Recurrent Neural Network for Optimization with Application to Computer Vision

By

Cheung Kwok-Wai

（張國威）

A Master Thesis

Submitted in partial fulfilment of the requirements

for the degree of

Master of Philosophy

in

Department of Electronic Engineering

The Chinese University of Hong Kong

June 1993

To the Cherished Memory

of

my father

Abstract

Applying recurrent neural network for problem-solving in past few years fascinated a lot of expertises in different domains from engineering to financing due to its simplicity for use, robustness, massive parallelism, etc. As recurrent neural network is considered by the domain expertises as a tool for solving problems within their domains, it's very often being modified to suit their own needs and thus variations of recurrent neural network existed in the literature are really multifarious. However, there are lack of research efforts to generalize the different cases of applications and unify them into a more general problem-solving technique. So, our project is working in this direction. The recurrent neural network model interested is Hopfield network. First, different versions of Hopfield network, a very popular recurrent neural network model, are studied, including a generalized Hopfield network model, whose connections are of nth order rather than 1st order for original Hopfield network. Also, the procedure of utilizing generalized Hopfield network for solving a particular problem is discussed, from problem formulation, neural network formulation to network convergence assurance.

After the neural network formulation step, the computational/optimization ability of seeking an optimal solution for neural network is investigated. Since, Hopfield network and further generalized Hopfield network are derived based on gradient method and thus are just gradient system, they can only give a sub-optimal solution. If the particular problem has many local minimum points within the problem domain, the solution's quality will be greatly limited. Incorporating tunneling algorithm - a global optimization technique, into the Hopfield network is proposed. As a result, a novel neural network model, called Tunneling network is derived. It has the capability to seek the global minimum state of the problem mapped to the network without *a prior* knowledge to the possible solutions of the problem, or in other words, it's a blind-searching method. Two versions of Tunneling network are suggested with different means to do the energy barrier tunneling. Through simulation, it's shown that Tunneling network is better than simulated annealing in both efficiency and effectiveness. For further performance enhancement, a network decomposition scheme is found to be able to

achieve the goal by "tunneling" just a small-group of neurons at a time. Furthermore, unlike simulated annealing, tunneling network does not require the determination of a cooling schedule for more effective performance thus is a much more handy global minimization tool for plugging into different applications.

For practical applications, we choose computer vision as the application domain. Recurrent neural network for both gaussian filtering and boundary detection are successfully derived. For the former application, the impulse response of the derived network is found to be very similar to the ideal gaussian shape and by adjusting a parameter embedded in the network's connection, different scale filtering can be achieved. Analogous to setting boundary value condition for many image processing algorithms, the assignment of network's connections in the image boundary required extra attention or error will arise in the boundary and propagate towards the interior part of the network. Formula are derived to ease the assignment task. For the second application, we adopted the problem formulation of Active Contour Model - Snake and used our novel Tunneling network to derive a recurrent neural network for boundary detection. We've applied the derived network to both synthesized and real image to detect boundaries. Due to its global minimum seeking ability, it's found to have much better performance than some existing implementations of "Snake" using local minimization technique.

The reasons for choosing the two applications are of two folds. Firstly, complete investigation of the recurrent neural network alternative for the two problems have not yet appeared in the literature. Secondly, the two problems can be considered as the representatives of two large categories of computer vision problems. So, it's hoped that our neural network solution provided for the two specific applications can be generalized to a wider range of applications.

Acknowledgements

It's my pleasure to express my gratitude to my supervisor, Dr. T. Lee for his novel idea on Tunneling network, his continual encouragement and appreciation throughout the course of this research work, as well as his useful and critical comments in the preparation of this thesis. Also, my thanks are due to Prof. R. Chin and his student, Mr. K. F. Lai for their invaluable discussions and ideas on applications in Computer Vision; and to the colleagues in our laboratory for making my research life to be an enjoyable and memorable one; and to Mr. D. W. Chan, Mr. H. K. Yau and Mr. C. M. Fung for providing a reliable system support; and to the Croucher Foundation for their financial support on my M.Phil. study in the Chinese University of Hong Kong.

Besides, I wish to express thanks to lots of my brothers and sisters in Christ, especially, Roy Li, Ian Cheung, Joanna Chung, Stephen Chan and the lively kids in Timothy fellowship for their support and prayers; and to my dearest Joey for her patient understanding during my thesis writing; and finally to my beloved mother for cherishing me throughout the course of my whole education.

Table of Contents

Chapter 1 Introduction

1.1 Programmed computing vs. neurocomputing....1-1

1.2 Development of neural networks - feedforward and feedback models....1-2

1.3 State of art of applying recurrent neural network towards computer vision problem....1-3

1.4 Objective of the Research....1-6

1.5 Plan of the thesis....1-7

Chapter 2 Background

2.1 Short history on development of Hopfield-like neural network....2-1

2.2 Hopfield network model....2-3

2.2.1 Neuron's transfer function....2-3

2.2.2 Updating sequence....2-6

2.3 Hopfield energy function and network convergence properties....2-7

2.4 Generalized Hopfield network....2-13

2.4.1 Network order and generalized Hopfield network....2-13

2.4.2 Associated energy function and network convergence property....2-13

2.4.3 Hardware implementation consideration....2-15

Chapter 3 Recurrent neural network for optimization

3.1 Mapping to Neural Network formulation....3-1

3.2 Network stability verse Self-reinforcement....3-5

3.2.1 Quadratic problem and Hopfield network....3-6

3.2.2 Higher-order case and reshaping strategy....3-8

3.2.3 Numerical Example....3-10

3.3 Local minimum limitation and existing solutions in the literature....3-12

3.3.1 Simulated Annealing....3-13

3.3.2 Mean Field Annealing....3-15

3.3.3 Adaptively changing neural network....3-16
3.3.4 Correcting Current Method....3-16
3.4 Conclusions....3-17

Chapter 4 A Novel Neural Network for Global Optimization - Tunneling Network

4.1 Tunneling Algorithm....4-1
4.1.1 Description of Tunneling Algorithm....4-1
4.1.2 Tunneling Phase....4-2
4.2 A Neural Network with tunneling capability --- Tunneling network....4-8
4.2.1 Network Specifications....4-8
4.2.2 Tunneling function for Hopfield network and the corresponding updating rule....4-9
4.3 Tunneling network stability and global convergence property....4-12
4.3.1 Tunneling network stability....4-12
4.3.2 Global convergence property....4-15
4.3.2.1 Markov chain model for Hopfield network....4-15
4.3.2.2 Classification of the Hopfield markov chain....4-16
4.3.2.3 Markov chain model for tunneling network and its convergence towards global minimum....4-18
4.3.3 Variation of pole strength and its effect....4-20
4.3.3.1 Energy Profile analysis....4-21
4.3.3.2 Size of attractive basin and pole strength required....4-24
4.3.3.3 A new type of pole eases the implementation problem....4-30
4.4 Simulation Results and Performance comparison....4-31
4.4.1 Simulation Experiments....4-32
4.4.2 Simulation Results and Discussions....4-37
4.4.2.1 Comparisons on optimal path obtained and the convergence rate....4-37
4.4.2.2 On decomposition of Tunneling network....4-38

4.5 Suggested hardware implementation of Tunneling network 4-48
4.5.1 Tunneling network hardware implementation 4-48
4.5.2 Alternative implementation theory 4-52
4.6 Conclusions 4-54

Chapter 5 Recurrent Neural Network for Gaussian Filtering
5.1 Introduction 5-1
5.1.1 Silicon Retina 5-3
5.1.2 An Active Resistor Network for Gaussian Filtering of Image 5-5
5.1.3 Motivations of using recurrent neural network 5-7
5.1.4 Difference between the active resistor network model and recurrent neural network model for gaussian filtering 5-8
5.2 From Problem formulation to Neural Network formulation 5-9
5.2.1 One Dimensional Case 5-9
5.2.2 Two Dimensional Case 5-13
5.3 Simulation Results and Discussions 5-14
5.3.1 Spatial impulse response of the 1-D network 5-14
5.3.2 Filtering property of the 1-D network 5-14
5.3.3 Spatial impulse response of the 2-D network and some filtering results 5-15
5.4 Conclusions 5-16

Chapter 6 Recurrent Neural Network for Boundary Detection
6.1 Introduction 6-1
6.2 From Problem formulation to Neural Network formulation 6-3
6.2.1 Problem Formulation 6-3
6.2.2 Recurrent Neural Network Model used 6-4
6.2.3 Neural Network formulation 6-5
6.3 Simulation Results and Discussions 6-7
6.3.1 Feasibility study and Performance comparison 6-7

6.3.2 Smoothing and Boundary Detection 6-9
6.3.3 Convergence improvement by network decomposition 6-10
6.3.4 Hardware implementation consideration 6-10
6.4 Conclusions 6-11

Chapter 7 Conclusions and Future Researches
7.1 Contributions and Conclusions 7-1
7.2 Limitations and Suggested Future Researches 7-3

References R-1

Appendix I The assignment of the boundary connection of 2-D recurrent neural network for gaussian filtering A1-1
Appendix II Formula for connection weight assignment of 2-D recurrent neural network for gaussian filtering and the proof on symmetric property A2-1
Appendix III Details on reshaping strategy A3-1

Chapter 1 Introduction

1.1 Programmed computing vs. neurocomputing [Nie90]

Today, the speed of digital sequential computer has come to an incredible status that nobody in ten or twenty years ago could predict. We are now talking of computational speed in MIPS (**M**illion **I**nstruction **P**er **S**econd) and MFLOPS (**M**illion **F**loating point **P**er **S**econd). With such a computer, a lot of physical systems can be simulated for analysis, which could trigger further development in technology. Also, a lot of stuffs required intensive computation can now be done in a split of second, which makes a lot of things that only happened in scientists' dream in the past, but are now possible in our daily life, eg. weather forecasting. All these applications are based on the computational paradigm of "*programmed computing*", introduced by von Neumann long long ago. Instructions have to be executed one after another and the computations are governed by *algorithms* and *rules*. This intrinsic characteristic of the paradigm, however, also pin-points its bottleneck.

Developing intelligent system using machines based on programmed computing has quite a long history. However, with so many powerful machines in our toolbox, the intelligence system developed so far is still a long way lagging behind in lots of aspects from a little kid, who may even be ignorant of simple addition and subtraction. Visual perception is a good example. A little kid can recognise accurately and robustly who his/her mother is among the women in the whole world instantly while a computer vision system supported by our MFLOPS computers may take some time to just have the concept of correctly recognising cubes, cylinders in a clean enough background. This discrepancy is mainly due to the completely different computational paradigm of our mind.

Neuroscientists told us that our neural system is a collection of tremendous amount of neural cells called neurons and interconnected in a complicated manner. Based on experimental results, they created functional concepts and models for the system. Speculated from these models, a very different computational paradigm is developed and called *neurocomputing*. The computation is mainly done by *transformation*. For a more formal definition, we use the one given by Hecht Nielsen [Nie90]. *Neurocomputing is the technology*

discipline concerned with parallel, distributed, adaptive information processing systems that develop information processing capabilities in response to expose to an information environment. Nielsen's neurocomputing definition contains no biological or neurophysiological terms. Thus, neurocomputing is considered to be a very broad term and in fact as a general description of Multiple Instruction Multiple Data (MIMD) parallel processing architecture. And *neural networks* are the primary information processing structure of interest in neurocomputing due to its possibility in implementation and its substantial information processing capability.

