4 The RLAM System

4.4.1 The Model

We chose the grid-world goal finding example to compare the RLAM with the table driven Q-learning system. A grid-world of 15×15 is used. Each grid corresponds to a state, and in each state, 4 different actions are possible: moving one step north, south, east or west. Each action traverses only one square.

A robot (the agent) starts at a random position in the grid, and the task for the agent is to find the goal point that is located at the center of the grid. This task can be extended to a more general task in which the agent starts at a random grid and the goal is also located at a random grid. Another variation is to add barriers which must be traversed, such as maze learning. We have experimented on the general grid-world task, and moving the smaller space to a larger space is fairly straightforward, so to simplify the discussion, we only discuss the grid-world task where the goal is always set at the center of the grid. But RLAM is very effective even in the presence of complex barriers.

In the grid task, when the robot finds the goal, a new training epoch begins and the agent is started at a new random state. The reward is either 1 or 0, corresponding to finding or missing the goal respectively. The states are represented by a 15×15 two-dimensional array of neurons. For each state that is not at the edge of the grid, the 9 neurons in a 3×3 field surrounding the neuron are activated. For some states that are at the edge or the corner of the grid, at least 4 neurons are activated.

We constructed two networks, one is the "actor" and one the "critic:" W_{act} and W_{critic}. W_{act} implements the mappings from the state S to the action a, and W_{critic} implements the mappings from the state S to the value function Q. Both networks take the state of the robot as input. The output of the network W_{act} is the robot action vector, which has only 4 elements, corresponding to the 4 directions an action may take. The action vector can have only 1 active element. The output mapped from the network W_{critic} is the Q-value represented by the state's value.

W_{act} is initialized to random values, and all the elements in W_{critic} are initialized with the small positive value of 0.1 to set up connections between all the states and the output value neuron. The action vector is calculated by Equation (4.2), which is the same rule used in the Palm retrieval process. In the action vector, there is always only one neuron that is active, though other kinds of encodings are possible. If, by Equation (4.2), there is more than one active neuron, the agent randomly chooses one action from all the possible options for that state. This means that more than one action is possible for that state. This technique also injects a certain degree of randomness into the action selection which is necessary for the network to adequately explore its environment ("exploration").

The algorithm for RLAM is described by Equations (4.2)-(4.9). The weight adaptation for W_{act} and W_{critic} is shown in Equation (4.8) and Equation (4.9) respectively. In the equations, $\cdot$ represents the inner product:

$$a_t = f(W_{act} \cdot S_t - \theta) \tag{4.2}$$

$$(S_t, a_t) \rightarrow S_{t+1} \tag{4.3}$$

$$Q_t = W_{critic} \cdot S_t \tag{4.4}$$

$$Q_{t+1} = W_{critic} \cdot S_{t+1} \tag{4.5}$$

$$Q_t^* = r + \gamma Q_{t+1} \tag{4.6}$$

$$\varepsilon = Q_t^* - Q_t \tag{4.7}$$

$$\Delta W_{act} = \varepsilon S \tag{4.8}$$

$$\Delta W_{critic} = \alpha \varepsilon S \cdot W_{critic} \Big/ \sum_{i,j} W_{critic}(i, j) \tag{4.9}$$

In the above algorithm, at time t, the agent is in state S_t. The action vector a_t is computed from W_{act}, then the agent takes the corresponding action, entering a new state S_{t+1}. The values for states S_t and S_{t+1} are obtained from equations (4.4) and (4.5) respectively. The error is calculated by equation (4.7), which in turn controls the adaptation of weight matrices W_{act} and W_{critic}.

After learning, both W_{act} and W_{critic} are normalized incrementally by the use of the Sigmoid function, so the weight values are constrained to be between 0 and 1. This forces a slight inhibitory effect on the weight values and prevents them from growing to

arbitrarily large values. The weights then require multiple bits to represent. We have not yet explored the minimum precision required for effective incremental learning. Limited precision also eliminates one problem with Hebbian where the weights can increase without bound. Since we are assuming a system with stationary probabilistic behavior, which is manifest by the goal and subsequently the reward signal being fixed in time, no additional subtractive or decay mechanism is required. The weight values can also be thought of as representing the confidence or probability of the strength between the input and output neurons.

4.4.2 Implementation – Grid-world Mapping

We compared the results from the RL based associative memory with results from the traditional Q-learning system. Using the RL based Associative Memory (RLAM) structure, the learning process converged more rapidly than the Q-learning Look-up Table (QLT) system, and RLAM's ability to generalize was much better than QLT. In the terminology discussed below, each action is one step, and an epoch is the series of steps that the agent takes to accomplish the task.

Figure 4.1 shows that, for a simple grid-world example, after 3000 training steps RLAM achieves a successful retrieval rate of 98%, which means that for 98% of all the grid positions, the agent can successfully accomplish the task. While in the QLT system, after 3000 steps, the system attains 11% successful retrieval rate, only after 40K training steps, can the agent achieve 97% successful retrieval rate. The data in Figure 4.1 are from one run of simulation, many independent runs of simulation are tested and show the similar results. The data discussed in this section are from the same sun of simulation as Figure 4.1.

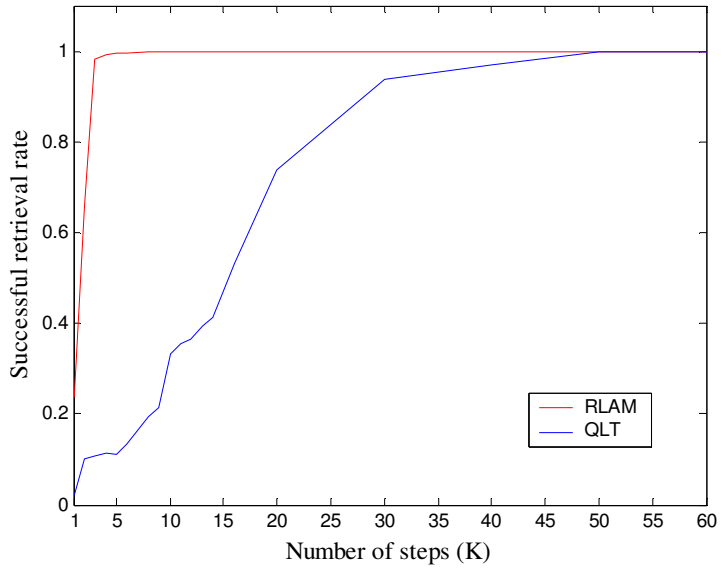

Figure 4.1 Retrieval performance versus total number of training steps

The RLAM learns the grid-world much faster than QLT in that RLAM can extrapolate each learnt position to a region of positions that are similar to the learnt one. This generalization ability allows RLAM to learn only a limited number of examples and can reasonably predict the unseen ones fairly quickly.

Even after 3000 training steps in RLAM, the agent has not traversed every grid point. As shown in Table I, to achieve a successful retrieval rate of 98% in RLAM, the agent has been trained by 137 epochs. For each new epoch, the agent starts at a new random grid point. There are a total of 225 grid points, so not all the grid points were visited by the agent. However, the agent has learned enough to be able to generalize successfully to the unvisited states. In the QLT system, the agent was trained for 2145 epochs, which means that each grid point was visited on average of 9 times ($2145 \div (15 \times 15)$). Since the look-up table does not generalize well, the agent has to traverse every state several times, until it acquires enough experience about the environment to operate successfully.

TABLE I

	Number of Steps	Number of Epochs	Successful Retrieval Rate
RLAM	3K	137	98%
QLT	40K	2145	97%

In the RLAM system, after training, the network W_{act} has self organized corresponding to the grid-world. The weight values can be illustrated as gray-level images with 0 representing black and 1 representing white. Figure 4.2 shows two examples of grid-world problems: A(1) is the simple grid-world with the goal at the center. B(1) adds some barriers to the grid-world used by A(1). After training, each output neuron associates with different groups of states, and the associated states vary with different road maps.

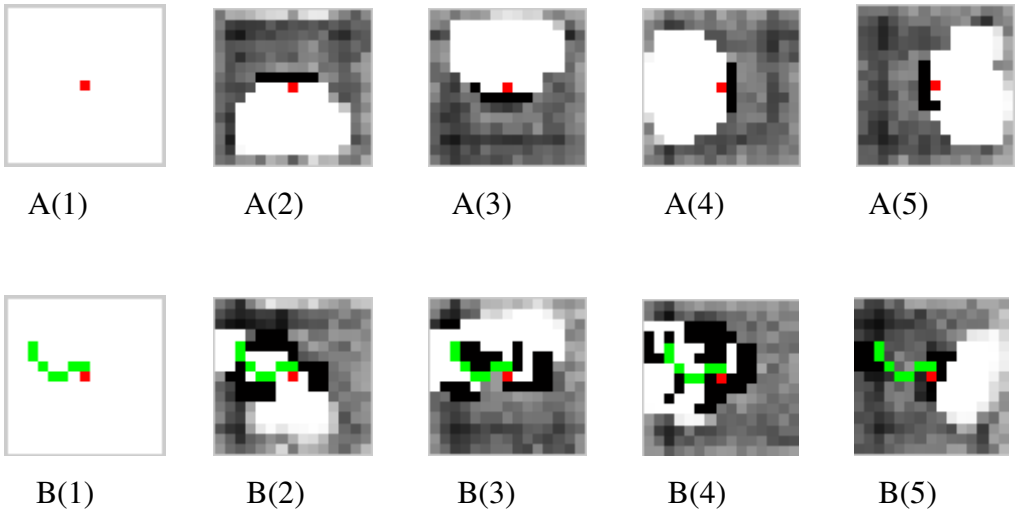

Figure 4.2 Different road maps in grid-world associate with different actions. A(1) is the simplest grid-world, and the goal (the red point) is at the center. A(2)-A(5) are pseudo gray-level images of weights corresponding to four output neurons, with 1 representing white and 0 representing black. A(2)-(5) illustrate the weight values that associate the states with the corresponding actions of going north, south, east and west respectively. B(1) is a grid-world that has barriers colored in green. B(2)-(5) correspond to the weights that associate the states with the actions of going north, south, east and west respectively.

Before training, the state-to-action mappings are random, and the weights are random. After several training iterations have taken place, the network will gradually self-organize. If more than two action neurons are prominent in any state, the agent can randomly choose one direction. For the final goal, there is not much difference between choosing which action to take, and in real world problem, there is usually more than one option of comparable value.

The learning speed and efficiency in RLAM is better than the QLT, and RLAM is more biological in that the system adapts incrementally via its interaction with the environment. Also, the memory of RLAM converges more quickly and generalizes better than QLT. As the training set derives from the various positions the agent is in when it explores the environment, to learn enough about the environment to have an adequate representation of the environment, the agent needs to explore the environment sufficiently to generate those training vectors. Therefore, for more complex mapping problems as shown in Figure 4.2 B, special training vectors around the barriers are needed. After training, every training vector owns its weighted region: if the training vectors around barriers are not trained, the normal training vectors would have large influence on the barriers and the barriers would not be represented by the normal training vectors; when the training vectors around barriers are trained, those special training vectors will gradually form their own regions to override the dominance of the normal training vectors so that the barrier details can be represented. As we discuss below, with sufficient exploration time, RLAM can learn arbitrarily complex landscapes in terms of arbitrarily placed barriers – as long as a path exists to the goal.

4.5 Complex Mappings

The grid problem shown in last section is relatively simple. The next step is to investigate the computational capabilities of RLAM in a real application and to assess its ability to create the necessary representations under these conditions.

It's been shown that Q-learning always converges to the optimal values with probability one as long as all actions are repeatedly sampled in all states and the action values are represented discretely (Watkins and Dayan 1992). This implies that, as long as

there is enough memory to store all the temporary estimation of the values for each possible state-action pair, i.e. in a look-up table, the optimal value function can be found.

In real-life problems with large state spaces, especially when it is impossible to visit all states, and there are many possible actions in each state, generalization is absolutely necessary. The use of the associative memory's function approximation capability with RL works very well. An associative memory implementation does not need nearly as much storage as the look-up table, as shown in Chapter 2, it thus speeds up the learning process. And, unlike gradient descent algorithms such as back propagation, it trains very rapidly. Generally, associative memory learning is one-shot learning, which learns very fast. Here we add incremental modification to the weight matrix, in most part, it still belongs to one-shot learning.

The Palm associative network has the ability to do basic generalization for noisy and incomplete input. In dynamic learning, the associative network partitions the input space into arbitrarily sized weighted regions, with each influenced by a corresponding influence coefficient. The influence coefficient is a value that represents how dominant a training vector is in the whole network, thus the weighted area of a region illustrates the generalization ability of that particular training vector. In the RLAM system, the influence coefficient of a particular grid position is represented by the number of times the agent visits that position. In the grid-world problems discussed here, at the beginning of training all grid points are almost equally visited by the agent, thus the influence of each grid point is almost equal. As training continues, the grids near the goal are visited more often, their dominance become stronger and they are gradually extended further into more peripheral areas. In the situation where there are barriers in the grid, as the grids near the barriers are visited more often, the influence of those grids may eventually override the dominance of the regular grids, so that the special actions corresponding to the barriers can be learned and reinforced over the regular actions corresponding to the normal grids.

The question then is whether RLAM can implement arbitrarily complex grid-world maps. Can the weighted regions represent arbitrary mappings? Although multi-layered perceptrons are good function approximators (Hecht-Nielsen 1987; Hornik, Stinchcombe et al. 1989), associative memory is normally not considered to be a function

approximator. A thorough study of this aspect of associative memories requires a much more rigorous treatment and is beyond the scope of this dissertation. However, Maass has shown that "any Boolean function f: $\{0,1\}^m \rightarrow \{0,1\}$ can be computed by a single k-winner-take-all gate applied to positive weighted sums of the input bits", and "circuits consisting of a single soft winner-take-all gate applied to positive weighted sums of the input variables are universal approximators for arbitrary continuous functions from R^m into [0,1]" (Maass 1999; Maass 2000).

Therefore, any mapping from states to actions can be implemented by a one-layer positively valued network using k-WTA as the only nonlinear operation. For a complex mapping problem, as long as the training vectors that describe the specifics of the input space are trained properly (with appropriate influence coefficients), such mappings can be represented by the RLAM system. Although the network parameters, such as the network size, the number of training vectors, and the exact value of influence coefficient for each training vector, etc., are not specifically determined and can vary with the problem, in theory, any complex mapping can be represented by a RLAM-like associative memory structure. We demonstrated in the previous section that associative memory models do well in this role and that for many real world applications they will be sufficiently robust. In our Enhanced Vision System (EVS) for aircraft landing guidance (Luk, Gao et al. 2004), an auto-associative memory is used to efficiently retrieve a clean best-matched image from the noisy input sensor image.

Although associative memory can be regarded as a function approximator, can the combination of function approximator with RL guarantee an optimal policy for arbitrary function approximation? There are limitations on how well a function approximator, when used with RL learning, can approximate arbitrary complex mappings. This limitation is due to the noise generated by the function approximator. According to Thrun and Schwartz (Thrun and Schwartz 1993), if the generalization error is independent and uniformly distributed, Q-learning may fail to provide the optimal policy. The failure is caused by overestimation of the value function by the critic. In Q-learning, the actor will always choose the action corresponding to the maximum estimated value of a state-action pair. Due to the approximation error, some values may get larger, and the max operation

makes it more likely that it will overestimate for the corresponding actions. Thus, the mean number of errors may become a positive number instead of zero.

The RLAM, as implemented here, uses two networks: one that maps the input states to value functions (the critic); and another that maps the states to actions (the actor). With the associative memory k-WTA, the network has the ability to handle non-linear maps. The ultimate policy that RLAM actually learns is a mapping between the state and the corresponding action. Although there are limitations on arbitrary approximation due to the combination of associative memory and RL, RLAM can learn complex functions if enough training patterns are sampled in a reasonably large network. In the early stage of training when insufficient training patterns have yet been sampled, there are a few large regions that are crude approximation to the final mapping, with ever more experience, a more complex structure that represents the final solution then evolves.

This incremental learning characteristic is one of the major strengths of RLAM, where some minor weight adjustment is applied at each epoch. In the dynamic learning of neural-like model, there are two different approaches to learn. One is the incremental method where the weights are adjusted according to some cost function. Back-propagation and restricted Boltzmann machine fall in this category. The other method is one shot learning where weights are constructed by a batch of training vectors. Palm network and Granger's model (Coultrip, Granger et al. 1992; Coultrip and Granger 1994) belong to this category. So far the incremental learning has built the most useful networks, the one shot learning, however, requires much larger and more complex networks. One shot learning is more biologically realistic in that it learns very quickly, can leverage custom hardware more effectively, requires much lower precision and has more efficient weight update. The RLAM system we built here is based on Palm network, for the most part, it belongs to one shot learning, though some incremental modification on weights is applied.

Chapter 5

Spiking Palm Model

5.1 Background

In previous chapters, we have seen certain nice properties of the Palm model: sparse representation, large capacity, fault tolerance to input and internal errors, and ease of implementation. However, it is still a very abstract model. Perhaps the biggest simplification is that no temporal information is used, or, as in some models, is represented spatially. Real neurons compute via complicated electrical and chemical processes that operate through time. And most cognitive tasks require a temporal dimension, and relative timing is crucial in learning (Hopfield 1995; Maass 1997; Abbott and Nelson 2000).

It has always been difficult to add temporal information to neural network models. Although there is no agreement on how inter-neuron information is encoded, many neuroscientists agree that the relative firing time or inter-spike interval of a neuron is important in encoding information in many real neurons and that the varying lengths of dendritic trees contributes delay sensitivity to neuronal information processing. This realization has led to the development of a range of mathematical spiking models (Maass and Bishop 1999; Hopfield and Brody 2000; Hopfield and Brody 2001; Gerstner and Kistler 2002).

The spiking model proposed by Knoblauch and Palm (Knoblauch and Palm 2001a; Knoblauch and Palm 2001b) is used as a model of primary visual cortex. This model is based on the basic binary Palm model, and a biologically plausible retrieval algorithm is introduced, where threshold control is performed by a number of inhibitory neurons. Though the membrane conductance of a neuron is dynamically adapted to the incoming

spikes, the temporal learning between the pre- and post- synapses is not a part of this network, and the connections are inherited from binary model.

There are other spiking models based on various associative networks. Ruf (Ruf and Schmitt 1997) applies the temporal coding in the self-organizing map. In Maass's spiking model, detailed compartmental neuron models are simulated in a Hopfield network architecture. Maass claims that any Hopfield network can be emulated by spiking neurons in temporal coding (Maass and Natschlager 1997; Maass and Natchlager 1998). In Gerstner's spiking associative memory (Gerstner 1990; Gerstner and Hemmen 1992), some biological characteristics of spiking neurons are incorporated in the network, and the static network performance is similar to the results obtained by a spike rate encoded network. The spiking model proposed by Sommer and Wennekers (Sommer and Wennekers 2001) incorporates the compartmental spiking neuron developed by Pinsky and Rinzel (Pinsky and Rinzel 1994), and simple clipped Hebbian learning is used.

Hopfield and Brody (Hopfield and Brody 2000; Hopfield and Brody 2001) propose a network of integrate and fire neurons that can implement spatiotemporal integration. In their network, the activity (spiking rate) decay rate is different for different spiking neurons, and the convergence or synchronization to a common activity level for different neurons represent the recognition of an event. This recognition is robust to temporal delay and invariant to the difference of source generation.

Another integrate-and-fire neuronal networks that can implement Bayesian inference for graphical models is discussed by Rao (Rao 2005). In these recurrent networks, the membrane potential dynamics of neurons is used to implement belief propagation in the log domain, and the spiking probability of a neuron is shown to approximate the posterior probability of the preferred state encoded by the neuron, given past inputs. In the experiments, a single-level network is able to generate inference of motion direction, and a two-level network is able to do hierarchical inference of object identity at an attended visual location.

All the models mentioned above incorporate a number of biological aspects of a neuron, such as postsynaptic potential, Poisson distributed inter-spike intervals, the absolute refractory period, decaying activity, etc. However, learning in the temporal domain is not fully discussed in those models. Learning in the temporal domain is a

complicated process, and it is not completely understood. In this chapter, we investigate and propose a way that temporal learning can be added to the Palm model, and then investigate the network operation in the temporal domain.

5.2 Description of Spiking Neuron Model

The state $u_i(t)$ of neuron i ($1 \le i \le n$), which is analogous to its somatic potential, is determined by two processes: the input from pre-synaptic neurons and its threshold:

$$u_i(t) = \sum_{j=1}^{n} \sum_{f=1}^{n_j} w_{ij}(t) \varepsilon_{ij}(t - t_j^f) + \eta_i(t - t_i) \tag{5.1}$$

where w_{ij} is the efficacy of the connection between neuron i and j; $\varepsilon_{ij}(t - t_j^f)$ is the postsynaptic potential of neuron j contributing to neuron i; $\eta_i(t - t_i)$, the threshold, describes the refractory effect of the output neuron i that fires at t_i.

In our simulation, to implement the neuron model in equation (5.1), we use $\eta(t)$ as

$$\eta(t) = \begin{cases} -\infty & 0 < t \le \tau_{ref} \\ -\eta_0 / (t - \tau_{ref}) & \tau_{ref} < t \end{cases} \tag{5.2}$$

where τ_{ref} is the absolute refractory period. In the simulation, after each neuron fires, we set the state to 0 in the refractory period.

The postsynaptic potential function $\varepsilon(t)$ we used is

$$\varepsilon(t) = \frac{t - T_d}{\tau_s} \exp\left(1 - \frac{t - T_d}{\tau_s}\right) H(t - T_d) \tag{5.3}$$

where $H(t)$ is Heaviside function, and T_d is the time delay.

Hebbian learning is used during the training procedure, and can be implemented by the timing window

$$\Delta w_{ij}(s) = \begin{cases} A \exp(s / \tau_{ltp}) & s \le 0 \\ -A \exp(-s / \tau_{ltd}) & s > 0 \end{cases} \tag{5.4}$$

where $s = t_j^f - t_i$. Also, $w_{ij}(t)$ is constrained to lie within the interval $0 \le w_{ij}(t) \le W_l$.

5.3 Implementation

5.3.1 Hebbian Learning

Using Equation (5.4), the Hebbian learning between the pre- and post- synapses is illustrated in Figure 5.1, where only one synapse is shown. There are two strings of spikes: the upper one is pre-synaptic spikes, and the lower one is the post-synaptic spikes. The blue line shows the strength of the connectivity (weight) of the synapse. In the classical Palm model, there are no explicit inhibitory neurons, the value of the weight is unchanged if no further adjustment is made. So no weight decay is considered in the simulations. The weight adjustment depends on the relative timing of pre- and post-pulses, one example of STDP (Spike Timing Dependent Potentiation) learning. The timing window of (-3,3) is used in Figure 5.1. At time step 50, both pre- and post-synapses fire exactly at the same time, the weight has the largest positive adjustment.

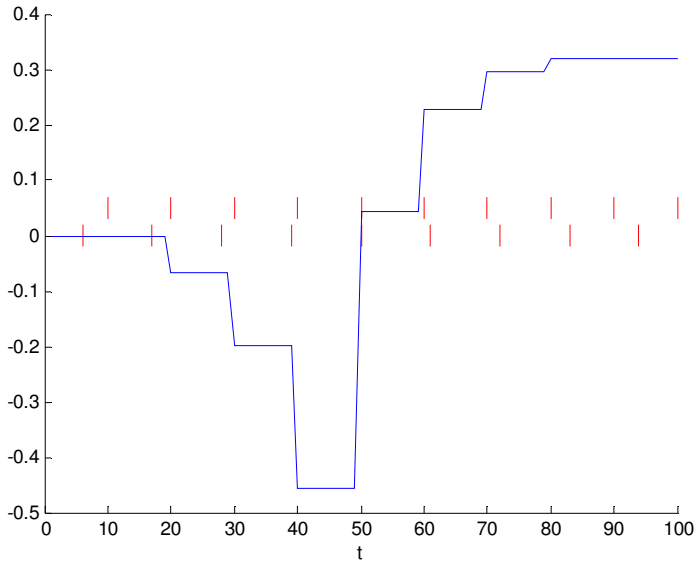

Figure 5.1 The Hebbian learning between pre- and post-synapse.

When a larger timing window of (-30,50) was used, the weight modification is as shown in Figure 5.2. The blue curve indicates the weight associated with the synapse. The upper string of spikes is the pre-synaptic neuron activity, and the lower string of spikes is the post-synaptic neuron activity. For $0 < t < 200$, both the pre- and post-

neurons fire at the same time; for $200 < t < 400$, the two are uncorrelated, both fire randomly; for $400 < t < 600$, they are anti-correlated; for $600 < t < 800$, the post-neuron fires 5 time steps earlier than the corresponding pre-neuron; and for $800 < t < 1000$, the post-neuron fires 5 time steps later than the pre-neuron.