An artificial neural network consists of many simple processing elements (called *neurons* throughout the thesis), which can carry out localized information processing operations characterised by their transfer functions, or sometimes called activation functions. Each neuron is connected to others via unidirectional signal channels called *connections*. Different connection patterns of an artificial neural network possess different computation abilities and determine the network's architecture. They can be mainly categorized into two main categories; i) feedforward & ii) feedback (recurrent) neural networks.

1.2 Development of neural networks - feedforward and feedback models

Perceptrons can be considered as the pioneer work of feedforward neural network model. This model together with the learning rule was developed during 1957 & 1958 by Frank Rosenblatt *et al.* with application on pattern recognition. During that period, some other models, like ADALINE were also proposed. However, the development stopped for a long time due to Minsky & Papert's book *Perceptrons* proving mathematically that a perceptron with just two layers of neurons (one input layer and one output layer) could not perform some functions, like XOR logical operation. Although it is pointed out that by simply adding another layer (hidden layer) of neurons can solve the problem, the learning algorithm for such a network was not available until 1986. D. Rumelhart, R. William, *et al.* [RM86] proposed the use of backpropagation for learning and proved that this feedforward model with hidden layer can be a powerful mapping network. Its main applications so far are clustering (eg. classification of numerical data, determining fuzzy membership sets for building fuzzy control systems, etc.) and function approximation (eg. image compression, financial forecasting,

process control, etc.) [Kli92]

For feedback model, which is also called recurrent neural network model, the history can be traced back to McCulloch & Pitts' work in 1943 [MP43]. Credit for recent attention on this type of model rests entirely with J.J.Hopfield, who was very active in the beginning of 1980s in promoting recurrent neural network model. "Hopfield network" is now a well-accepted name for a recurrent neural network model which was proposed by him in 1982 [Hop82]. Since then, a lot of similar network models are developed. Their applications mainly fall into associative memory and mathematical programming. For associative memory, it can be used, for example, to recover an damaged pattern, like image, instantly without searching through the whole database [HKP91, Chapter 2]. The current research direction is to study the storage capacity of recurrent neural network with the ultimate goal to increase it. On the line of mathematical programming, different researchers demonstrated that recurrent neural network model can be employed for linear, nonlinear, integer and combinatorial programming [KC88] [CS92] [AKH92]. As it's a generic mathematical discipline utilized by a lot of researchers in different fields of engineering, the neural network approach for problem-solving attracts a large group of audience, from Process Scheduling [Hul91], VLSI layer assignment [KLKH91], to Computer Vision [ZC92b]. Since our project selected Computer Vision as the application area, a brief state-of-art survey of recurrent neural networks applied in this field is given in the coming section.

1.3 State of art of applying recurrent neural network towards computer vision problem

Researches on computer vision have an ultimate objective of acquiring different levels of human visual perception abilities, integrating and implementing them by today's technology as a computer vision system. However, so far, the achievement in computer vision is still in a very primitive stage due to our limited knowledge on our visual system. A well-accepted model of a computer vision system is hierarchial where the high-level visions are making some interpretation while the low-level ones are performing preprocessing on images. The high-level vision is very much depending on the lower-level vision. Inadequate quality of the

low level vision performance will greatly hinder the development of the higher level processing and thus the whole hierarchial computer vision system. So, low-level vision plays a very important role in the development and caught a lot of attention for years. In general, the cues that provided by low-level vision include stereo vision correspondence, detected edges and boundaries, optical flow, detected motion, etc. Also due to noise presence, different image filtering (both linear and nonlinear) methods are proposed to further enhance the reliability of the information extracted by the low-level vision. A lot of different algorithms are suggested for different branches of low-level vision. In particular, using regularization theory [PTK85], most of the above problems can be modelled by a mathematical programming framework to minimize an error function subjected to certain constraints inherited from physical properties of visible surfaces. This theory is known to be able to get rid of the ill-condition problem existed for various low-level visions. With a proper design of the error function and constraints, this minimization algorithm can be very robust and if an effective minimization tool is available, the optimal result can be obtained. However, such a robust and generic framework obviously requires heavy computational power. For a 512x512 image, the number of variables involved for the minimization is 262144 (512x512). The scale of the problem is much, much greater than the limit manageable for existing algorithms based on sequential computer. Furthermore, for real application, we are talking about computational speed in real time. Then, this will remain to be an unsolvable problem if we still stick on digital sequential machines. Therefore, it is natural that we have to turn to parallel machines.

Artificial neural network with one of its merit being possessing massive parallelism, is an attractive outlet. After J.J. Hopfield showed the possibility of using Hopfield network to solve NP-complete combinatorial optimization problem [Hop85], a lot of low-level vision problems were mapped to neural network formulation and solved using the dynamic of the network. Table 1.1 is a summary of the low-level vision branches that have already been mapped to neural network formulation of Hopfield network.

Ref.	Application	Type of Optimization	Optimum guaranteed	(S)td. or (M)odified Hopfield Network	Modification Remark
[ZCVJ88]	Image Restoration	CmQP	Yes	M	Annealing method is used
[BT90]	Image Restoration	CxQP	Yes	M	Neuron's transfer function is exponential
[GPP91]	Image Segmentation	CmQP	No	S	/
[LLTL91]	Object Recognition	CmQP	No	S	/
[NC92]	Stereo Vision	CmQP	No	S	/
[PK92]	Image Restoration	CxQP	Yes	M	Paik's neuron model*
[BT92]	Image Restoration	CxQP	Yes	M	Neuron's transfer function is linear
[Hua92]	Image Segmentation	CmQP	No	S	Simulated annealing is used
[WZX92]	Image Segmentation	CmQP	No	S	Simulated annealing is used
[YT92]	Relaxation	CmQP	No	S	/

CxQP - Convex Quadratic Programming with one and only one local minimum state
CmQP - Combinatorial Quadratic Programming with lots of local minimum states at the corners of unit hypercube

* Refer to 2.2.1 for detailed definition

Table 1.1: Showing the existing applications of Hopfield network on computer vision

It can be observed from Table 1.1 that the types of optimization problems fall into two categories: i) convex quadratic programming (CxQP), where there is one and only one local minimum state in the solution space, which is the interior of the unit hypercube, ii) combinatorial quadratic programming (CmQP), where the desired solutions exist only at the corners of the unit hypercube and there are lots of local minimum states.

For the first category - CxQP, the utilization of Hopfield network is not quite the same as the original one proposed for Travelling Salesman Problem. In fact, the problems are much easier than those of the second category - CmQP, which were what J.J. Hopfield targeted at the very beginning. As the Hopfield energy function of problems in this category is convex, it's proved that the energy profile is a convex hyperparabola with one and only one minimum state. Thus, global minimum state can always be guaranteed by any gradient systems, like Hopfield network.

For the second category - CmQP, they are the real hard ones. The Hopfield network can only be stable if only the diagonal elements of the connection matrix are greater than zero and the connection matrix is symmetric. However, even so, the network only converges to a stable state, which is in fact just a local minimum state and global minimum can't be guaranteed. If the initial state is badly placed, the sub-optimal solution's quality will be far, far away from the desired optimal one. In Table 1.1, it's noted that a lot of researchers incorporated annealing methods with the intention to escape from the local minimum states and ultimately be able to obtain a near-optimal solution. Despite its popularity, heavy computational complexity greatly limits its practical use. In the literature, some alternative methods are proposed but they all bear their own limitations. Thus, this remains to be an open question in the neural network field.

1.4 Objective of the Research

C.C. Klimasanskas [Kli92] stated that the development of neural computing over the past five years has been in the area of plug-compatible replacements for existing technology. This statement, to our opinions, best describes the situation. A lot of problems, which were long tackled by some classical methods, are plugged in neural network methodology recently

to provide an alternative solution means. We too are following this direction.

For our research, the recurrent neural network model of interest is Hopfield network. Although there are already a lot of research works on using Hopfield network for some specific applications, a more general and systematic discussion on the procedure of utilizing the neural network, in Klimasanska's wordings - the plugging procedure, is not well presented in the neural network field. Such a discussion will be of great help to domain expertises who may not have much knowledge on recurrent neural network but intend to apply it for specific tasks in their expert domains. So, the main objectives of this research is to study the topics on mapping to neural network formulation, stability of the formulated network and the local minimum limitation of the network with the intention of providing a more *general* and *repeatable* methodology for designing an appropriate and effective neural network solution for problems in, at least, Computer Vision domain. In fact the scope of the applicable domains can be much wider as long as the domain problem can be written down as a mathematical programming model.

1.5 Plan of the thesis

In this thesis, we will first provide background knowledge on Hopfield network in **Chapter 2**. Summary of existing variations of Hopfield network are included and also a generalized Hopfield network model, which is an generalized extension of original Hopfield network, will be described.

Thorough discussions on the procedure of utilizing generalized Hopfield network model for optimization, from the very beginning - problem formulation, to the end - network convergence property, are found in **Chapter 3**. In particular, a reshaping strategy, which is complementary to the neural network formulation step, is proposed and any unstable formulated network can then be transformed to a stable one using this strategy. Also, in the chapter, the local minimum problem of generalized Hopfield network is addressed and a survey on existing methods to solve the problem is given.

Chapter 4 is devoted to describe a novel recurrent neural network model, named

"*Tunneling network*" which is proposed to solve the local minimum problem of generalized Hopfield network. Our Tunneling network comes from the marriage of Hopfield network and a global optimization technique known as tunneling algorithm [Yao89]. The intuitive idea of the tunneling algorithm is to escape the local minimum by "tunneling" through the nearby barriers for a new local minimum valley. In order to well define the Tunneling network capability, a thorough convergence analysis is performed and its global minimization ability is compared with simulated annealing. Promising results are obtained in terms of both efficiency and effectiveness. Besides, to further enhance the quality of the solution, network decomposition is proposed and it's found that certain decomposition configuration can push the network to a solution closer to the optimal one.

As described in Section 1.3, Computer Vision problems, in the views of mathematical programming, can be categorized mainly into two groups. To have a complete study of our neural network solution to the area of Computer Vision, two problems - *gaussian filtering* and *boundary detection* are chosen from the two categories respectively for study. In **Chapter 5**, recurrent neural networks for both 1D and 2D gaussian filtering are derived and the filtering properties of the derived networks, based on simulation, are also illustrated. Besides, during the application, the difficulty for boundary connection assignment has been identified and formula are derived to ease the problem. In **Chapter 6**, a recurrent neural network for boundary detection is derived. Its problem formulation is adopted from one of the active contour models, which is of order three. So, we are in fact working on a *third-order* combinatorial problem. To the best of our knowledge, this is the *first* time using a recurrent neural network to tackle a practical problem with order higher than quadratic's.

Finally, conclusions are made in **Chapter 7**, where future research suggestions are also given.