There are minor adjustments of the weight during $200 < t < 400$ when both synapses fire randomly. This is due to the occasional coincident firing of the pre- and post-synaptic neurons.

The learning method used here is biologically plausible. Experimental results show similar STDP in cortex and the hippocampal neurons of rat barrel and cerebellum-like structure of electric fish (Abbott and Nelson 2000). However, it is unlikely that a single pair of pre- and post- synaptic activities would incur the learning mechanism used here. How many pre- and post- synaptic spikes are needed for learning is still not known. The length of the timing window used here is just an empirical value. Still, Hebbian learning is clear.

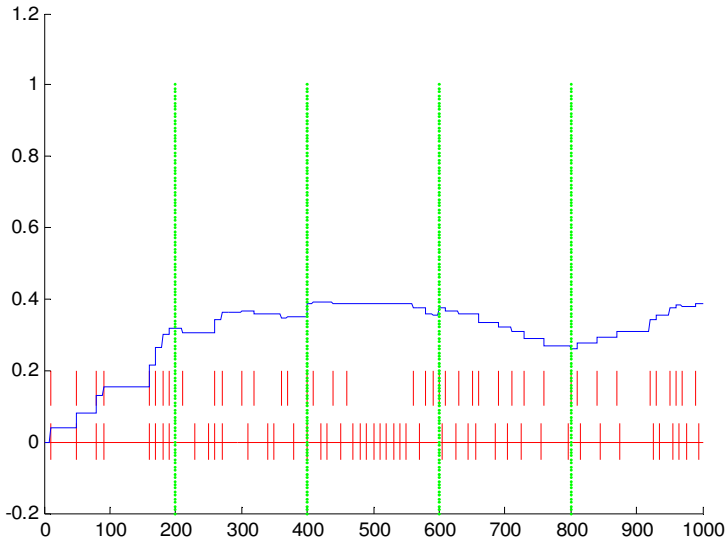

Figure 5.2 Hebbian learning between input and output strings of spikes.

5.3.2 Network Retrieval Rrobustness

To train a spike based associative network, the training spikes are clamped (forced) at both the input and output. The output neurons will fire later than the input neurons. After training, the weights that result from clamping are very similar to the weights obtained from clipped Hebbian learning. Consequently, for simplicity, we can just use the weights that are established by clipped Hebbian learning, which means that the values of those connections are either 1 or 0.

In our simulation (Zhu and Hammerstrom 2002), there is not much performance difference between linear and non-linear function approximation of post-synaptic potential. Therefore, piece-wise linear functions are used to approximate the post-synaptic potential. The linear function is very easy to implement in digital simulation and FPGA like hardware. We also assumed that there is no time delay involved in transmitting the action potential from the dendrite to the soma. The simplest post-synaptic potential is illustrated in Figure 5.3, where the pre-synaptic neuron fires at time step 1, and the post-synaptic potential is shaped like a triangle. The charging period from time step 1 to 10 is referred as the *rising range* of the neuron.

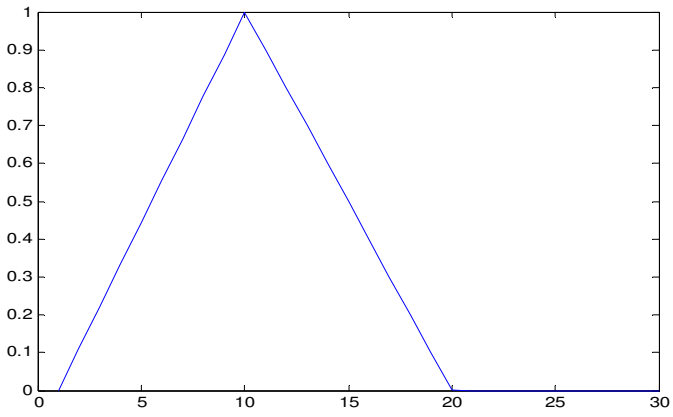

Figure 5.3 Piece-wise linear post-synaptic potential

Suppose there are n pre-synapses contributing to an output neuron, and the firing times of the n synapses can be denoted as s_i $(i=1,2,\ldots,n)$. Also, suppose all the firing times are within the rising range T of the first input synapse firing at s_1, that is

$s_1 \le s_j \le s_1 + T$ *(j=2,3,…,n).* For simplicity, if the discharging5 phase has the same slope as the charging phase, then the accumulated peak potential *P* at the output neuron is:

$$P = k \cdot n \cdot T \tag{5.5}$$

where *k* is the slope of charging phase, and is constant.

In the binary model, the k-WTA rule is used to determine the threshold. In the spiking model, a population of inhibitory neurons can be included in the network such that the threshold is determined adaptively (Knoblauch and Palm 2001b). To be consistent with the binary model, for the experiments performed here, a modified k-WTA rule was used: the threshold of the spiking neuron is set to allow only those neurons that have the k largest peak potential to fire. It can be easily seen that, as long as all the firing times are within the rising range of the first spike synapse, the spiking network will have exactly the same retrieval performance as in the binary (synchronous) network. In addition, the network performance does not depend on the specific firing times of the pre-synaptic neurons, and the network is robust to jitter within the interval T.

When there is false (incorrect) firing and false silence (no firing), let the number of true (correct) firing neurons be n_T, and the number of false firing neurons be n_F. Suppose the true firing neurons all fire within T, while the false firing times are uniformly distributed. The peak potential of neuron output can be written as

$$H = k \cdot n_T \cdot T + \delta \tag{5.6}$$

where δ is the noise caused by n_F false firing neurons.

If there is a connection between the false firing neuron and the output neuron, for the binary network, the false firing neurons will always contribute to the output potential, so the noise δ' in the binary model satisfies the relationship $\delta' = k \cdot n_F \cdot T$. In the spiking model, only those false firing neurons that fire within $s_1 < t < s_1 + 2T$ can contribute to δ, so the number of false firing neurons is less than or equal to n_F, thus $\delta \le k \cdot n_F \cdot T$, and $\delta \le \delta'$. Therefore, the spiking network can always have equal or even better performance than the binary network, and is more robust to random temporal noise. Figure 5.4 illustrates the activity of a simple spiking model.

In the spiking network, for a binary input vector, a “1” will be represented by a spike that stochastically fires within a certain time interval, that is, the vector becomes a

set of pulses that occur near each other in time, but they need not occur at exactly the same time. In the simulation, for the testing on an input pattern, the firing times of the true firing neurons are normally distributed, the mean firing time is 5 and the variance is 2; and the false firing neurons are allowed to fire uniformly between 1 and 10.

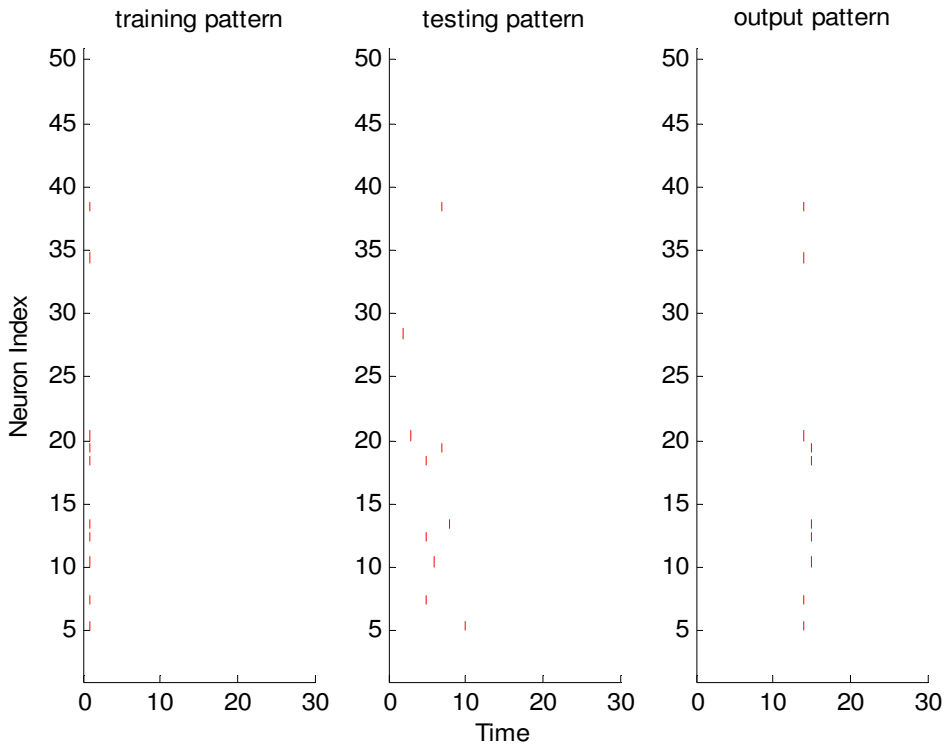

Figure 5.4 Pulses representation of training, testing and output patterns.

In the simulation shown in Figure 5.4, 50 neurons were used, and 15 training vectors are stored in the 50×50 network. The bit-flip error rate in the test vectors is 0.1. The output error rate in the binary network is 0.0465, and the output error rate in the spiking network is 0.0067. In the testing pattern shown in Figure 5.4, neuron 34 is not firing though it is supposed to fire in the training pattern, and neuron 28 is incorrectly firing in the testing pattern, since it does not fire in the training pattern. The output pattern recalled is exactly the same as the training pattern, except for a little jitter. As can seen, the spiking model is more robust to temporal jitter, and outperforms the binary model.

The simulation results shown above demonstrate the robustness of the spiking model to temporal noise (jitter). Since there is no temporal correlation between different nodes, association through time is not shown here. In sequential association, several time varying inputs corresponding to the same input pattern should be mapped to the same output pattern. With the current model, to show sequential association, time varying weights, rather than 0 or 1, would need to be assigned to the nodes, and the weighted temporal integration of the inputs should be unique to a particular input pattern. In this case, small temporal error tolerance would still be possible.

This chapter shows a simple example of how the Hebbian learning can be implemented and lead to good performance in a spiking version of the Palm network. We have also discussed a simple implementation of spiking model which utilizes the temporal information in training. How the inter pulse interval can be utilized to represent information is not discussed here. The temporal information captured in this spiking neuron model is the correlation between input and output neurons. In real world application, the temporal information between the input sequences of patterns is important too (George and Hawkins 2004; George and Hawkins 2005; Hawkins and George 2006; George and Jaros 2007). In Chapter 4, we have discussed a reinforcement learning based RLAM that enables an interactive reaction of the system to the outside world. The interaction actually learns correlation between inputs, which shows an example of learning sequences of inputs.

Chapter 6

Hierarchical Associative Networks

6.1 Scaling Problem

In the Palm network, even though the information capacity is relatively large, as more information is added to the network, the weight matrix can become quite full; for an auto-associative network, the maximum storage capacity is about 35%. As Palm shows (Palm 1980), the maximum capacity of the associative memory occurs when there are roughly an equal number of 1s and 0s in the weight matrix. Palm networks can scale to reasonably large sizes, but ultimately they still become limited by the connectivity. For example, if we have a one million node associative memory with 50% connectivity, each node will take 500K inputs. Assuming a reasonably random distribution of 1s and 0s, here the law of large numbers causes the sums computed by each node to asymptotically equal, which makes the network incapable to distinguish between nodes.

Biological neural systems have huge storage capacities and yet very sparse connectivity. For example, in human neocortex there are about 10^{10} neurons with each node (neuron) having around 10-20K inputs, for a connectivity of about one $10{,}000^{th}$ of 1%. If we ran a Palm network of about that size and connectivity, it is unlikely to work at all. Clearly we are still missing some piece to this puzzle. And it is an important piece, since to be able to use associative memories in many real applications, we may need to scale to very large structures.

Since the numbers of connections would increase exponentially, 50% connectivity is simply not possible. As mentioned, real networks are extremely sparse. So how do we create versions of our basic association algorithm that operate with such sparse connectivities. According to Braitenberg (Braitenberg and Schuz 1998), there are two

kinds of connections in cortex: metric and ametric. The *metric* connections are very dense connections which are localized to a node's local neighborhood. The *ametric* connections are much sparser, random, and are point-to-point connections with seeming arbitrary random connectivity to a wide variety broadly dispersed groups. Consequently, the fully connected network is not biologically plausible. Furthermore, it is hypothesized that large groups of modules may have different learning and connectivity patterns, which allows them to specialize on certain types of functionality, creating a larger and more complex system.