Chapter 2 Background

Hopfield network is the most popular neural network model applied for optimization during recent years. Our neural network solution to optimization problems throughout the thesis are developed mainly based on it. In this chapter, background knowledge on Hopfield network which are essential to the development of our work are summarized. Also, a generalized Hopfield network model, which is an extended model for catering higher-order problems, will be introduced.

2.1 Short history on development of Hopfield-like neural network

The first suggestion of neural network with architecture similar to Hopfield network can be found as early as the one developed by McCulloch & Pitts in 1943 [MP43]. At that time, they proposed a very simple neuron model with binary threshold. Specifically, each neuron computes a weighted sum of its inputs from other neurons, and outputs a one or zero according to whether this sum is above or below a certain threshold. Let x_i be the state of ith neuron, μ_i be its threshold, w_{ij} be the connection weight between i^{th} and j^{th} neurons and $g(x)$ be the step function,

$$x_i(t+1) = g\left(\sum_j w_{ij} x_i(t) - \mu_i \right) \qquad ..(2.1)$$

Eqn.(2.1) characterized McCulloch & Pitts' neuron and it's proved that a synchronous assembly of such neurons is capable, in principle, of universal computation when the weights w_{ij} are chosen suitably. After the McCulloch & Pitts' threshold network proposal, considerable follow-up work were found in literature [Min67] [Ama72].

In 1981, J.J. Hopfield [Hop82] restudied the collective computational ability of the network and was able to add some physical insight by introducing an energy function, and by emphasizing the notion of memories as dynamically stable attractors. The neuron model was very similar to McCulloch & Pitt's, except the two possible neuron states being +1/-1 instead of 1/0 such that inhibitory force can be introduced for a faster convergence. Hopfield

demonstrated the possibility of associative memory application. In 1984 [Hop84], he proposed a continuous updating version of Hopfield network, where the neuron model was replaced by a sigmoid function which can be represented by the response of an Operation Amplifier. He showed that such continuous network possesses similar computational ability as the discrete network and the most important implication was that Hopfield network can be implemented by realistic analog circuit instead of just a theoretical model.

The value of Hopfield network became much more apparent since the proposal of its application to combinatorial optimization. Hopfield *et al.* [Hop85] attempted to map the Travelling Salesman Problem(TSP) to Hopfield network formulation and showed that nearly-optimal solution can be obtained by the network. As TSP is an NP-complete combinatorial optimization problem, which is known to be a really hard problem, the argument of the computational power of the neural network is here to stand.

This section is just a short history on Hopfield network. For more details, interested readers are recommented to refer to the books, [HKP91] [Nie90].

2.2 Hopfield network model

Hopfield network model is basically represented by a group of neurons, which are fully interconnected with an external bias for each individual and form a feedback architecture. This framework can also be fully represented by the updating rule,

$$x_i = g\left(\sum_j T_{ij}x_j + I_i\right) \qquad ..(2.2)$$

where x_i is the state value of ith neuron,

T_{ij} is the connection weight between i^{th} and j^{th} neurons. All the connection weights form a connection weight matrix and it must be symmetric,

I_i is the external bias for i^{th} neuron,

$g(x)$ is the neuron's transfer function

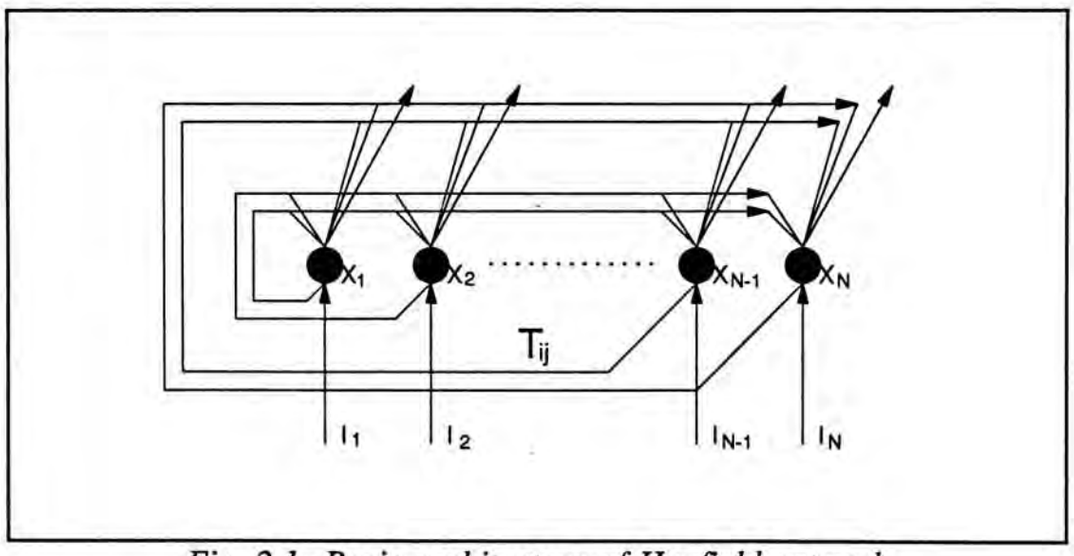

Fig. 2.1: Basic architecture of Hopfield network

As there are lots of variations of Hopfield network in the literature, to clarify exactly what version of Hopfield network is being used, it's necessary to clearly specify the following two aspects, i) neuron's transfer function & ii) neuron's updating sequence.

2.2.1 Neuron's transfer function

The characteristics of neuron's transfer function is multifarious. Some even include exponential response, logarithm, etc. Here, we will give a summary of them. As some of them

were proposed in *ad hoc* manner or just for specific applications, therefore are hard to be generalized. Only the more common ones in terms of application possibility are summarized.

Let $\hat{u}$ be the input vector to a neuron

v be the output state of that neuron

$g(x)$ be the neuron's transfer function

Then, $v_i = g\left(\sum_j u_j\right)$

a) McCulloch & Pitts' model

The neuron's transfer function is simply a step /hardlimiting function defined as:

$$g(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{if } x > 0 \end{cases}$$

and it's a two-state neuron.

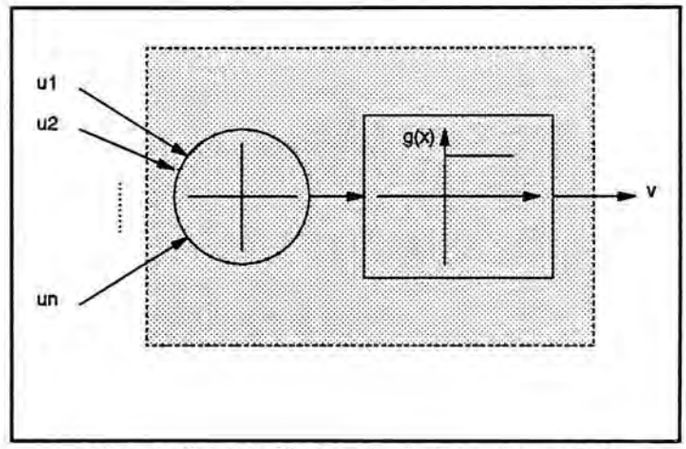

Fig. 2.2: McCulloch & Pitts' model

AR[1]: [MP43] [NC92]

b) Bipolar model

The neuron's transfer function is simply a sign function defined as:

$$g(x) = \begin{cases} -1 & \text{if } x \le 0 \\ 1 & \text{if } x > 0 \end{cases}$$

and it's a two-state neuron.

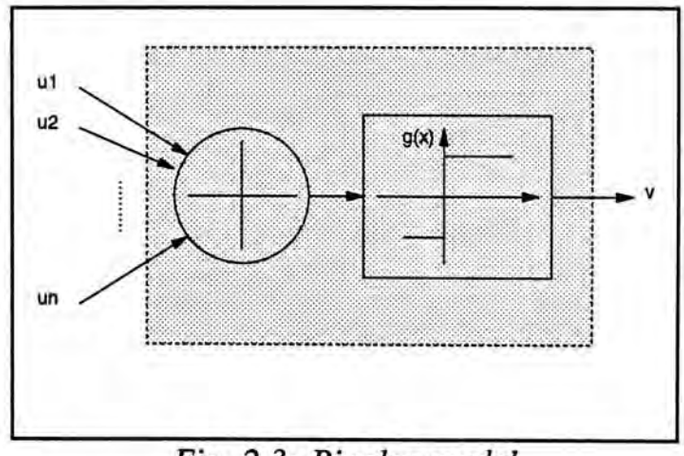

Fig. 2.3: Bipolar model

AR: [Hop82]

[1] AR stands for references applying the neuron model

c) Sigmoid model

The neuron's transfer function is a sigmoid function. There are two typical sigmoid function examples.

$$g_1(x) = \frac{1}{1 + \exp(-\beta x)}$$

$$g_2(x) = \tanh(\ \beta x\ )$$

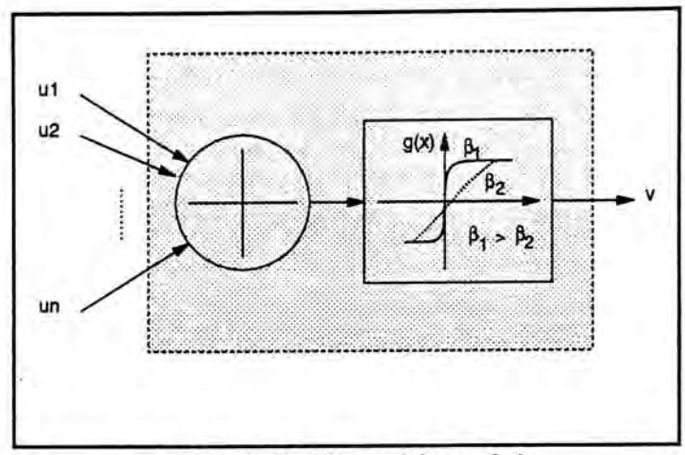

Fig. 2.4: Sigmoid model

The range of $g_1(x)$ is (0,1) and that of $g_2(x)$ is (-1,1). ß is the gain of the sigmoid. When ß approaches positive infinite, the sigmoid function approaches the hard-limiting function.

This neuron model takes continuous value within the range of the sigmoid function.

AR: [HT85] [TH86] [LLTL91]

d) Paik's model

This neuron model is proposed by Paik *et al.* for image restoration application.

$$v_i(t+1) = g(\ v_i(t) + \Delta v_i\ )$$

where

$$g(x) = \begin{cases} 0 & x<0 \\ x & 0 \le x \le 255 \\ 255 & x>255 \end{cases}$$

$$\Delta v_i = d_i(w_i) = \begin{cases} -1 & w_i < -\theta_i \\ 0 & -\theta_i \le w_i \le \theta_i \\ 1 & w_i > \theta_i \end{cases}$$

where $\theta_i = -T_{ii}/2 > 0$

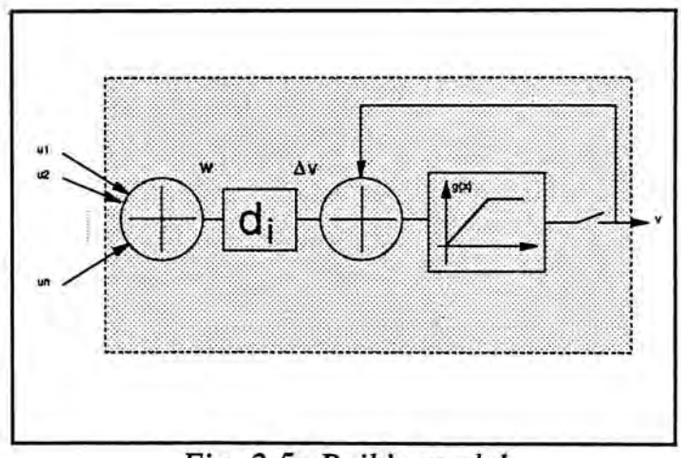

Fig. 2.5: Paik's model

It's a multi-state neuron, which can be any integer in the range of [0,255] instead of just {0,1}. Thus, we consider it as a generic model for integer programming.