In this chapter, we will discuss the scaling of associative memory and, in particular, that of a hierarchically connected network structure that is scalable. We present a modular associative model based on a structure we call a Bayesian Memory (BM). We propose that neocortex can be hierarchically organized from a large number of Bayesian Memories. In the hierarchy, for example, some lower level networks may store different aspects of phenomenal information of an event, such as color, shape, length, etc., while other, higher level networks can store more abstract information of an event, such as concept, language, thinking, etc. These higher level networks can have longer range connections across the lower level networks, and they can compose even higher level networks. Information is propagated both downward and upward.

6.2 Implementation Cost

Given the same total number of neurons, there are different ways to build a large network. We can build a group of smaller networks, hierarchically connected, or, we can build a plain large network. In this section, we compare the computational cost of constructing a huge network with that of building groups of smaller networks. We use the binary Palm network as the basis for our comparisons.

In Chapter 3, we showed that network performance is mainly determined by the network density, since the number of active nodes k does not vary significantly with the network size n. If $k = \log n$, then the density ρ_l of a network that determines the network performance is:

$$\rho_l = 1-(1-\frac{k^2}{n^2})^M \tag{6.1}$$

where M is the number vectors being stored. Equation (6.1) can be simplified as:

$$\rho_l = 1-\exp(1-\frac{M\cdot k^2}{n^2}) \tag{6.2}$$

Now suppose we need to store $l\cdot M$ vectors, we have two strategies to fulfill the task: one by constructing a larger network with network size of n' to keep the same retrieval performance; and the other by using l smaller networks, with each being size of n and storing M vectors.

According to the neuroanatomy model of Braitenberg (Braitenberg 2001), the relationship between the volume V of the brain and the number of neurons N satisfies:

$$V = N^{3/2} \tag{6.3}$$

Then the averaged length *d* of the connection between any two neurons in volume V is:

$$d = O(V^{1/3}) = O(N^{1/2}) \tag{6.4}$$

Therefore, for a fully connected network, the total length, *L,* of connections for all the neurons in volume V is:

$$L = N^2\cdot d = O(N^{5/2}) \tag{6.5}$$

The implementation cost is directly related to the total length of connections. Now to compare the implementation costs for the two strategies:

Strategy 1: Build a larger network, with size n'

To keep the same retrieval performance, the density of the larger network should still be kept the same as ρ_l. Since, the number of active nodes need not change much, we assume the number of active nodes in the larger network is still k. By equation (6.2), the network size n' of the larger network should be $n' = \sqrt{l}\cdot n$.

By Equation (6.5), the total length of all connections is:

$$L_1 = O[(n\sqrt{l})^{5/2}] = l^{5/4}O(n^{5/2}) \tag{6.6}$$

Strategy 2: Building a group of smaller networks without inter-connections

There are a total of *l* networks, each with size of *n*. Then the total length of connections in *l* networks is:

$$L_2 = l\cdot O(n^{5/2}) \tag{6.7}$$

Comparing Equations (6.6) and (6.7), it can be seen that having a group of small networks can potentially be less costly than building a larger network, without sacrificing basic network performance. Also, these small networks can operate in parallel (including inhibitory k-WTA like operation), which is more computationally efficient than the larger network. Another potential benefit of building a series of small networks is that small networks can represent lower-level information via interconnections among these smaller networks, higher level of networks can be formed to implement more complex mappings and can represent more high level information or abstractions. Hierarchical structure is common in biological system.

In brain, the volume of cortical gray matter *G* correlates with the volume of white matter *W* by a power law across nearly the full range of mammalian brain sizes:

$$W = \frac{c}{T} G^{4/3} \tag{6.8}$$

Where *c* is a dimensionless constant and *T* is the cortical thickness of the gray matter (Zhang and Sejnowski 2000). The gray matter contains cell bodies, dendrites and axons for local information processing, and white matter contains long axons between distant cortical areas. This suggests that the long connection volume increases more quickly than the surface area. Theoretical analysis (Zhang and Sejnowski 2000) shows this regularity might arise as a consequence of the compact wiring of neuronal interconnections.

6.3 Brief Review of Some Hierarchical Memory Models

In the nervous system, it is impossible for each neuron connects to all the other neurons considering the conduction time and volume expansion of the brain (Ringo, Doty et al. 1994; Anderson 1999). Also, the longer the communication channel, the more energy is needed for communication and the longer it would take to transmit signals. Prey and predators both have reasons to keep reaction times as short as possible. Neural circuits are capable of massive computation, and are very energy efficient and fault tolerant (Laughlin and Sejnowski 2003). Anatomy shows that the long range connections are much sparser than more local connections.

As already mentioned, Braitenberg and Schuz (Braitenberg and Schuz 1998) divided intra-cortical connectivity into two classes, metric and ametric connectivity for

the short and long range connections respectively. The metric connections are local and depend on the distance in that the probability of a connection is inversely related to the distance between the cells, while the ametric connections have no such simple dependence on the distance and appear much more random. It is not clear how the long range connections are established by either relay or transponder neurons in cortex or by the white matter (of myelinated connections). Braitenberg proposes an architecture for this connectivity, where the N neurons in the brain are subdivided into $n = N^{1/2}$ compartments, with each containing $n = N^{1/2}$ neurons; the neurons in a compartment can each send one fiber to one of the other compartments (Braitenberg 1978).

There is a certain amount of randomness in neural connectivity, among other things there simply not sufficient genetic material to specify the destination of every axon. However, the number of connections and the specific areas connected seems are non-random. One interesting example concerns the work of Hasson et al. (Hasson, Nir et al. 2004), where they showed that human beings watching the same video clip (with no other constraints placed on the task) tended to activate the same basic areas all over visual cortex, even in the higher order visual areas. The individual connections between neurons seem to connect randomly within the target area.

There has been very little work on the scalability of artificial neural network models and in particular of associative memory models, which, though still limited, have been shown to scale to reasonably large sizes, as was discussed earlier in this thesis. O'Kane and Treves (O'Kane and Treves 1992a; O'Kane and Treves 1992b) proposed a similar neural model similar to Braitenberg's. In their model, there are also short-range and long-range connections, and both connections are formed by Hebbian learning. They assumed that the short-range connections are densely connected, while the long-range connections are random and uniformly distributed. Their analysis shows that this structure is not an appropriate model for the associative memory structure in the brain. Their simple model does not explain either the high information storage capacity or the stability of information retrieval in very sparsely connected associative networks. It is important not to extrapolate too much from their work, since there are a number of short-comings in the biological plausibility of their model. For example, their assumption that

the long-range connections are random and uniformly distributed does not fit what we know about neocortex.

Two other attempts to come to scalable associative memories are the network of networks structure proposed by Anderson and Sutton (Anderson and Sutton 1995), and Cortronics developed by Hecht-Nielsen (Hecht-Nielsen 1998; Hecht-Nielsen 1999; Hecht-Nielsen 2003b; Hecht-Nielsen 2003a). In Cortronics, neocortex is divided into regions (approximately 150K of them), each region is tightly connected with a partner region in the thalamus creating a BAM (Bidirectional Associative Memory). These regions are roughly comparable to cortical hypercolumns and act as separate independent networks on the order of about 10K nodes each. During the training period (gestation and early childhood) each region forms a "lexicon" of specific patterns. These regions are then connected to each other in higher level structures that allow for complex relationships among the regions, such as "antecedent support networks" as well as hierarchies.

The antecedent support networks use a method similar to that of BCPNN, where the conditional information between two nodes is learned incrementally. It should be noted that BCPNN also uses hypercolumn like regions that do winner-take-all processing. Among hypercolumns, a patchy connective mechanism is used to improve the retrieval performance and storage capacity. The patchy connection is similar to the ametric connection that Braitenberg defines, but is more or less random.

Jeff Hawkins's Hierarchical Temporal Memory (HTM) (George and Hawkins 2004; George and Hawkins 2005; Hawkins and George 2006; George and Jaros 2007) is organized as a hierarchy of nodes, where each node implements learning and memory function. Each node on HTM is similar to the node in a Bayesian Networks and uses belief propagation as the fundamental computation. In addition, the nodes can learn temporal correlation via a transition matrix.

Human neocortex implements complex information abstraction mechanisms. During recognition, the basic input provided by visual and auditory senses is passed from the low level reception area to high level cortex, gradually while the data pass through several layers more abstract representations or concepts are formed. These more abstract representations are invariant to many specifics of local input information. For example,

visually different (color, size, variety) apples can be mapped to one common concept in mind that can represent the whole class of apples, even the word “apple” is mapped to the same concept in the mind, as can object position in visual field and scale. This common concept is abstract, and can be recalled by sensory inputs or even inner brain activities.

The information flow, as one sees in Bayesian Belief Propagation (BBP) (Pearl 1988), between low and high levels is bidirectional. A group of low level networks contribute to the high level network. Once a high level concept is formed, it can in turn feed back the high level concept to its lower level networks. To represent abstract concepts, the node or nodes in the higher level network must be more complex and represent the common information embedded in the low-level networks that individually represent a specific aspect of information. Therefore, if some low level network has noisy ambiguous input, the information fedback from the high level can leverage this information by “forcing” the network to a stable state.

One possible model for such a structure is build an associative network where each node represents a multi-dimensional vector. Every input to the node and the output of the node are vectors, with every component in the vector representing the state of the combination of some lower-level networks. With this structure, it is possible that the network can be scaled to very large sizes, and this structure provides a possible way to represent the high order information.

In this thesis, we propose a Bayesian Memory (BM) model that can represent either a low or high level associative network. A BM is a specialized associative network that, in addition to the mapping features of associative network, performs a maximum entropy data reduction of its input to generate its outputs. A network of BMs can then be integrated to form a hierarchical structure.

There is a wide range of implementation possibilities for basic BM functionality. Here we will assume that internally a BM has very densely connected neurons. Between BMs, there are sparser long range connections. We can still assume some kind of Hebbian learning exists between the low and high level BMs. The proposed BM based hierarchical network provides one explanation on how the information is communicated and represented in the neural circuitry.

6.4 Bayesian Memory Based Hierarchical Network

We propose that hierarchical associative networks can be constructed using Bayesian Memories (BMs). A BM works like a communication channel that can convey the most efficient information transportation without minimal data loss. It can also be viewed as a decoder that can retrieve the original message that is "closest" to the noisy input, as discussed in Chapter 3.

6.4.1 Bayesian Memory

A BM can be modeled as an associative memory, however, in addition to the "best match" for a given input, BM partitions the input and output spaces into regions that are centered around its input and output vectors. For our work, we assume that both the input and output vector spaces are discrete. Also, temporal information is ignored for the time being.

The BM internally represents the probability distribution of its input vector space. To be a maximum entropy device, the output vectors, each of which maps to some range of input vectors, should be equally likely. Ideally the output vectors should be a sparse, distributed code, and should form some metric over the output space.

In the simplest case, where there is only one direction of information flow, data is input from the lower level network, with BM output up to the next higher level network. In most cases, information flow is bidirectional, since there is also feedback coming from a higher level network and also being sent to the lower level network. To be specific, the information propagated contains the conditional probability information as the updated belief that is defined in Bayesian networks (Pearl 1988). Another hierarchical model that is built on Bayesian belief propagation algorithm is Numenta's HTM learning model (George and Hawkins 2004; George and Hawkins 2005; Hawkins and George 2006; George and Jaros 2007). In the HTM, the nodes are hierarchically connected as shown in Figure 6.1(A). The message λ sent to its parent is proportional to $P(c_j \mid g)$, the conditional probability of seeing coincidence c_j given that we have seen the group g. Inside each node, messages from below nodes are collected and computed first in the Spatial Pooler, where the bottom-up probability y is generated as a vector that is

proportional to the probability of evidence from below given the coincidences in below nodes. Spatial Pooler contains the learned quantization centers which are the basic component patterns in inputs. The Spatial Pooler output y is then passed to the Temporal Pooler as shown in Figure 6.1(B). The Temporal Pooler groups the quantization centers according to their temporal proximity. A time-adjacency matrix is created while training the Temporal Pooler. The feedback information π in Node k provides top-down probability z to Node *m1* and *m2*, where $z = \sum P(c \mid g_i) \cdot \pi_i / \lambda_i$ (George and Hawkins 2005; George 2008). The *belief* of a node in its coincidence is the product of the top-down and bottom-up probabilities over coincidences: $Bel = y \times z$.

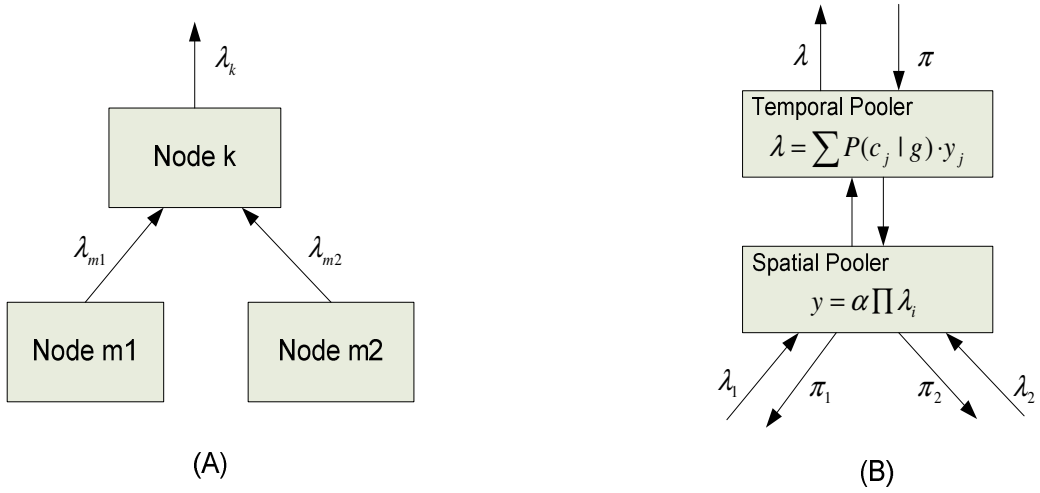

Figure 6.1 (A) HTM structure, (B) Belief propagation computations within a node.

Bayesian Belief Propagation (BBP) networks (Pearl 1988) provides the fundamental theory for HTM. Our BM model is also built on BBP, and borrows from HTM. The interface of a BM is shown in

Figure 6.2, where a BM can have two inputs: one from below V_{Lin} and the other from the above V_{Hin}, and two outputs: one goes to below V_{Lout} and the other goes to above V_{Hout}. The input dimension is L_{size} for V_{Lin} and V_{Lout} , and the output dimension is H_{size} for V_{Hin} and V_{Hout}. For information redundancy reduction, generally the lower level has larger dimension than the upper level, with $H_{size} \leq L_{size}$.

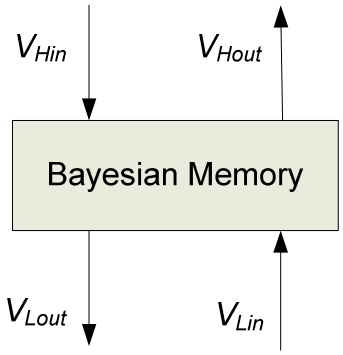

Figure 6.2 Bayesian Memory External Connections. V_{Lin} is the input vector from the lower level network, V_{Hout} is the output vector feeding to the higher level network, V_{Hin} is the feedback vector coming from the higher network, and V_{Lout} is the output vector feeding down to the lower level network.

The lower level input vector V_{Lin} can be a given noisy input or the output of a lower level BM. The upper level input vector V_{Hin} is the feedback vector coming from the upper network. In case there is no higher level feedback, V_{Hin} should be NULL. However, for an auto-associative BM, V_{Hin} can equal to V_{Hout}. If V_{Hin} is NULL, then the V_{Lout} should be NULL too, and there is no feedback in the BM. When an output vector V_{Hout} is produced, the BM output is propagated upward to the higher level.

Inside a BM there is a set of input and output vector pairs that map the input vector space to the output vector space. The vectors provide approximate representation of their spaces. The input and output vector pairs can be thought of as creating a "codebook" within the BM, as one sees in Vector Quantization (VQ). As shown in Figure 6.3, each input codebook vector is an "index" of the index vector *I* and a weight of the weight vector *W*. Each codebook entry maps to an output vector. When evidence for input vectors is presented at the input, the winning index is chosen for the vector that is the most likely based on the distance weighted by *W*, where the distance is measured via some metric, the metric is arbitrary, though VQ tends to use Euclidean. A BM draws inference on the evidence, and this "belief" is the BM output. The codebook vector's weight allows incremental learning when the BM is in dynamic environment by varying the influence that a vector has relative to its neighbors.

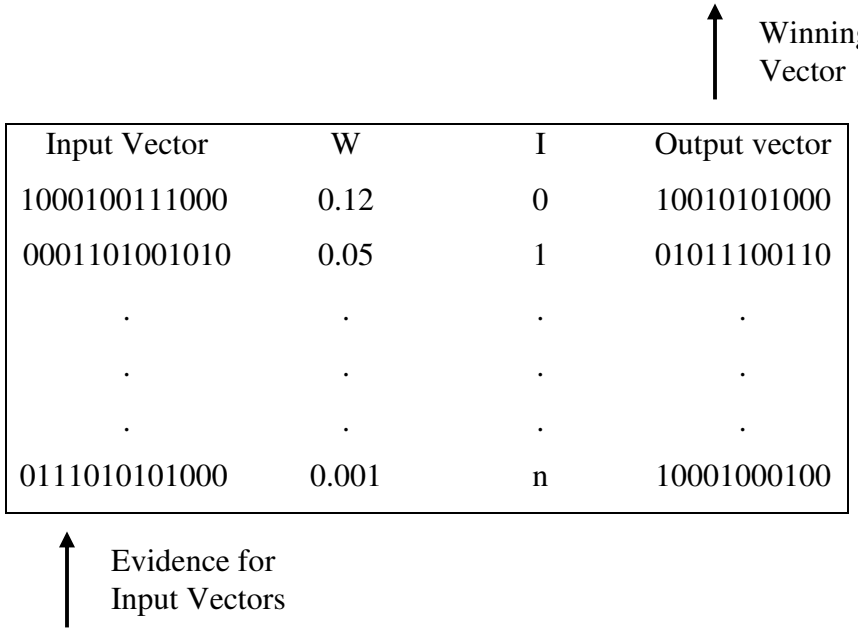

Figure 6.3 A BM codebook.

As mentioned, an ideal BM, as we have defined here, is really a kind of Vector Quantization (VQ). In VQ, there is a code book of vectors that divide up the input vector space. Each vector creates a Voronoi region where all the vectors inside the region are assumed to be noisy versions of the codebook vector. Each vector then has an associated output vector. When an input vector is presented to a VQ, the closest codebook vector in some weighted distance is selected and its corresponding output vector is used as output. Even during non-learning, there is a threshold for the weighted distance. If the distance is too far away from the winning vector, a NULL vector is generated at the output instead. A NULL vector is useful in that it can prevent the memory been distracted too far away from the original one.

In the BM shown in Figure 6.3, data flows in one direction only (upward). If a BM is part of a large hierarchy, the capability of the hierarchy can be significantly improved if, in addition to the winner output vector, each BM node passes likelihood information up the tree. In fact the best results happen when a likelihood vector, of all the possible outputs, is passed forward. Likewise, the upper nodes can pass its beliefs back down to its children. If the tree is non-cyclic (a “poly-tree”), a technique called

Bayesian Belief Propagation (BBP) (Pearl 1988) can be used to pass optimal information up and down the hierarchy. This "belief propagation" we defined here is similar to the message passing that Pearl defines (Pearl 1988). If there is a NULL vector from below, but a valid vector from above, then an upper and lower level output may still result if the vector from above is likely enough. If there are non-NULL vectors from both above and below, they both contribute to the winning vector, and the contribution of each vector is proportion to the likelihood of each vector. The computation of the most likely output codebook vector is the influence of "belief" propagated both from above and from below, where a Bayesian decision rule deciding who wins.

Summarizing, the BM has the following components:

- A VQ like codebook that is a compact representation of the probability density function (PDF) of the input vector space
- An output vector table, which has the same number of entries as the VQ table
- A "winner" table entry selection mechanism, which incorporates evidence from below and from above
- A feed forward path for the output vector of the winning codebook entry
- A feed back path which outputs the winning VQ input entry

Learning in a BM is essentially building the code book and the likelihoods (region size) of the codebook entries. Both the input and output vectors are learned so that theoretically, they are equal likely distributed in their spaces. This is very similar to that of configuring a Vector Quantizer or cluster center. Techniques developed for training Vector Quantizers can be used (Gersho and Gray 1992). Kohonen's LVQ and LVQ2 algorithms are also possibilities. Another approach is to store every training vector and estimate the PDF by smoothing this space like Parzen Windows (Specht 1990).

When learning is enabled, for any input vector that does not exactly match a codebook vector, the BM generates incremental changes in the codebook vector positions and weights. In this process, the winning vector are moved closer to the true codebook vector, and the losers are moved slightly farther away. If no winner is within a certain threshold, a new codebook vector is allocated and is generated from the input. There can be continual adaptation, and the dynamic learning will be required in a number of applications. The learning process basically filters noise from the input stream and

creates a maximum entropy encoding in the vector space, which means that the learning should attempt to make all codebook entries more or less equally likely. The output codebook vectors should be allocated roughly equal distance from each other in a sparse, distributed encoding space. In addition, if there is some meaning to the "closeness" in input vector metric, then such a relationship should be retained as much as possible in the output vector space.

Although most learning is unsupervised, a BM can incorporate top down information into the calculation by biasing the V_{Hin} for the top layer of the network. This increases the likelihood of certain outputs when these particular vectors occur in conjunction with corresponding input vectors. In this case, the supervision may contribute to the adjustment of codebook weight *W*. It is also possible to put a simple classifier at the output layer, which is trained by supervision.

Preliminary data indicate that systems with large numbers of small BMs rather than a few large BMs, more closely models biology and has certain structural advantages. We conjecture that systems with large numbers of small BMs are more tolerant of simplifications and approximations in BM implementation. In a hierarchical network of BMs, the BM needs to merge information from multiple lower level BMs. As shown in Figure 6.4, the BM receives two input vectors V_{Lin1} and V_{Lin2} from below. The two input vectors are concatenated to form a single input vector V_{Lin}, with the VQ being done on the combined vector. Likewise, the appropriate subparts are fed back accordingly.

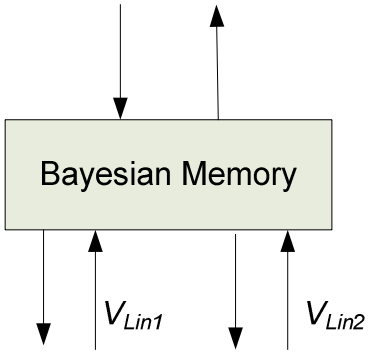

Figure 6.4 A BM receives two inputs from below

In the network of BMs, each BM receives "belief" from above and below BMs through forward and feedback paths. It draws inference on the input, and then converges to a stable state, where it finds the corresponding codebook vector and output its "belief". The "belief" is then passes up to above and down to below BMs. The BM works similar to the standard "node" in a belief network, and the bidirectional information flow is similar to the "belief propagation" defined in (Pearl 1988). A hypothesis is that the most efficient way to implement a BM is via association as shown in the next section. This hypothesis will be the subject of future work.

6.4.2 Bayesian Memory Implementation

We now propose an implementation model of BM based on associative memories. To implement such a BM, some simplifications and approximations are made in our model. For example, we assume the codebook vectors are binary and have a fixed length and that the number of 1s is constant. All vectors have a separate "likelihood" field that is associated with the vector. The NULL vector has zero likelihood. Although the input and output vectors are binary, the internal vectors are not binary so that the best-match computation can be performed at a higher precision.

We believe that one of the most important aspects of a BM is that output is in a distributed data representation. There is reasonable agreement that neural structures represent data in a "distributed" manner, but there is less agreement on the details of this representation. Different models and algorithms distribute their representations in different ways. An important kind of distributed representation is the sparse code (Field 1999), k-WTA codes, incidentally, are usually sparse.

In sparse distributed codes, all nodes have an equal response probability across all representations, but a low response probability for any single representation. The dimensionality of the space is spanned by the input vectors, but is not necessarily reduced in dimension. Sparse distributed coding represents data with a minimum number of active units. Although there are ways to convert data from a distributed representation back to a more traditional representation – the de-sparsification generally is generally complex. A distributed representation of higher dimensional elements can be represented by using

multiple units with some overlap of subspace dimensions. Also, for simplification, only binary sparse vector is considered here.