AR: [PK92]

e) Peterson's model

This neuron model is the most complicated one among the four mentioned. It requires communication with other neurons of a group, G to define its transfer function.

$$g(x_i) = \frac{1/(1+\exp(-\beta x_i))}{\sum_{x_j \in G} 1/(1+\exp(-\beta x_j))}$$

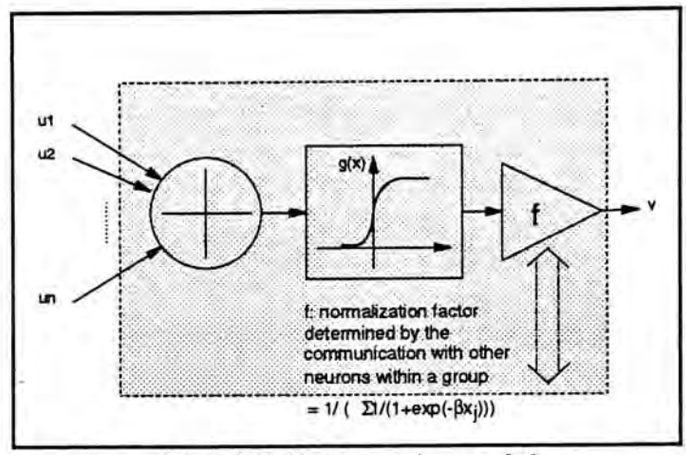

Fig. 2.5 Peterson's model

Firstly, it takes continuous value within the range [0,1]. Moreover, summation of the neuron's output among the group will be one. This is a constraint of the neurons' state embedded in the model.

AR: [PS89] [BM90]

As a neuron is the basic processing element of an artificial neural network, different transfer characteristics will imply different collective computational capability. So, for different applications, an appropriate models should be chosen accordingly.

2.2.2 Updating sequence

Different updating sequence for the neurons will affect the converging behaviour of the network. It can be categorised as:

i) *Asynchronous updating*. One neuron is selected to be updated at a time.

ii) *Synchronous updating*. At each time step, all neurons are updated simultaneously.

iii) *Continuous updating*. All the neurons are updated continuously according to the network's differential equation corresponding to the network's updating rule.

For continuous updating, instead of discrete updating rule as described by eqn.(2.2), the dynamics of the network is characterised by a differential equation:

$$\frac{du_i}{dt} = \sum_{j \neq i}^{N} T_{ij} x_j + I_i - \frac{u_i}{\tau} \qquad ..(2.3)$$

$$x_i = g(u_i)$$

where τ is the neuron's decay constant,

$g(x)$ is sigmoid function.

The discrepancy is mainly due to the fact that continuous updating is simulating the time evolution of Hopfield network realized by realistic analog circuitry.

Although all three updating sequences can be simulated by a digital computer, the continuous updating, which is corresponding to the real dynamics of a system defined in an n-dimensional continuous domain, can only be implemented using numerical method and thus takes a much longer time for convergence compared with the other two. Also, using asynchronous/synchronous updating, the neuron can only take discrete value while the neuron should take continuous value for continuous updating. Besides, an algorithm being capable to be implemented using asynchronous updating is more desirable than the synchronous counterpart as it can ease the synchronization problems during the implementation step.

After the proper choice of neuron's transfer function and updating sequence, the network can be programmed for different applications by assigning suitable values to the connection weight matrix and the external bias vector accordingly. Utilizing the dynamics of the Hopfield network, which is massively parallel, real-time applications become possible. For a more comprehensive understanding of the network dynamics, the "energy" concept described in the coming section will be of great help.

2.3 Hopfield energy function and network convergence properties

One of the most important contributions of the paper for the original Hopfield network was to introduce the idea of an energy function into neural network theory. Hopfield illustrated the fact that the updating rule for the neurons, which determines the network's dynamics, is in effect minimizing an energy function defined by,

$$E = -\frac{1}{2}\sum_i\sum_j T_{ij}x_ix_j - \sum_i I_ix_i \qquad ..(2.4)$$

Assume the ith neuron is to be updated. According to eqn.(2.1),

$$x_i(t+1) = g(\sum_{j\neq i} T_{ij}x_j(t) + I_i)$$

where $g(x)$ is bipolar/sign function for original Hopfield network and the updating sequence is asynchronous.

As the matrix $\{T_{ij}\}$ is symmetric, $\frac{\partial E}{\partial x_i} = -\sum_{j\neq i} T_{ij}x_j - I_i$.

i) If x_i remains unchanged, $\Delta x_i = 0$ and thus $\Delta E = 0$.

ii) If x_i changes state,

$$\begin{aligned}\Delta x_i &= x_i(t+1) - x_i(t)\\ &= 2\ x_i(t+1)\\ &= 2\ (\sum_{j\neq i} T_{ij}x_j(t) + I_i)\\ &= -\ sign\left(\frac{\partial E}{\partial x_i}\right)\end{aligned}$$

Thus, $\Delta E = -|\frac{\partial E}{\partial x_i}| \leq 0$

Combining i) and ii), we can see that as the network evolves, the energy function, E described by eqn.(2.4) is nonincreasing. Also, as E is bounded below, it is a Lyapunov function and this implies convergence of the Hopfield network. Furthermore, the converged state is a local minimum point of E.

Thus, the dynamics of the network can be understood as a trajectory along the energy profile. Very often, this energy function is named as "Hopfield energy". For the continuous Hopfield network, where the neuron's nonlinearity is a sigmoid function, the Hopfield energy will become,

$$E_c = -\frac{1}{2}\sum_{j\neq i}\sum T_{ij}x_ix_j - \sum_i I_ix_i + \sum_i (1/\tau)\int_0^{x_i} g_i^{-1}(x)\ dx \qquad ..(2.5)$$

The convergence consideration is similar to the discrete model's. Again consider the ith neuron,

$$\frac{du_i}{dt} = \sum_{i \neq j} T_{ij} x_j + I_i - \frac{u_i}{\tau}$$

$$x_i = g(u_i)$$

where $g(x)$ is the sigmoid function.

As $\{T_{ij}\}$ is symmetric,

$$\frac{\partial E_c}{\partial x_i} = - \sum_{j \neq i} T_{ij} x_j - I_i + \frac{g^{-1}(x_i)}{\tau}$$

$$= - \sum_{j \neq i} T_{ij} x_j - I_i + \frac{u_i}{\tau}$$

$$\frac{dE_c}{dt} = \frac{\partial E_c}{\partial x_i} \frac{dx_i}{dt}$$

$$= \frac{\partial E_c}{\partial x_i} g'(u_i) \frac{du_i}{dt}$$

$$= - g'(u_i) \left(\frac{\partial E_c}{\partial x_i} \right)^2$$

$$\leq 0 \qquad \textit{,as } g(x) \textit{ is a nondecreasing differentiable function}$$

Thus, again E_c is monotonic decreasing as time proceeds. Also, the Hopfield energy described by eqn.(2.5) is bounded below. Thus, it is shown to be a Lyapunov function and the convergence of the network towards a local minimum point is guaranteed.

For the relationship between the discrete and continuous model, it can be shown [Hop84] that when the gain of the sigmoid function is very large, the energy profile E_c will be very close to E. Fig. 2.7 illustrate a typical energy profile of E and corresponding E_c with different neuron's gain for a neural based 2-bit AD convertor [TH86]. When the gain is very small, the only minimum state is all-zero state. As the gain increases, the energy profile will start to modify. When the gain is increased to a very large value, E_c will be very close to E. This verifies Hopfield's argument that the continuous Hopfield network with large gain is in fact a practical implementation of original Hopfield network with very similar computational capability.

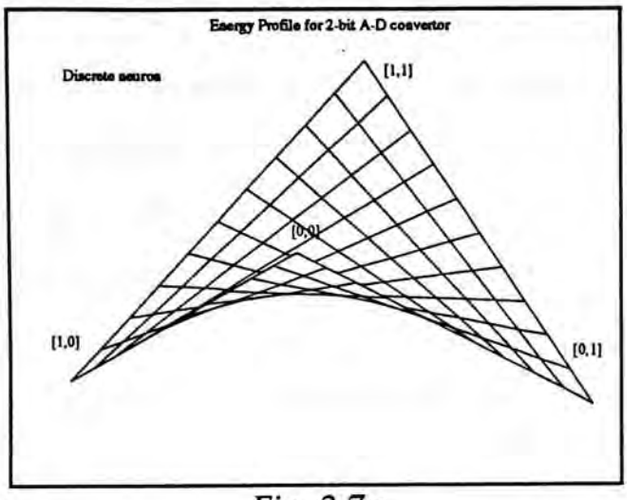

Fig. 2.7a

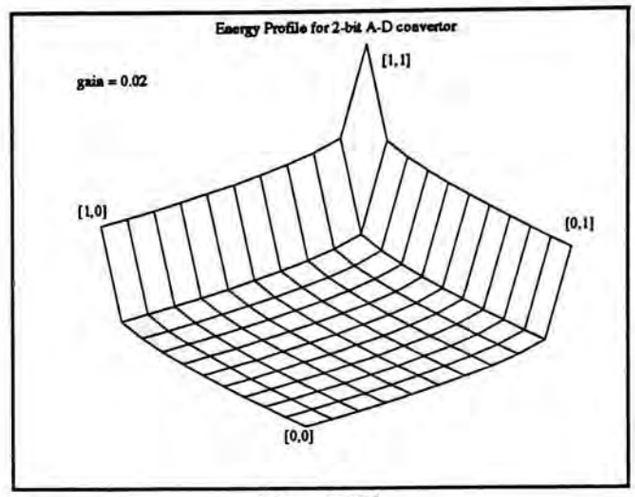

Fig. 2.7b

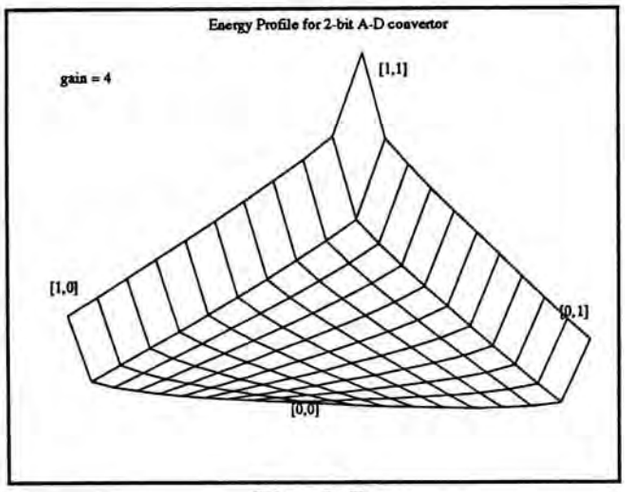

Fig. 2.7c

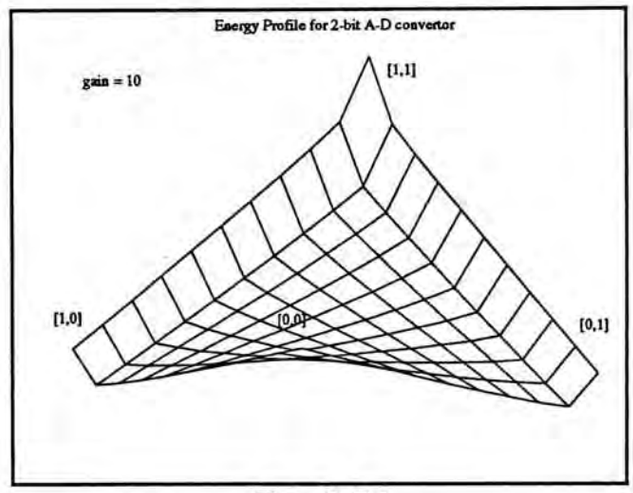

Fig. 2.7d

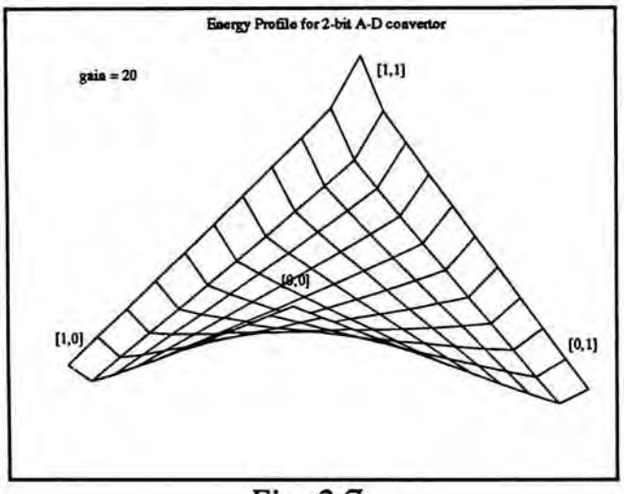

Fig. 2.7e

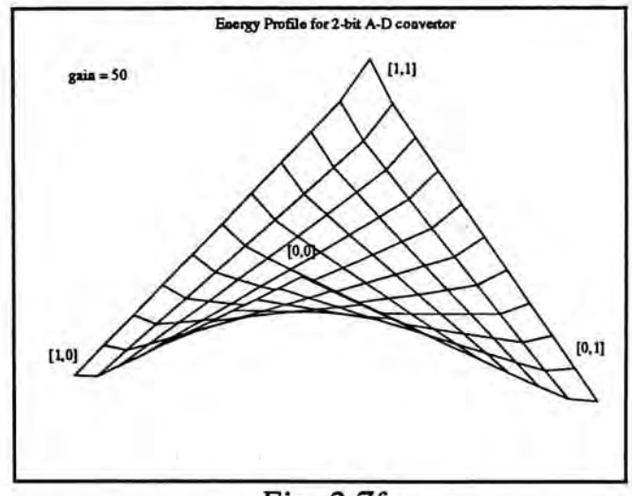

Fig. 2.7f

Fig. 2.7a shows the energy profile for discrete Hopfield network. Fig. 2.7b-f shows the energy profiles for continuous Hopfield network with the neuron's gain equal, b) 0.02; c) 4; d) 10; e) 20; f) 50. It's noted that the energy profile for continuous Hopfield network with large gain is very similar to that for the discrete network.

Using the energy concept, the convergence proofs of two most fundamental versions of Hopfield networks are shown. However, by changing the neuron's transfer function and the updating sequence, possible configurations of Hopfield network are really abundant. As long as a new configuration of Hopfield network is proposed, the dynamics of the network will change. This means different trajectory on the Hopfield energy profile and thus its convergence property has to be studied again to guarantee the network's stability. Following is a summary of some existing important theorems for the convergence properties of certain configurations of Hopfield network. Their proofs will not be restated here. Interested reader can refer to the appropriate references.

Theorem 2.1 [Hop82] [BG88]

An asynchronous updating Hopfield network with neurons of *McCulloch & Pitts' model* will converge to a stable state, which is equivalently a local minimum point of the corresponding Hopfield energy and is on the corners of the unit-hypercube, provided that the connection weight matrix of the network is *symmetric* and the diagonal elements are all *non-negative*.

Note: For neuron model replaced by large gain sigmoid function, Theorem 2.1 remains to be valid, except the stable state will not be exactly on the corners of the unit-hypercube but very close to it. The situation is revealed in Fig. 2.7f. The Hopfield networks satisfying the conditions are mostly the ones considered for combinatorial optimization.

Theorem 2.2 [LMP88] [FMM92]

A continuous updating Hopfield network with neurons of *sigmoid model* will converge to a stable state, which is equivalently a unique local minimum point of the corresponding Hopfield energy and is the interior of the unit-hypercube, provided that the connection weight matrix of the network is *symmetric* and everywhere *negative semi-definite* within the unit-hypercube.

Note: The Hopfield networks satisfying the conditions will associate with a Hopfield energy profile of a convex hyperparabola. As the problem domain - unit-hypercube is also

convex, this network is in fact solving convex programming. This theorem is used in in Chapter 5 to guarantee the convergence of a continuous version of recurrent neural network for gaussian filtering.

Theorem 2.3 [PK92]

An asynchronous updating Hopfield network with neurons of *Paik's model* will converge to a stable state, which is close to the unique local minimum point of the corresponding Hopfield energy of the network described by Theorem 2.2 with a small and bounded residue error in terms of Hopfield network, provided that the connection weight matrix is *symmetric* and the diagonal elements are all *negative*.

Note: This Hopfield network configuration, in fact, can be considered as a discrete replacement of the one described in Theorem 2.2. Although Paik's model takes value in the set {0, 1,.... 255} and the range of sigmoid's is [0,1], simple scaling of Paik's model's (or sigmoid's) output can unify the model's upper and lower limits but it's obvious that Paik's model only takes 256 discrete values in the range of [0,1]. Due to that quantization, the converged state can't be exactly the minimum point. Paik *et al.* proved that the error is within a small bound. If that small error is tolerable, this discrete replacement is very efficient for simulation. Such configuration is used in Chapter 5 for a discrete version of recurrent neural network for gaussian filtering and Theorem 2.3 is important for the network convergence guarantee.

These three convergence theorems are important as they clearly state the stability requirement of different versions of Hopfield network studied and adopted in our research. Before utilizing particular versions of Hopfield network, the convergence conditions stated in the theorems have to be checked beforehand.

2.4 Generalized Hopfield network

2.4.1 Network order and generalized Hopfield network

From the energy function perspective, the original Hopfield network is in effect minimizing a *quadratic* function. To tackle with a higher-order problem, it's natural to generalize the network model by introducing *higher-order connections* to increase the order of the network. To define the *network order* quantitatively, a Hopfield network is of nth order if the input to each neuron is determined by an nth order polynomial expression. Under this definition, the original Hopfield network is a 1st order Hopfield network. We name this generalized model as *generalized Hopfield network.* Inherited from Hopfield network's theory, each generalized Hopfield network also associates with a generalized Hopfield energy function. For an nth order network, the corresponding energy function will be of $(n+1)^{th}$ order, as depicted in next subsection.

Although the generalized Hopfield network model has already been described by Xu *et al.* [XT91] in great details. However, our generalized Hopfield network model is a little bit different from that proposed by Xu *et al.*. Thus it's necessary to state clearly our adopted model's characteristics. Let N be the total numbers of neurons whose state variables denoted by x_i. The updating rule for our generalized Hopfield network is:

$$x_i = f_h\left[\sum_{i_1}^{N}\sum_{i_2}^{N}\cdots\sum_{i_{n-1}}^{N} T_{i_1 i_2 \cdots i_{n-1} i} x_{i_1}\cdots x_{i_{n-1}} + \sum_{i_1}^{N}\cdots\sum_{i_{n-2}}^{N} T_{i_1 \cdots i_{n-2} i} x_{i_1}\cdots x_{i_{n-2}} \cdots\cdots + \sum_{i_1}^{N} T_{i_1 i} x_{i_1} + I_i\right] \quad ..(2.6)$$

where the value of $T_{i_1 i_2 \cdots i_k}$ is independent of the ordering of the index for k=1 to N. For Xu's model, a coefficient of $1/(k\text{-}1)!$ is introduced to the kth order term for all possible k. The discrepancy is in fact quite minor. By embedding the coefficients into the connection weights, Xu's model will be transformed to our's. We incorporate the modification just for the sake of updating rule's simplicity.

2.4.2 Associated energy function and network convergence property

As mentioned in 2.4.1, each generalized Hopfield network is also associated with an energy function, which is called generalized Hopfield energy function and is described by eqn.(2.7).

$$E = -\frac{1}{n}\sum_{i_1}^{N}\sum_{i_2}^{N}\cdots\sum_{i_n}^{N} T_{i_1 i_2 \cdots i_n} x_{i_1} x_{i_2} \cdots x_{i_n} - \frac{1}{n-1}\sum_{i_1}^{N}\sum_{i_2}^{N}\cdots\sum_{i_{n-1}}^{N} T_{i_1 i_2 \cdots i_{n-1}} x_{i_1} x_{i_2} \cdots x_{i_{n-1}} \quad ..(2.7)$$
$$\cdots\cdots \quad - \frac{1}{2}\sum_{i_1}^{N}\sum_{i_2}^{N} x_{i_1} x_{i_2} - \sum_{i}^{N} I_i x_i$$

It can be proved that the dynamics of the generalized Hopfield network is to minimize the energy function depicted in eqn.(2.7) and the convergence property is summarized in Theorem 2.4.

Theorem 2.4

An asynchronous updating generalized Hopfield network with McCulloch & Pitts' neuron model will converge to a stable state, which is equivalently a local minimum point of the corresponding generalized Hopfield energy, and is on the corners of the unit-hypercube provided that the value of $T_{i_1 i_2 \cdots i_k}$ is independent of the ordering of the index for k=1 to N and all self-reinforcement connections, ie. some indices of $T_{i_1 i_2 \cdots i_k}$ being equal, are zero.