For the given input vectors from below V_{Lin}, and from above V_{Hin}, a BM needs to find the most likely codebook vectors, that is, $\arg\max_i P(V_{codebook}(i) \mid V_{Lin}, V_{Hin})$. In our implementation, we assume that the input vectors V_{Lin} and V_{Hin} are independent. The likelihood of the input vector is factored into the computation, and the winning input-output codebook entry is the one with the largest "combined" likelihood.

BM provides maximum entropy decoding. In previous chapters, we have discussed that associative memory has the similar characteristics, and the Palm based associative memory approximates the Vector Quantizer, and provides Bayesian classification with equal priors. We use Palm based associative memory as an approximation of a BM. Although Palm network doesn't implement the full belief propagation, and the network doesn't directly represent the conditional probability, the network output itself provides the best representation given the input in MAP sense.

The binary Palm model operates by partitioning the input space into Voronoi regions, with each region centered at a training vector. As discussed earlier, under certain convergent conditions, if all the training vectors are equally distributed, then the network implements Bayesian classification with naïve or "equal" priors. This result also holds for dynamic learning, where the training vectors have different prior probabilities. Arbitrary vector prior allows for weighted Voronoi regions. If a vector's Voronoi region is roughly proportional to the prior probability, then the network retrieves an output vector that has approximately the maximum posterior probability.

The BM is a special associative memory that requires the codebook vectors to have the maximum entropy reduction in the corresponding vector spaces. The Palm model can be regarded as one specific example of BM with only one direction of information flow. Inside a general BM, there is also a feedback path. Therefore, when using a Palm model to simulate a BM, two mappings are needed, with one simulating the forward path and the other simulating the feedback path.

6.4.3 Simulations and Results

The interface of a BM is similar to that of a bidirectional associative memory (BAM) (Kosko 1988), which can accept input vector from below and feedback vector from above network. In this section we discuss a simple implementation of BM network consisting of 3 BMs as shown in Figure 6.5. In the 3-BM network, each BM is implemented by Palm networks.

In Chapter 2, we have discussed the implementation of Palm network using our Csim tool. In the 3-BM network implementation, Csim has been modified so that each BM contains 2 Palm networks, with one mapping in each direction.

To demonstrate the basic principles, the 3-BM network is connected in a basic tree structure. In the 3-BM network, BM1 and BM2 are at the low level of the network, and BM3 is at the high level of the network. The outputs of BM1 and BM2 are fed into BM3. Input vectors V_{Lin1} and V_{Lin2} have the same size (vector length) of L_L, and the output vectors V_{Hout1} and V_{Hout2} have the same size of L_H, with $L_H < L_L$. The input vector V_{Lin3} for BM3 is a concatenation of the output vectors V_{Hout1} and V_{Hout2} with $V_{Lin3} = V_{Hout1} \,||\, V_{Hout2}$, where || signifies concatenation. Therefore, V_{Lin3} has the vector size of $2*L_H$. We assume $V_{Hin3} = V_{Hout3}$, and BM3 works as an auto-associative memory. The BM3 output vector V_{Lout3} consists of into two parts: one feeding back to BM1 as $V_{Lout1,}$ and the other feeding back to BM2 as V_{Lout2}, and $V_{Lout3} = V_{Hin1} \,||\, V_{Hin2}$.

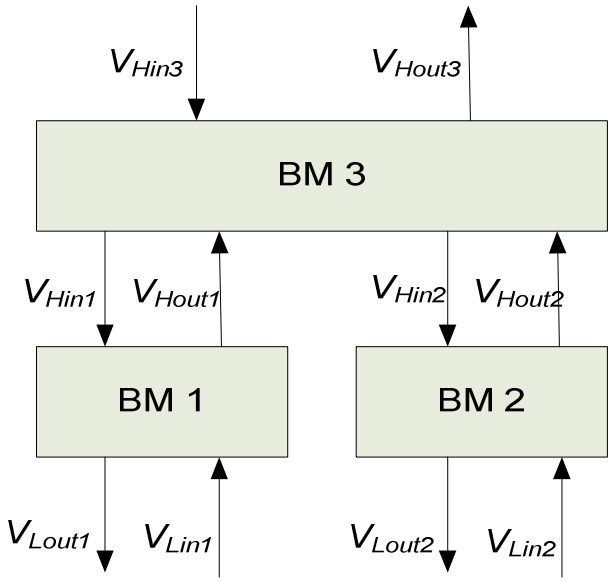

Figure 6.5 A three-BM network

In the simulation presented here, all the vectors are random and sparsely distributed. For BM1 and BM2, we create a list of codebook-vector pairs that are one-to-one mappings of input vector to output vector. For simplicity, all the input and output vectors are uniformly distributed, so the weights *W* associated with each vector are equal. For now, we do not simulate the weight influence that different vector has different weight *W*, as shown in Figure 6.3. With one codebook, two Palm networks are created to simulate the bidirectional path. The number of training vectors in BM1 is the same as in BM2. The training vector in BM1 is concatenated with the training vector in BM2 that has the same index, so they are each a part of one larger vector. The new concatenated vector becomes the training vector in BM3. More complicated concatenation of BM1 and BM2 training vectors is possible. However, as suggested in (Pearl 1988), "the elementary building blocks of human knowledge are not entries of a joint-distribution table. Rather, they are low-order marginal and conditional probabilities defined over small clusters of propositions". We only simulate a small portion of the "joint-distribution table", and the number of training vectors in BM3 is the same as that in BM1 or BM2.

In our simulation, noisy inputs are feed to BM1 and BM2 as V_{Lin1} and V_{Lin2}, and each generates an output vector consisting of V_{Hout1} and V_{Hout2}, which are concatenated to

form V_{Lin3} , which is the input to BM3. The output vector V_{Hout3} of BM3 is then fed back to itself, and generating the output vector V_{Lout3} that forms V_{Hin1} and $V_{Hin2,}$ which are then passed down to BM1 and BM2 respectively. In the simulation, V_{Lout1} and V_{Lout2} are not passed further down to lower BM. In a more than 2 layer BM network, V_{Lout1} and V_{Lout2} are fed to the lower BMs providing the top-down information. With the BM3 feedback, BM1 and BM2 can return the codebook vectors that are "closest" to the given input, taking into consideration the larger set of information that BM3 has about its inputs. We did not simulate the situation where both inputs, one from below and one from above, are available at the same time. In this case, the weight for each codebook vector should be factored into calculation according to Bayes rule. This implementation is left for future research.

The K-WTA method is still used, and a NULL vector may results from an absolute threshold. In deciding which codebook vector wins in a BM, if the number of active nodes in the output vector is more than k, BM will generate a NULL vector, avoiding the noise being passed further down the data path.

Figure 6.6 shows an example of the simulation of our 3-BMs network. In the network, the input codebook vectors for BM1 and BM2 have the same size of $L_L = 80$, and the number of active nodes in the vectors is 6. The output codebook vectors for BM1 and BM2 have the same size of $L_H = 40$, and the number of active nodes in the vectors is 4. There are 10 codebook vector pairs in both BM1 and BM2, and these 10 codebook vectors are concatenated as the codebook vectors for BM3. All the codebook vectors in BM1 and BM2 are random vectors.

When testing the network, the noisy vector is generated by randomly de-activating some of the activate nodes in the codebook vector according to the bit-flip probability, and then randomly activating some of the non-active nodes in the same vector with the same bit-flip probability. By doing this, the number of active nodes in a vector is the same as that in the codebook vector. The "*Performance*" is defined as the correlation between a vector and its corresponding codebook vector. When the *Performance* is 1, the vector is the same as the codebook vector. When a vector is NULL, no *Performance* is calculated. Each *Performance* value shown in Figure 6.6 is calculated by averaging 8 independent experiment results.

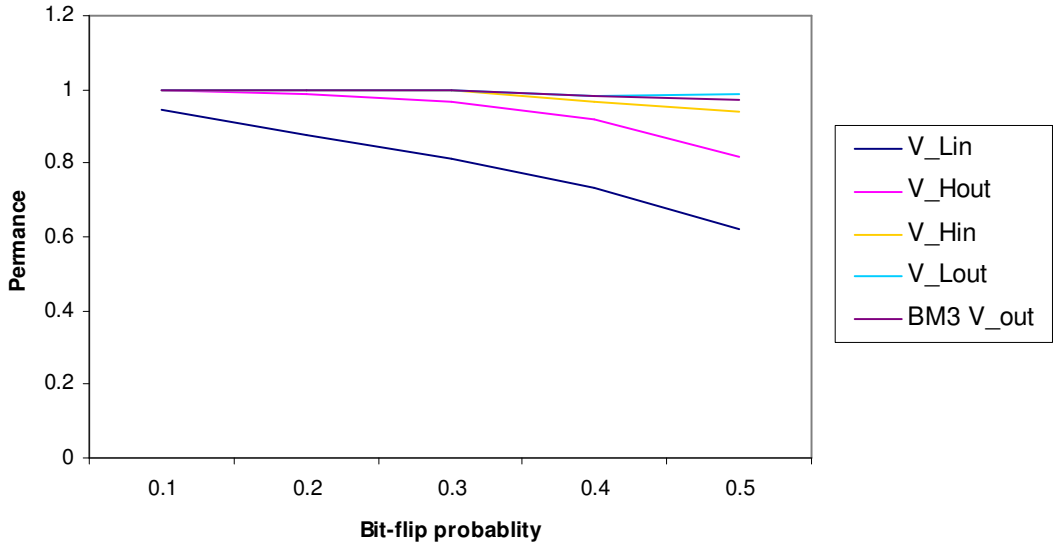

Figure 6.6 Performace of V_{Lin}, V_{Hout}, V_{Hin} and V_{Lout} for BM1 and BM2, and the performance of BM3 output

Figure 6.6 shows the performance of all the interface vectors V_{Lin}, V_{Hout}, V_{Hin} and V_{Lout} of BM1 and BM2, and the performance of BM3 output. BM3 acts as a regular auto-associative memory, its input is the concatenation of V_{Hout1} and V_{Hout2}, and its output is the concatenation of V_{Hin1} and V_{Hin2}. BM3 output performance is comparable to that of V_{Hin1} and V_{Hin2}, just that the vector length is different. BM3 performance represents the performance of the entire 3-BMs network. Vector V_{Lin} is the low level noisy input to BM1 and BM2. As shown in Figure 6.6, the BM1 and BM2 improve the accuracy of output vector V_{Hout} without any feedback from BM3. When BM3 is enabled, the feedback vector V_{Hin} partitioned from V_{Lout3} in BM3 gets better performance than $V_{Hout,}$ which implies that V_{Hin} gets closer to the output codebook vector than V_{Hout}. As the feedback vector V_{Hin} is passed down to BM1 and BM2, even more noise is removed, and V_{Lout} has the best performance, approaching 1 even when the bit-flip probability is 0.5. V_{Lout} is not fed further down to lower network in this simulation. Because of the top-down information passed by BM3, V_{Lout} nearly approaches the original codebook vector.

Although a small codebook vector size is simulated here, and only a very limited network scale is tested, our simulation result shows that a hierarchy of associative networks in a Bayesian Belief Propagation like implementation has significant

performance advantages. We believe that based on this preliminary work, the use of such hierarchical structures will lead to large, scalable systems.

In the simulation, when the bit-flip probability is 0.5, on average, we get one NULL at V_{Lout} for every 10 test inputs. For most bit-flip probabilities that are less than 0.5, no NULL vector is found at V_{Lout}, which indicates that, when the error rate is relatively low at V_{Lin}, the feedback vector V_{Hin} from BM3 provides additional information that is strong enough to "pull" the noisy input, $V_{Lin,}$, closer to its original codebook vector.

In our simulations, we only tested one BM configuration. With the same number of mappings, a different network of BMs can be built, with different number of BMs and each with a different network size. We did not search for preferred configurations, where the number and size of BMs fits best with a particular network size. The number of hierarchy levels and the number of vectors stored in each BM are also related. This correlation shall be an interesting topic in future research.

In our BM model, temporal information is not simulated. A more general BM model should contain the temporal correlation. This is another topic for future research, since most pattern recognition occurs in an environment that has both spatial and temporal data. Such a more general BM model would have far more complex computations. Some simplified versions are being explored, such as the HTM algorithm (George and Jaros 2007).

Chapter 7

Summary and Conclusions

Associative memory is one of the primary components in any cognitive system. This thesis investigates the implementation of some of the computational characteristics of an associative memory that are essential for just basic cognitive operations. We have chosen the Palm network as a foundation model, since it is straightforward to implement, and yet works well. Based on our and other researchers' studies of the Palm network, we propose a Bayesian Memory model that uses Palm as its computation and that can be considered as a building block in large neural based cognitive system.