Proof:

On satisfaction of the two conditions of the theorem, i) the value of $T_{i_1 i_2 \cdots i_k}$ is independent of the ordering of the indices for k = 1 to N and ii) all the self-reinforcement connections being zero, eqn.(2.6) can be written as

$$x_i(t+1) = f_h\left[-\frac{\partial E}{\partial x_i(t)}\right]$$

Computing ΔE gives,

$$\Delta E = \frac{\partial E}{\partial x_i(t)} \cdot \Delta x_i$$

i) If x_i remains unchanged, $\Delta x_i = 0$ and thus $\Delta E = 0$.

ii) If x_i changes state,

$$\begin{aligned}
\Delta x_i &= x_i(t+1) - x_i(t) \\
&= x_i(t+1) - x_i(t) \\
&= x_i(t+1) - (1-x_i(t+1)) \\
&= 2x_i(t+1) - 1 \\
&= 2\, f_h\left(-\frac{\partial E}{\partial x_i(t)}\right) - 1 \\
&= -\, sign\left(\frac{\partial E}{\partial x_i(t)}\right)
\end{aligned}$$

$$\begin{aligned}
\Delta E &= \frac{\partial E}{\partial x_i(t)} \cdot \Delta x_i \\
&= -\frac{\partial E}{\partial x_i(t)} \cdot sign\left(\frac{\partial E}{\partial x_i(t)}\right) \\
&= -\left|\frac{\partial E}{\partial x_i(t)}\right| \le 0
\end{aligned}$$

So, E is nonincreasing. Also, as E is bounded below, the property of convergence towards local minimum should follow. ■

Refer to Xu's work, our generalized Hopfield network model can be easily extended to include continuous updating version, which reveals the possibility of implementing it on chip by similar argument as Hopfield's continuous updating version.

2.4.3 Hardware implementation consideration

The possibility of implementing Hopfield network by VLSI technology is a quite well accepted fact in the neural network field due to a lot of successful works in this direction [LS91a] [LS91b] [Als89] [LSCC92]. However, when the implementation of generalized Hopfield network is considered, it's obvious that the network model will require a very large amount of connection weights, which are hard to be put on silicon chip, and the increase is exponentially proportional to the network order. In Chapter 6, we adopted the 2nd order Hopfield network for a boundary detection problem. The huge requirement of connection weight for a practical application will be discussed there. Also, the massive connectivity is comparatively difficult in electronic systems because electrical signals must travel through a physical wires that consumes area on the chip and require careful design to minimize interference and crosstalk. Therefore, establishment of a large network using present VLSI

technology is almost impossible.

A very promising alternative instead of electrical system is optical one. Optical signals can propagate through space and pass through on another without interacting. This critical characteristics solved all the drawbacks of electrical systems and the huge amount of connection weights can be integrated in a smaller space. One of the possible ways to provide the massive interconnections required in most neural network models is holography proposed by Psaltis *et al.* [PYHLGB89]. Psaltis *et al.* demonstrated an heteroassociative memory using optically implemented Hopfield network and the network connections are all embedded in a *volume hologram.* Using this technology, light rays can be redirected in a programmable fashion which makes the programming of the Hopfield network possible. In the future, success in implementing a larger network using this optical means definitely will push generalized Hopfield network to much wider range of practical applications.

Chapter 3 Optimization by recurrent neural network

In this chapter, we will consider the steps in utilizing recurrent neural network utilized for optimization. First neural network formulation for a particular problem will be addressed, with special attention being paid to the formulation for hard constraints. Next, we will describe a *"reshaping strategy"* we proposed to guarantee network convergence. With this strategy, any neural network formulations can be transformed to the ones with convergence guaranteed. Finally, the local minimum limitation of Hopfield network is pointed out and a review on the existing solutions in the literature is given, including Simulated Annealing, Mean Field Annealing, Adaptively Changing Neural Network and Correcting current method.

3.1 Mapping to Neural Network formulation

To have a problem solved by recurrent neural network, we have to map it to a neural network formulation. A standard procedure can be followed. Firstly, identify the problem and try to define it using a mathematical programming model and a cost function defined within the problem domain bounded by certain constraints. The cost value associated with a solution is used to judge the optimality of the solution acquired. About the problem domain, the considered one is just the unit-hypercube, which is in fact a convex set. Through scaling the neurons' output, it can be extended to a hypercube with arbitrary length. (Although there exists some methods, by which the problem domain can be restricted arbitrarily using both equality and inequality constraints [CS92] [AKH92], they will not be considered in our study.) This problem formulation step is the most important and difficult one as it requires conceptual understanding of the problem and ability to write it down using a reliable and objective mathematical programming model. Sometimes, the concept may not be so easily defined accurately and objectively. For example, the quality of a restored image is very often measured by mean-square-error (*mse*) with the original image. However, our subjective visual system may consider an image with greater *mse* to be of better visual quality than the one with smaller error. Consider, a 512x512 image with every pixel value shifted up by 1 and another identical image with just say, twenty salt and pepper noise added. The former one definitely has a greater *mse* but the one with worse visual impression is obviously the latter one.

After the problem is properly defined, select a proper network configuration according to your own need. For examples, for problems with solutions inside the interior of unit-hypercube, we may need a network configuration with neuron's transfer characteristic of continuous type, like sigmoid model. If in some cases where computational complexity is important, Park's model can be used instead of sigmoid's (Refer to Section 2.2.1). However, if existing configurations can't fulfil your requirement, we have to derive a new one. For a new network configuration, you have to perform convergence analysis to study the network's stability and this may require *great* work load! Then, to obtain the optimal solution with the help of the chosen recurrent neural network, we have to map the problem to neural network formulation.

If the problem is of continuous type, the mapping is as straight forward as comparing coefficients between the cost function and the generalized Hopfield energy function; then the connection weight matrix and the external bias of the network can be defined accordingly. However, so far, Hopfield network is only used to solve continuous convex quadratic programming and due to the problem's specific nature, the network can always converge to global minimum. For higher order cases, only sub-optimal solution can be guaranteed and probably, that's the reason why Hopfield network has not been used for solving continuous higher-order programming and we don't consider this situation in our research.

If the problem is of combinatorial type, more have to be considered. If the combinatorial problem is quadratic, then the connection weight between i^{th} and j^{th} neurons, T_{ij} will be defined by their cost contributed to the Hopfield energy when they are activated and form a part of the solution. This step is not much different from the one for the continuous optimization. However, as the problem is of combinatorial type, solution must be equivalent to a meaningful combination. As in most of the situations, we are interested in the best combination of fixed number of activated neurons, Hopfield network with such a connection weight matrix is not sufficient to obtain a desired solution.

Let's assume that the energy term to be optimized is always positive. (This is in fact a very common fact, eg. *mse*) If the network dynamic is to bring the associated Hopfield energy to a minimum state, obviously the state with all neurons inactivated will be the

solution, ie. $x_i = 0\ \forall i$, where x_i is the state variable of ith neuron. However, this is definitely not a meaningful solution. In most of the situations, we are interested in the best combination of fixed number of activated neurons. So, we have to force the network to make *decision* instead of just stopping at a all-zero state. With the objective of getting a valid and meaningful solution, an additional energy term is then necessary to impose a hard constraint to the neural network. To illustrate the concept, let $E_{opt}(x)$ be the energy to be optimized and $E_{hard}(x)$ be the additional energy term for hard constraint. An overall energy term can be defined as:

$$E_{all}(x) = E_{opt}(x) + \lambda\ E_{hard}(x)$$

where $E_{hard}(x)$ is formulated such that

$$E_{hard}(x) \begin{cases} = 0 & \text{,when } x \text{ is a valid solution} \\ > 0 & \text{,otherwise} \end{cases}$$

The constraint λ has to be chosen such that $\lambda\ E_{hard}(x) >> E_{opt}(x)$ whenever x is corresponding to an invalid state, all the minimum states of $E_{all}(x)$ will be equivalent to those of E_{opt} subjected to the hard constraints. This fact is also true for $E_{opt}(x)$ being not always possible as long as λ is large enough. Such a method is commonly used in mathematical programming for changing a constrained problem into an unconstrained one and is known as the penalty method. According to the term $\lambda\ E_{hard}(x)$, another connection weight matrix can be defined.

The following stage is obvious that proper weighting, ie. λ, between the two connection matrices (or the energy terms) has to be determined in a trade-off manner. As the energy term for the hard constraint must be satisfied or the solution will be meaningless, of course λ must be relatively large enough. However, if this weighting is too large, the quality of the solution will have to be sacrificed. So how to set the optimal weighting is itself not an easy problem and is problem dependent. A lot of researchers fixed it by trial and error while some spends a lot of effort to perform analysis to determine the value for just a particular application [ANF90] [CMS91]. Recently, a Lagrange Programming Neural Network [ZC92] is proposed where the weighting, which is also very often named as *Lagrange multiplier*, is obtained dynamically by the network. The robustness and adaptability advantages make it a very promising solution for fleeing from the optimal weighting determination. However, in this project, this network model has not been tried.

Before proceeding to next section, it's worthwhile to point out the step of determining the energy term for the hard constraints. Travelling Salesman Problem (TSP) can be considered as the first NP-completeness combinatorial optimization problem mapped to neural network formulation by J.J.Hopfield [HT85]. A lot of follow-up work proves that the network very often converges to an invalid solution no matter how you adjust the weighting for the different energy terms[WP88]. In fact, the penalty energy term, adopted in [HT85], composes several hard constraints. Each has its own constrained solution space. The intersection of the constrained solution spaces is the space for valid solution. Setting the weighting among the constraints such that the converged solution of the network is valid, is found to be not so straight forward. Aiyer *et al.* [ANF90] performed an eigenvalue analysis and in fact reformulated the energy term before a valid solution can be guaranteed. This is an discouraging fact as so much effort is necessary to guarantee just *valid* solution while we are in fact expecting the *optimal* one. Instead of performing involved analysis, it would be nice to have a simple guideline to write the energy term for the hard constraints such that it confines the space for valid solution directly.

For example, let's consider our old TSP problem again. The solution is represented by a matrix of neurons state variables $\{v_{ij}\}$ with possible state value equal 0 or 1. The change of the denoting variable from x to v for the neuron state is just for the sake of sticking to the one used in [HT85]. Row index stands for different cities and the column index stands for the sequence of reaching a particular city. The energy term for the hard constraints, $E_{hard}(v)$ is,

$$A \sum_{x} \sum_{i \neq j} \sum v_{xi} v_{xj} + B \sum_{x \neq y} \sum_{i} \sum v_{xi} v_{yi} + C \left(\sum_{x} \sum_{i} v_{xi} - N \right)^2 \qquad ..(3.1)$$

For the first two terms, whenever there is more than one neuron activated for each row or column, the first or second term will be greater than zero respectively. Thus, they constrain the solution to each row and column consisting of less than or equal to one activated neuron. For the third term, it is to make sure that total number of activated neurons being N, the number of cities or it will be greater than zero. If these three constraints are satisfied simultaneously, a valid path will be defined. In order to enforce the constraints effectively, the corresponding energy terms have to be added together in a proper trade-off manner by setting the coefficients A, B and C appropriately. However, it's extremely difficult to set the coefficients such that valid solution can be guaranteed for most of the time. In fact, Wilson

& Pawley [WP88] found that just around one percent of the network converging trials can give a valid solution. A lot of converged states are in fact invalid solutions, which is mainly due to the improper formulation of eqn. (3.1).

Instead of tuning the set of weighting to obtain a proper formulation, in fact, we can modify eqn. (3.1) to eqn. (3.2),

$$A \left(\sum_i (\sum_x v_{xi} - 1)^2 + \sum_x (\sum_i v_{xi} - 1)^2 \right) \qquad ..(3.2)$$

The first and the second term are corresponding to the constraint of each row only consisting of one activated neuron and that of each column only consisting of one activated neuron. It's obvious that the importance of the two constraints are of equal rating and also due to their expressions' similarity, the weighting for each of them can be simply set to 1. And then, for large enough A, the valid solution can be guaranteed easily without the troublesome parameters tuning.

So, in determining the formulation of energy term for the hard constraints, try not to use the one with several terms inter-related with one another as eqn.(3.1) such that careful relative weighting tuning is necessary to allow the valid solutions to coincide with the local minimum states. This will generate a lot of unfavourable local minimum states in the space belonging to invalid solutions.

3.2 Network stability verse Self-reinforcement

After properly mapping a problem to the neural network formulation, the problem will then be fed to a neural network black box and one would expect that the *optimal* solution can be obtained in *real time*. Unfortunately, we are still far from this goal. Certain primitives which are also important properties of the neural network have to be well studied before the development can be proceeded. Stability consideration is an example. For discrete version of Hopfield network, the condition for network stability is symmetric connection weight matrix

with non-negative diagonal elements as stated in Theorem 2.1 in Chapter 2. However, this rule is easily violated during practical applications. Consider a constraint for only one neuron being activated among n neurons. The corresponding energy term can be written as,

$$\begin{aligned} E &= \left(\sum_{i=1}^{n} x_i - 1 \right)^2 \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j - 2\sum_{i=1}^{n} x_i + 1 \\ &= \left(\sum\sum_{i \neq j} x_i x_j - 2\sum_{i=1}^{n} x_i + 1 \right) + x_i^2 \end{aligned} \qquad ..(3.3)$$

By comparing with the Hopfield energy expression eqn.(2.4) stated in Chapter 2, the diagonal element of the connection weight matrix should equal -2. The stability condition requirement is broken. This is consistent to the experiment done by Wilson and Pawley [WP88], viz. the Hopfield network for solving TSP doesn't converge very often. From eqn. (3.1), it is easily noted that the diagonal element of the connection weight matrix (*self-reinforcement weighting*) equals -*C*. This account for the nonconverging cases. Even if with our modification, and eqn.(3.2) is used instead, the network instability remains. Some researchers solve the problem by adding energy difference calculation. In fact, the instability problem can be easily solved by a simple strategy, named here *"reshaping strategy"*. In the coming section, we will describe the strategy and explain why it works by simple geometric analysis of the energy profile. Then, we extend the strategy from quadratic to higher-order cases. With the generalized reshaping strategy, any formulation of generalized Hopfield network can be transformed to one with convergence guaranteed.

3.2.1 Quadratic problem and Hopfield network

We will consider the quadratic case first. Based on Theorem 2.1, to ensure the stability of Hopfield network, two conditions have to be satisfied:

i) <u>Symmetric condition, $T_{ij} = T_{ji}$:</u>

For a quadratic problem,

$$Q = \sum_{i}^{n} \sum_{j}^{n} a_{ij} x_i x_j + \sum_{i}^{n} b_i x_i \qquad ..(3.4)$$

it's well-known in linear algebra that eqn.(3.4) can be written as,

where A^s is an nxn symmetric matrix, with Q remaining unchanged. The symmetric matrix

$$Q = x^T \cdot A^s \cdot x + B \cdot x \qquad ..(3.5)$$

A^s can be obtained by averaging and can be expressed as:

$$a_{ij}^s = a_{ji}^s = \frac{a_{ij} + a_{ji}}{2} \qquad ..(3.6)$$

ii) <u>Diagonal element non-negative, $T_{ii} \geq 0$:</u>

An easy and sufficient way to ensure this condition is to reformulate the Hopfield network connection by removing the T_{ii} connection but an equivalent term, $T_{ii}/2$ is added to the bias term, I_i. The process may be appreciated from a geometric perspective.

At a particular instance, in the asynchronous model of Hopfield network, say, the ith neuron is being updated. If $T_{ii}<0$, Fig. 3.1 depicts a typical plot of the corresponding Hopfield energy verse the state variable of ith neuron, x_i. This is the case that will lead to network oscillation. When $x_i = 0$, referring to the gradient direction, the network will drive x_i to 1. When $x_i = 1$, the network again will drive x_i to 0. So, the restriction $T_{ii} \geq 0$ is to avoid this situation from happening.

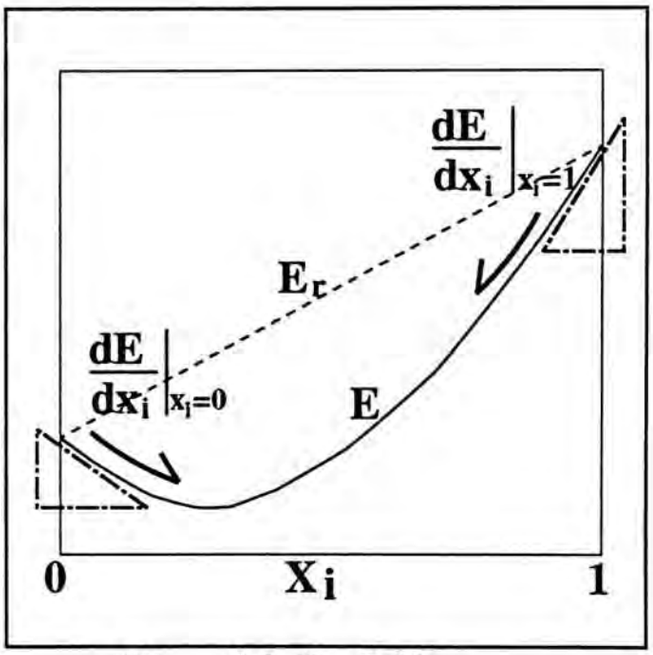

Figure 3.1: Hopfield energy plot for $T_{ii} < 0$

Suppose now, if we can fit a straight line through the points ($0,E(x_i=0)$) and ($1,E(x_i=1)$) The gradient of that line then will provide the right information on which state has lower energy and therefore it is desired to reshape the energy profile this way. Instead, we can make such a modification to the Hopfield energy function by letting $x_i^2 = x_i$, and based on the assumption that each neuron only takes on the value of zero or one. For the original Hopfield energy,

Applying the proposed modification, a new energy function, E_r is obtained.

$$E = -\frac{1}{2}\sum\sum_{i\neq j} T_{ij}x_i x_j - \frac{1}{2}T_{ii}x_i^2 - \sum_i I_i x_i \qquad ..(3.7)$$

$$E_r = -\frac{1}{2}\sum\sum_{i\neq j} T_{ij}x_i x_j - \sum_i \left(I_i + \frac{T_{ii}}{2}\right)x_i \qquad ..(3.8)$$

Now, the modified Hopfield network has zero self-feedback connections and the non-negative condition on the diagonal elements can always be satisfied. Note that the new bias for i^{th} neuron is now ($I_i + T_{ii}/2$). From eqn.(3.7) & (3.8), it is obvious that E and E_r will take on the same value when $x_i=\{0,1\}$ and E_r is linear with respect to x_i. Thus, the modification $x_i^2 = x_i$ leads to a straight line approximation, which is in fact the desired shape for the energy profile. As the energy profile has been reshaped, this strategy is named as *reshaping strategy*.

3.2.2 Higher-order case and reshaping strategy[1]

Hopfield network only deals with quadratic problem. If we intend to solve higher-order problem, obviously generalized Hopfield network is required. Recall from Chapter 2, Section 2.4 that energy function corresponding to generalized Hopfield network is,

$$\begin{aligned} E = &-\frac{1}{n}\sum_{i_1}\sum_{i_2}\cdots\sum_{i_n} T_{i_1 i_2 \ldots i_n} x_{i_1} x_{i_2} \ldots x_{i_n} - \frac{1}{n-1}\sum_{i_1}\sum_{i_2}\cdots\sum_{i_{n-1}} T_{i_1 i_2 \ldots i_{n-1}} x_{i_1} x_{i_2} \ldots x_{i_{n-1}} \\ &\cdots\cdots \quad -\frac{1}{2}\sum_{i_1}\sum_{i_2} x_{i_1} x_{i_2} - \sum_i I_i x_i \end{aligned} \qquad ..(3.9)$$

The network updating rule is given by,

$$\begin{aligned} x_i = f_h\Bigg[&\sum_{i_1}\sum_{i_2}\cdots\sum_{i_{n-1}} T_{i_1 i_2 \ldots i_{n-1} i} x_{i_1} \ldots x_{i_{n-1}} + \sum_{i_1}\cdots\sum_{i_{n-2}} T_{i_1 \ldots i_{n-2} i} x_{i_1} \ldots x_{i_{n-2}} \\ &\cdots\cdots \quad + \sum_{i_1} T_{i_1 i} x_{i_1} + I_i \Bigg] \end{aligned} \qquad ..(3.10)$$

given the value of $T_{i_1 i_2 \ldots i_k}$ is independent of the ordering of the index for k=1 to n. For example, for k=3, the ordering independence implies $T_{123} = T_{132} = T_{213} \ldots$. This condition is analogue to the symmetric requirement on the Hopfield network's connection weight matrix.

[1] This work has been submitted for consideration towards presentation in IJCNN '93

To achieve this condition, averaging process can be performed and is expressed as follows:

Let $<i_1,....,i_p>$ be the set containing all possible permutations of p chosen neuron index.

$$\forall\, s : s \in <i_1,....,i_p> \qquad \& \qquad \forall\, p : 0 < p \le n$$

$$T_{i_1,....,i_p}^{(new)} = T_{<i_1,....,i_p>} = \frac{\sum_{s \in <i_1,....,i_p>} T_s^{(old)}}{size(<i,....,i_p>)} \qquad ...(3.11)$$

As the energy function is now a higher-order polynomial, the shape of the energy profile is governed by a lot of coefficients and can be very complicated. Fig. 3.2 shows a typical example. Definitely, updating rule based on eqn.(3.10) will not be a correct one in judging which state of the neuron has a lower energy value for such type of energy function and thus the convergence cannot be guaranteed. We propose to apply the similar reshaping strategy as to the Hopfield energy function by letting $x_i^k = x_i$ for k being an integer. The energy function eqn.(3.9) will become (Appendix III),