Palm's network has sparse distributed data representations, large information capacity and fault tolerance to noise in the input data and the network connections. We investigated the feasibility of scaling this network. As a part of this study, we built large networks on both a PC cluster and parallel supercomputers. These simulations demonstrated that the retrieval performance of Palm network is fairly stable for small network sizes. However, for very large networks, performance begins to degrade. Though we did find that executing this algorithm on multiple processors, where a large portion of processing is spent on communication among processors, the computation efficiency was still very good.

To apply the Palm model in real applications, one needs to understand the basic functionality of the model and its limitations. We analyzed the retrieval performance of the Palm model, and proposed a functional analogy to Voronoi classification and subsequently to Bayesian classification. When the input vector is within the attractor basin of a training vector, the associative network can return the most likely training vector in a Bayesian sense. Furthermore, we introduce the concept of weighted Voronoi regions and then define a version that learned incrementally during operation. In such

dynamic learning the network approximates weighted Voronoi classification. In addition, dynamic learning approximates the influence coefficient of each training vector and allows the representation of prior probabilities in the Bayesian inference process. The Bayesian inference approximation provides a straightforward and useful theoretical foundation for the operation of the Palm model and provides an analytical basis for the application of associative networks to real world problems.

Dynamic adaptation or learning during operation is also important. Reinforcement learning, as demonstrated here, enables a cognitive system to learn through its interaction with the outside world and forms an action mapping of the environment. We developed an associative network that learned by reinforcement, RLAM. We demonstrated that system learns complex mappings quickly. In RLAM, there are two networks: one maps the input states to values (the critic); and another that maps the states to actions (the actor). With k-WTA, the network has the ability to handle non-linear maps. Although there are limitations on arbitrary approximation, RLAM can learn complex functions if enough training patterns are presented to it. In the early stages of training when only a small subset of training patterns are sampled, there are a few large regions that are crude approximation to the final mapping, with increasing experience, a more complex structure that represents the final solution then evolves.

The binary Palm model we used for our scaling experiments operated over spatial data only, ignoring the temporal domain. We built a spiking version of the Palm model to add temporal information and study the Hebbian learning between the spiking neurons. These simulations showed that the weight matrix formed by Hebbian learning in spiking neurons is very similar to the weight matrix calculated in binary model, and, even with a spiking representation, the model is quite tolerant to temporal noise (jitter) in the input. Since in most real applications, there is as much information in the temporal domain as in the spatial domain creating hierarchical networks that learn through time as well as space constitutes an important topic for follow on research.

Based on Voronoi classification, we proposed a Bayesian Memory model based on the Palm network that has the potential to scale to very large sizes. The lack of scaling is a major limitation of most artificial neural network algorithms, and yet it is an absolute requirement for solving complex problems in Intelligent Computing. The Bayesian

Memory extracts the most important information from its vector spaces, provides maximum entropy reduction to the data, and then generates codebook vectors that map from input space to output space. The BM receives inputs from above and below through forward and feedback paths. It draws inference on the inputs, and then converges to a stable state, where it finds the corresponding codebook vectors. The BM then outputs its "belief", and passes it up to above and down to below BMs.

Under convergence conditions, the Voronoi classification characteristic of the basic Palm auto-associative memory mimics a single BM node. Using an auto-associative memory as an approximation to a Bayesian Memory node, we built a simple 3-BM network and simulated its behavior. The simulation demonstrated that, with the higher level BM providing top-down information, the lower level BMs can improve their performance by converging to their stable states more easily. We believe that based on these results, using an associative memory for each node in a hierarchical Bayesian Memory, has significant potential for creating large, scalable associative networks. This then is a promising direction for future research.

In the work presented here, we mostly used random data in our simulations. In real applications, there is significant information embedded in the structures of data. It will be interesting to study how the real data can be utilized and applied in a more general BM model. Likewise we did not attempt to capture temporal information in our model. Obviously a more general BM model will need to use combined spatial and temporal information. However, this does lead to more complex computation in BM.

The convergence of BMs under dynamic inputs is a complicated process. There have been a number of research directions regarding the network convergence. Myllymaki (Myllymaki 1999) presents a method mapping a given Bayesian network to a Boltzmann machine model where a converged state can be mapped back to a maximum a posteriori probability state in the Bayesian network. Free energy approximation is another approach (Yedidia, Freeman et al. 2006) that can be used to construct the Bayesian Propagation (BP) algorithm such that the BP algorithm always correspond to the stationary points of the region graph approximating the free energy. One important research topic is to study the convergence conditions of BMs in dynamic inputs, the

relationship between network size and the number of BMs, as well as the number of hierarchy levels and the number of vectors stored in each BM.

In addition to new results, one of the most important accomplishments of any doctoral research is the generation of new ideas for follow on work. We believe that this work has resulted in important new results in the construction and operation of associative networks, it has also contributed significant numbers of new ideas and promising areas for follow on research. The hardware architecture and implementation of distributed associative networks and BM (Zaveri and Hammerstrom; Gao 2008) has been investigated in our research group.

Bayesian inference has become an increasingly important area for research and application in the field of computational neuroscience. Animals learn from their surroundings and seem to do a kind of probabilistic inference from learned knowledge acquired through their interaction with the environment. Bayesian networks provide a general framework for the probabilistic inference, and the major contribution to this is the work of Judea Pearl (Pearl 1988). Bayesian Belief Propagation algorithm also provides the basic learning rule for Numenta's HTM (George and Hawkins 2004; George and Hawkins 2005; Hawkins and George 2006; George and Jaros 2007). Our Bayesian Memory model is based on HTM like structure. The Bayesian inference theory in associative memory and the belief propagation provide the general communication method between BMs. Unlike the traditional Bayesian networks where the probability information is specifically stored, we hypothesize that the BM can use the associative mappings to do inference, and the probability information is intrinsically embedded in the vector representation.

Bibliography

Abbott, L. F. and S. B. Nelson (2000). "Synaptic Plasticity: Taming the Beast." *Nature Neuroscience* 3 Supplement: 1178-1183.

Amari, S.-I. (1989). "Characteristics of Sparsely Encoded Associative Memory." *Neural Networks* 2: 451-457.

Amari, S.-I. and K. Maginu (1988). "Statistical Neurodynamics of Associative Memory." *Neural Networks* 1: 63-73.

Andersen, R. (1995). "Encoding of Intention and Spatial Location in the Posterior Parietal Cortex." *Cerebral Cortex* 5(5): 457-469.

Anderson, B. (1999). "Commentary : Ringo, Doty, Demeter and Simard, Cerebral Cortex 1994;4:331-343: A Proof of the Need for the Spatial Clustering of Interneuronal Connections to Enhance Cortical Computation." *Cerebral Cortex* 9: 2-3.

Anderson, J. A. (1993). The BSB Model: A Simple Nonlinear Autoassociative Network. *Associative Neural Memories: Theory and Implementation*. M. H. Hassoun, Oxford University Press.

Anderson, J. A., P. Allopenna, et al. (2007). Programming a Parallel Computer: The Ersatz Brain Project. *Studies in Computational Intelligence*, Springer Berlin / Heidelberg: 61-98.

Anderson, J. A., J. W. Silverstein, et al. (1977). "Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of a Neural Model." *Psychological Review* 84: 413-451.

Anderson, J. A. and J. P. Sutton (1995). "A Network of Networks: Computation and Neurobiology." *Proceedings of World Congress Neural Networks* 1: 561-568.

Baldassarre, G. (2002). "A Modular Neural-Network Model of the Basal Ganglia's Role in Learning and Selecting Motor Behaviours." *Cognitive Systems Research* 3: 5-13.

Bosch, H. and F. J. Kurfess (1998). "Information Storage Capacity of Incompletely Connected Associative Memories." *Neural Networks* 11: 869-876.

Braitenberg, V. (1978). Cortical Architectonics, General and Areal. *Architechtonics of the Cerebral Cortex*. M. Brazier and H. Petsche. Raven, New York.

Braitenberg, V. (2001). "Brain Size and Number of Neurons: An Exercise in Synthetic Neuroanatomy." *Journal of Computational Neuroscience* 10: 71-77.

Braitenberg, V. and A. Schuz (1998). *Cortex: Statistics and Geometry of Neuronal Connectivity*, Springer.

Buckingham, J. and D. Willshaw (1992). "Performance Characteristics of the Associative Net." *Network: Computation in Neural Systems* 3: 407-414.

Buckingham, J. and D. Willshaw (1993). "On Setting Unit Threshold in an Incompletely Connected Associative Net." *Network: Computation in Neural Systems* 4: 441-459.

Buneo, C. A., M. R. Jarvis, et al. (2002). "Direct Visuomotor Transformations for Reaching." *Nature* 416: 632-636.

Coultrip, R., R. Granger, et al. (1992). "A Cortical Model of Winner-Take-All Competition Via Lateral Inhibition." *Neural Networks* 5: 47-54.

Coultrip, R. L. and R. H. Granger (1994). "Sparse Random Networks with LTP Learning Rules Approximate Bayes Classifiers Via Parzen's Method." *Neural Networks* 7(3): 463-476.

Duda, R. O., P. E. Hart, et al. (2001). *Pattern Classification*, Wiley-Interscience.

Field, D. J. (1999). What Is the Goal of Sensory Coding? *Unsupervised Learning*. G. Hinton and T. J. Sejnowski. Cambridge, MA, MIT Press: 101-143.

Gao, C. (2008). Hardware Architectures and Implementations for Hierarchical Distributed Neural Associative Memories. *Electrical and Computer Engineering*, Portland State University.

George, D. (2008). How the Brain Might Work: A Hierarchical and Temporal Model for Learning and Recognition. *Electrical Engineering*, Stanford University.

George, D. and J. Hawkins (2004). Invariant Pattern Recognition Using Bayesian Inference on Hierarchical Sequences, Stanford University.

George, D. and J. Hawkins (2005). "A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex." *Proceedings of the International Joint Conference on Neural Networks* 3: 1812-1817.

George, D. and B. Jaros (2007). The HTM Learning Algorithms, Numenta Inc.

Gersho, A. and R. M. Gray (1992). *Vector Quantization and Signal Compression*. New York, Springer.

Gerstner, W. (1990). "Associative Memory in a Network of Biological Neurons." *Advances in Neural Information Processing Systems* 3: 84-90.

Gerstner, W. and J. L. v. Hemmen (1992). "Associative Memory in a Network of 'Spiking' Neurons." *Network* 3: 139-164.

Gerstner, W. and W. M. Kistler (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. New York, Cambridge University Press.

Gibson, W. G. and J. Robinson (1992). "Statistical Analysis of the Dynamics of a Sparse Associative Memory." *Neural Networks* 5(4): 645-661.

Glimcher, P. W. (2003). *Decisions, Uncertainty, and the Brain: The Science of Neuroeconomics*, The MIT Press.

Glimcher, P. W., M. C. Dorris, et al. (2005). "Physiological Utility Theory and the Neuroeconomics of Choice." *Games and Economic Behavior* 52: 213-256.

Goldberg, M. E., C. L. Colby, et al. (1990). "Representation of Visuomotor Space in the Parietal Lobe of the Monkey." *Cold Spring Harbor Symposia in Quantitative Biology* LV: 729-739.

Goodman, S. J. and R. A. Andersen (1990). "Algorithm Programmed by a Neural Network Model for Coordinate Transformation." *Proceedings of the International Joint Conference on Neural Networks* 2: 381-386.

Graham, B. and D. Willshaw (1994). "Capacity and Information Efficiency of a Brain-Like Associative Net." *Advances in Neural Information Processing Systems* 7: 513-520.

Graham, B. and D. Willshaw (1995). "Improving Recall from an Associative Memory." *Biological Cybernetics* 72: 337-346.

Graham, B. and D. Willshaw (1997a). "Capacity and Information Efficiency of the Associative Net." *Network: Computation in Neural Systems* 8: 35-54.

Graham, B. and D. Willshaw (1997b). "A Model of Clipped Hebbian Learning in a Neocortical Pyramidal Cell." *Proceedings of the International Conference on Artificial Neural Networks*: 151-156.

Graham, B. and D. J. Willshaw (1999). "Probabilistic Synaptic Transmission in the Associative Net." *Neural Computation* 11(1): 117-137.

Hasson, U., Y. Nir, et al. (2004). "Intersubject Synchronization of Cortical Activity During Natural Vision." *Science* 303(5664): 1634-1640.

Hassoun, M. H., Ed. (1993). Associative Neural Memories: Theory and Implementation. New York, Oxford University Press.

Hassoun, M. H. and P. B. Watta (1996). "The Hamming Associative Memory and Its Relation to the Exponential Capacity Dam." *Proceedings of the IEEE International Conference on Neural Networks* 1: 583-587.

Hawkins, J. and S. Blakeslee (2004). *On Intelligence*. New York, Times Books.

Hawkins, J. and D. George (2006). Hierarchical Temporal Memory: Concepts, Theory, and Terminology, Numenta Inc.

Haykin, S. (2001). *Communication Systems*. New York, Wiley.

Hecht-Nielsen, R. (1987). "Kolmogorov's Mapping Neural Network Existence Theorem." *Proceedings of the IEEE International Conference on Neural Networks* III: 11-14.

Hecht-Nielsen, R. (1998). "A Theory of the Cerebral Cortex." *Proceedings of the International Conference on Neural Information Processing* 3: 1459-1464.

Hecht-Nielsen, R. (1999). "Cortronic Neural Networks." *International Joint Conference on Neural Networks*(Tutorial).

Hecht-Nielsen, R. (2003a). A Theory of Cerebral Cortex, University of California, San Diego, Institute for Neural Computation.

Hecht-Nielsen, R. (2003b). A Theory of Thalamocortex. *Computational Models for Neuroscience*. R. Hecht-Nieldsen and T. Mckenna, London: Springer-Verlag: 85-124.

Henson, R. N. A. and D. Willshaw (1995). Short-Term Associative Memory. *Proceedings of the INNS World Congress on Neural Networks*, Washington D. C.

Hopfield, J. J. (1982). "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." *Proceedings of National Academy of Sciences of the United States of America* 79(8): 2554-2558.

Hopfield, J. J. (1995). "Pattern Recognition Computation Using Action Potential Timing for Stimulus Representation." *Nature* 376(July): 33-36.

Hopfield, J. J. and C. D. Brody (2000). "What Is a Moment? "Cortical" Sensory Integration over a Brief Interval." *Proceedings of National Academy of Sciences of the United States of America* 97(25): 13919-13924.

Hopfield, J. J. and C. D. Brody (2001). "What Is a Moment? Transient Synchrony as a Collective Mechanism for Spatiotemporal Integration." *Proceedings of National Academy of Sciences of the United States of America* 98(3): 1282-1287.

Hornik, K., M. Stinchcombe, et al. (1989). "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2: 359-366.

Houk, C. J. (1995). Information Processing in Modular Circuits Linking Basal Ganglia and Cerebral Cortex. *Models of Information Processing in the Basal Ganglia*. C. J. Houk, L. J. Davis and G. D. Beiser, The MIT Press.

Johansson, C. (2004). Towards Cortex Isomorphic Attractor Neural Networks. *Numerical Analysis and Computer Science*. Stockholm, Sweden, Royal Institute of Technology.

Johansson, C., M. Rehn, et al. (2006). "Attractor Neural Networks with Patchy Connectivity." *Neurocomputing* 69: 627-633.

Johansson, C., A. Sandberg, et al. (2002). "Attractor Neural Networks with Hypercolumns." *Lecture Notes in Computer Science, Proceedings of the International Conference on Artificial Neural Networks* 2415: 192-197.

Kanerva, P. (1988). *Sparse Distributed Memory*. London, The MIT Press.

Knoblauch, A. and G. Palm (2001a). "Pattern Separation and Synchronization in Spiking Associative Memories." *Neural Networks* 14: 763-780.

Knoblauch, A. and G. Palm (2001b). "Spiking Associative Memory and Scene Segmentation by Synchronization of Cortical Activity." *Lecture Notes in Computer Science, Emergent Neural Computational Architectures Based on Neuroscience - Towards Neuroscience-Inspired Computing* 2036: 407-427.

Kohonen, T. (1982). "Self-Organized Formation of Topologically Correct Feature Maps." *Biological Cybernetics* 43: 59-69.

Kohonen, T. (1989). *Self-Organization and Associative Memory*. New York, Springer.

Kosko, B. (1988). "Bidirectional Associative Memories." *IEEE Transactions on Systems, Man, Cybernetics* 18(1): 49-60.

Lansner, A. and O. Ekeberg (1989). "A One-Layer Feedback Artificial Neural Network with a Bayesian Learning Rule." *International Journal of Neural Systems* 1(1): 77-87.

Laughlin, S. B. and T. J. Sejnowski (2003). "Communication in Neuronal Networks." *Science* 301(5641): 1870-1874.

Lengyel, M. and P. Dayan (2004). "Rate- and Phase-Coded Autoassociative Memory." *Advances in Neural Information Processing Systems* 17: 769-776.

Luk, C. H., C. Gao, et al. (2004). "Biologically Inspired Enhanced Vision System (EVS) for Aircraft Landing Guidance." *Proceedings of the International Joint Conference on Neural Networks* 3: 1751-1756.

Maass, W. (1997). "Fast Sigmoidal Networks Via Spiking Neurons." *Neural Computation* 9: 279-304.

Maass, W. (1999). "Neural Computation with Winner-Take-All as the Only Nonlinear Operation." *Advances in Neural Information Processing Systems*: 293-299.

Maass, W. (2000). "On the Computational Power of Winner-Take-All." *Neural Computation* 12: 2519-2535.

Maass, W. and C. M. Bishop (1999). *Pulsed Neural Networks*. Cambridge MA, MIT Press.

Maass, W. and T. Natchlager (1998). Associative Memory with Networks of Spiking Neurons in Temporal Coding. *Neuromorphic Systems: Engineering Silicon from Neurobiology*. L. S. Smith and A. Hamilton: 21-32.

Maass, W. and T. Natschlager (1997). "Networks of Spiking Neurons Can Emulate Arbitrary Hopfield Nets in Temporal Coding." *Network: Computation in Neural Systems* 8(4): 355-372.

MacKay, D. (1991). Maximum Entropy Connections: Neural Networks. *Maximum Entropy and Bayesian Methods*. W. T. Grandy and L. Schick. Boston, MA, Kluwer: 237-244.

MacKay, D. J. C. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge, Cambridge University Press.

Mazzoni, P., R. A. Andersen, et al. (1991). "A More Biologically Plausible Learning Rule for Neural Networks." *Proceedings of the National Academy of Sciences of the United States of America* 88(May): 4433-4437.

Myllymaki, P. (1999). "Massively Parallel Probabilistic Reasoning with Boltzman Machines." *Applied Intelligence* 11: 31-44.

O'Kane, D. and A. Treves (1992a). "Short- and Long-Range Connections in Autoassociative Memory." *Journal of Physics A: Mathematical and General* 25: 5055-5069.

O'Kane, D. and A. Treves (1992b). "Why the Simplest Notion of Neocortex as an Autoassociative Memory Would Not Work." *Network* 3: 379-384.

Palm, G. (1980). "On Associative Memory." *Biological Cybernetics*: 19-31.

Palm, G. (1988). On the Asymptotic Information Storage Capacity of Neural Networks. *Proceedings of the Nato Advanced Research Workshop on Neural Computers*. R. Eckmiller and C. v. d. Malsburg, Springer-Verlag New York, Inc.: 271-280.

Palm, G., F. Schwenker, et al. (1997). Neural Associative Memories. *Associative Processing and Processors*. E. A. K. a. C. C. Weems. Los Alamitos, CA, IEEE Computer Society: 307-326.

Palm, G. and F. T. Sommer (1992). "Information Capacity in Recurrent Mcculloch-Pitts Networks with Sparsely Coded Memory States." *Network* 3: 177-186.

Palm, G. and F. T. Sommer (1995). Associative Data Storage and Retrieval in Neural Networks. *Models of Neural Networks Iii*: 79-118.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Francisco: Morgan Kaufmann.

Pinsky, P. R. and J. Rinzel (1994). "Intrinsic and Network Rhythmogenesis in a Reduced Traub Model for Ca3 Neurons." *Journal of Computational Neuroscience* 1: 39-60.

Pouget, A. and T. J. Sejnowski (1997). "Spatial Transformations in the Parietal Cortex Using Basis Functions." *Journal of Cognitive Neuroscience* 9: 222-237.

Rao, R. P. N. (2005). "Hierarchical Bayesian Inference in Networks of Spiking Neurons." *Advances in Neural Information Processing Systems* 17: 1113-1120.

Ringo, J., R. Doty, et al. (1994). "Time Is of the Essence: A Conjecture That Hemispheric Specialization Arises from Interhemispheric Conduction Delay." *Cerebral Cortex* 4: 331-343.

Ruf, B. and M. Schmitt (1997). "Unsupervised Learning in Networks of Spiking Neurons Using Temporal Coding." *Proceedings of the International Conference on Artificial Neural Networks*: 361-366.

Sandberg, A., A. Lansner, et al. (1999). An Incremental Bayesian Learning Rule. Stockholm, Sweden, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology.

Sandberg, A., A. Lansner, et al. (2000). "A Palimpsest Memory Based on an Incremental Bayesian Learning Rule." *Neurocomputing* 32-33: 987-994.

Sandberg, A., A. Lansner, et al. (2002). "A Bayesian Attractor Network with Incremental Learning." *Network: Computation in neural system* 13: 179-194.

Schwenker, F., F. T. Sommer, et al. (1996). "Iterative Retrieval of Sparsely Coded Associative Memory Patterns." *Neural Networks* 9(3): 445-455.

Snir, M., S. Otto, et al. (1996). *MPI: The Complete Reference (Scientific and Engineering Computation Series)*. London, The MIT Press.

Sommer, F. T. and P. Dayan (1998). "Bayesian Retrieval in Associative Memories with Storage Errors." *IEEE Transactions on Neural Networks* 9(4): 705-713.

Sommer, F. T. and G. Palm (1998). "Bidirectional Retrieval from Associative Memory." *Advances in Neural Information Processing Systems* 10: 675-681.

Sommer, F. T. and T. Wennekers (2001). "Associative Memory in Networks of Spiking Neurons." *Neural Networks* 14: 825-834.

Sommer, F. T., T. Wennekers, et al. (1998). "Bidirectional Completion of Cell Assemblies in the Cortex." *Computational Neuroscience: Trends in Research 1998*.

Specht, D. F. (1990). "Probabilistic Neural Networks." *Neural Networks* 3: 110-118.

Steinbuch, K. and U. A. W. Piske (1963). "Learning Matrices and Their Applications." *IEEE Transactions on Electronic Computers* EC-12: 846-862.

Thrun, S. and A. Schwartz (1993). "Issues in Using Function Approximation for Reinforcement Learning." *Proceedings of the Fourth Connectionist Models Summer School*: 255-263.

Watkins, C. and P. Dayan (1992). "Q-Learning - Technical Note." *Machine Learning* 8: 279-292.

Wennekers, T., F. T. Sommer, et al. (1995). Iterative Retrieval in Associative Memories by Threshold Control of Different Neural Models. *Supercomputing in Brain Research: From Tomography to Neural Networks*. H. Herrmann, D. Wolf and E. Poppel, Singapore: world scientific: 301-319.

Willshaw, D. and P. Dayan (1990). "Optimal Plasticity from Matrix Memories: What Goes up Must Come Down." *Neural Computation* 2: 85-93.

Willshaw, D. J., O. P. Buneman, et al. (1969). "Non-Holographic Associative Memory." *Nature* 222: 960-962.

Yedidia, J. S., W. T. Freeman, et al. (2006). "Constructing Free Energy Approximations and Generalized Belief Propagation." *IEEE Transactions on Information Theory* 51(7): 2282-2313.

Zaveri, M. and D. Hammerstrom "Bayesian Memory Based Cortical Model onto CMOS and CMOL – an Architecture Assessment Methodology." *Submitted for Publication*.

Zhang, K. and T. J. Sejnowski (2000). "A Universal Scaling Law between Gray Matter and White Matter of Cerebral Cortex." *Proceedings of the National Academy of Sciences of the United States of America* 97(10): 5621-5626.

Zhu, S. and D. Hammerstrom (2002). "Simulations of Associative Neural Networks." *Proceedings of the International Conference on Neural Information Processing* 4: 1639-1643.

Zhu, S. and D. Hammerstrom (2003). "Reinforcement Learning in Associative Memory." *Proceedings of the International Joint Conference on Neural Networks* 2: 1346-1350.

Zipser, D. and R. A. Andersen (1988). "A Back-Propagation Programmed Network That Simulates Response Properties of a Subset of Posterior Parietal Neurons." *Nature* 331(25): 679-684.