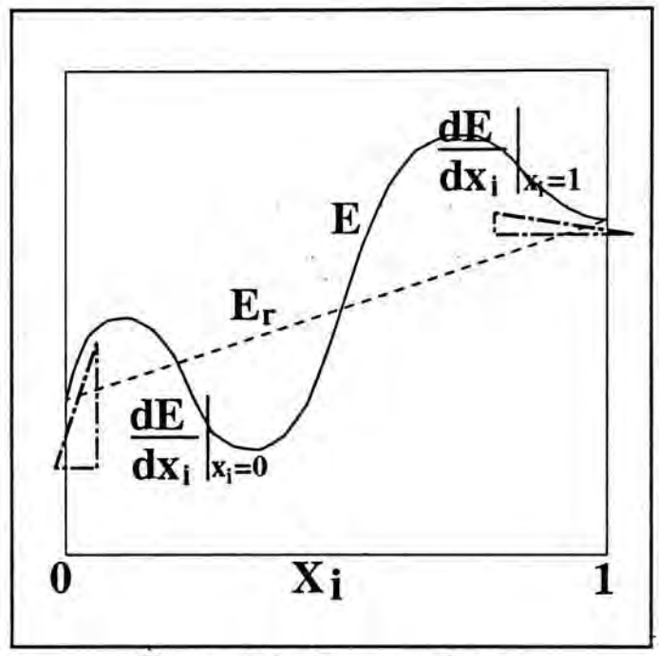

Figure 3.2: Energy plot for higher-order network

$$E_r = -\frac{1}{n}\sum\sum_{i_1 \ne i_2 .... \ne i_n}....\sum T_{i_1 i_2 .... i_n} x_{i_1} x_{i_2} .... x_{i_n}$$

$$-\sum_{m=1}^{n-1}\sum\sum_{i_1 \ne i_2 .... \ne i_m}....\sum \sum_{\lambda_1=1}^{n-m+1}\sum_{\lambda_2=1}^{n-\lambda_1-m+2} .... \sum_{\lambda_{m-1}=1}^{n-1-\sum_{z=1}^{m-2}\lambda_z} \frac{(n-1)!}{(m)!}\cdot\frac{1}{\lambda_1!\lambda_2!....\Lambda_m^n!} T_{i_1^{\lambda_1} i_2^{\lambda_2} .... i_m^{\Lambda_m^n}} x_{i_1} x_{i_2} .... x_{i_m}$$

$$-\frac{1}{n-1}\sum\sum_{i_1 \ne i_2 .... \ne i_{n-1}}....\sum T_{i_1 i_2 .... i_{n-1}} x_{i_1} x_{i_2} .... x_{i_{n-1}}$$

$$-\sum_{m=1}^{n-2}\sum\sum_{i_1 \ne i_2 .... \ne i_m}....\sum \sum_{\lambda_1=1}^{n-m}\sum_{\lambda_2=1}^{n-m+1-\lambda_1} .... \sum_{\lambda_m=1}^{n-2-\sum_{z=1}^{m-2}\lambda_z} \frac{(n-2)!}{(m)!}\cdot\frac{1}{\lambda_1!\lambda_2!....\Lambda_m^{n-1}!} T_{i_1^{\lambda_1} i_2^{\lambda_2} .... i_m^{\Lambda_m^{n-1}}} x_{i_1} x_{i_2} .... x_{i_m}$$

$$- \frac{1}{2}\sum_{i_1 \neq i_2}\sum T_{i_1 i_2} x_{i_1} x_{i_2} - \sum_{i_1} \frac{1!}{0!} \cdot \frac{1}{2!} T_{i_1 i_1} x_{i_1}$$

$$- \sum_{i_1} I_{i_1} x_{i_1} \qquad ..(3.12)$$

where $T_{i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}}$ is connection weight with λ_1 index i_1, followed by λ_2 index i_2, ... followed by λ_m index i_m & $\Lambda_m^k = k - \sum_{z=1}^{m-1} \lambda_z$

By grouping the terms with same number of variables, E_r becomes,

$$E_r = - \frac{1}{n}\sum_{i_1 \neq i_2 \ldots \neq i_n} \ldots \sum T_{i_1 i_2 \ldots i_n} x_{i_1} x_{i_2} \ldots x_{i_n}$$

$$- \frac{1}{n-1}\sum_{i_1 \neq i_2 \ldots \neq i_{n-1}} \ldots \sum \left(T_{i_1 i_2 \ldots i_{n-1}} + \sum_{\lambda_1=1}^{2} \sum_{\lambda_2=1}^{3-\lambda_1} \ldots \sum_{\lambda_{n-2}=1}^{n-1-\sum_{z=1}^{n-3}\lambda_z} \frac{(n-1)!}{(n-2)!} \cdot \frac{1}{\lambda_1! \ldots \lambda_{n-2}! \Lambda_{n-1}^n!} T_{i_1^{\lambda_1} \ldots i_{n-1}^{\Lambda_{n-1}^n}} \right) x_{i_1} \ldots x_{i_{n-1}}$$

$$\vdots$$

$$- \frac{1}{j}\sum_{i_1 \neq i_2 \ldots \neq i_j} \ldots \sum \left(T_{i_1 i_2 \ldots i_j} + \sum_{k=j+1}^{n} \sum_{\lambda_1=1}^{k-j+1} \sum_{\lambda_2=1}^{k-\lambda_1-j+2} \ldots \sum_{\lambda_{j-1}=1}^{k-1-\sum_{z=1}^{j-2}\lambda_z} \frac{(k-1)!}{(j-1)!} \cdot \frac{1}{\lambda_1! \ldots \lambda_{j-1}! \Lambda_j^k!} T_{i_1^{\lambda_1} \ldots i_j^{\Lambda_j^k}} \right) x_{i_1} \ldots x_{i_j}$$

$$\vdots$$

$$- \sum_i \left(I_i + \frac{T_{i_1^2}}{2} + \frac{T_{i_1^3}}{3} + \ldots + \frac{T_{i_1^n}}{n} \right) x_i \qquad ..(3.13)$$

which determines the connection weights of the "reshaped" network. According to eqn.(3.13), it is noted that every terms in E_r is linear with respect to x_i and partial derivative of E_r with respect to x_i will then be independent of x_i. Therefore, imposing our strategy leads to an approximation of energy profiles by straight lines as illustrated in Fig. 3.2. The "reshaped" network can now be guaranteed to converge towards a local minimum situated at the corners of $[0,1]_n$ hypercube according to Theorem 2.4 in Chapter 2.

3.2.3 Numerical Example

The reshaping strategy is illustrated through a numerical example. Assume a combinational programming with the function to be optimized being defined by,

$$f(x) = ax_1^3x_2 + bx_1^2x_2^2 + cx_1x_2^3 + dx_1^3 + ex_2^3 + fx_1^2 + gx_2^2 + ix_1 + jx_2 \quad ..(3.14)$$

Obviously, it's a higher-order problem with 2 independent variables. According, a higher-order Hopfield network with two neurons of McCulloch and Pitts' model is required. Comparing with generalized Hopfield energy function eqn.(3.9) and applying eqn.(3.11), the connection weight assignment can be obtained as eqn.(3.15).

$$T_{1^32} = T_{\begin{pmatrix}1112\\1121\\1211\\2111\end{pmatrix}} = (-4)\frac{a}{4} = -a$$

$$T_{1^22^2} = T_{\begin{pmatrix}1122\\1212\\1221\\2211\\2121\\2112\end{pmatrix}} = (-4)\frac{b}{6} = \frac{-2b}{3}$$

$$T_{12^3} = T_{\begin{pmatrix}1222\\2122\\2212\\2221\end{pmatrix}} = (-4)\frac{c}{4} = -c$$

$$\begin{aligned} T_{111} &= -3d \\ T_{222} &= -3c \\ T_{11} &= -2f \\ T_{22} &= -2g \\ I_1 &= -i \\ I_2 &= -j \end{aligned} \quad ..(3.15)$$

Following the reshaping strategy of eqn.(3.13), the new connection weight assignment is,

$$\begin{aligned} T'_{\left\{\begin{smallmatrix}12\\21\end{smallmatrix}\right\}} &= \frac{3!}{(2-1)!}\frac{1}{3!1!}T_{1^32} + \frac{3!}{(2-1)!}\frac{1}{3!1!}T_{12^3} + \frac{3!}{(2-1)!}\frac{1}{2!2!}T_{1^22^2} \\ &= T_{1^32} + T_{12^3} + \frac{3}{2}T_{1^22^2} \\ &= -(a+b+c) \end{aligned}$$

$$I_1' = I_1 + \frac{T_{11}}{2} + \frac{T_{111}}{3} = -(i+f+d)$$

$$I_2' = I_2 + \frac{T_{22}}{2} + \frac{T_{222}}{3} = -(j+g+e) \quad ..(3.16)$$

and the corresponding energy function is,

$$E_r = (a+b+c)\,x_1x_2 + (d+f+i)\,x_1 + (e+g+j)\,x_2$$

which is identical to the original function, eqn.(3.14), with x^k being replaced by x. Fig. 3.3a-b illustrate the original and reshaped energy profiles. It can be noted that all the rippling features are removed after the reshaping step.

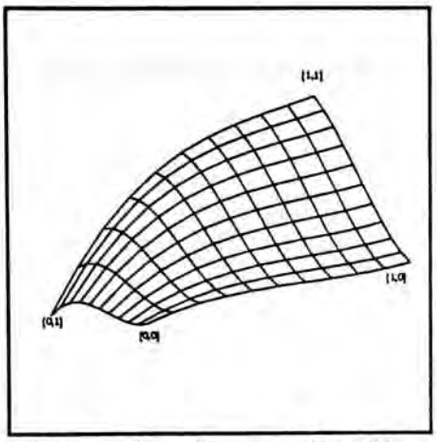

Fig 3.3a: Energy Profile without reshaping

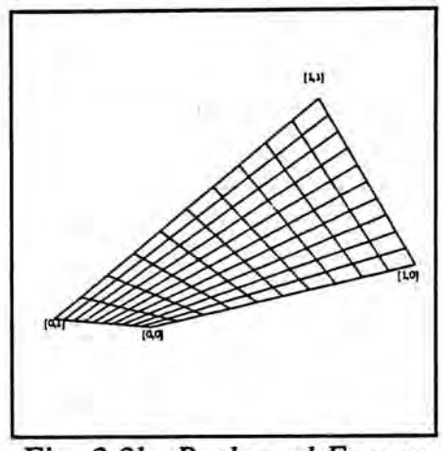

Fig. 3.3b: Reshaped Energy Profile

3.3 Local minimum limitation and existing solutions in the literature

After going through the neural network formulation procedure described in Section 3.1 and performing the reshaping strategy in Section 3.2, what we have now is a proper neural network formulation describing the problem and the neural network with the corresponding formulation will converge to a stable state, which is equivalent to a local minimum state and representing a valid/meaning solution. The most undesirable wording of the above statement is "local minimum". In fact, the neural network formulation we adopted here will result in an energy profile with a great set of local minimum states. Among all of them, only a small amount are corresponding to the global minimum states. In order to yield the optimal solution, the initial state of the network has to be put near enough to a global minimum state. However, Hopfield network was proposed with the intention of providing an optimum seeking method without a prior knowledge on the problem itself. So, we should have no idea of where to put the initial state and thus the optimal solution can't be guaranteed.

To escaping from the local minimum state, existing solutions in the literature can be categorized into two main streams. One stream solves the problem by using stochastic means, like *Simulated Annealing* while the other stream modifies the Hopfield energy profile, like, *Mean Field Annealing, Hardware Annealing, Adaptively changing neural network* and *Correcting Current Method.*

3.3.1 Simulated Annealing (SA) [KGV83] [HSA84]

Simulated Annealing method applied in optimization problem is originated from statistical mechanics. Statistical mechanics is the central discipline of condensed matter physics, a body of methods for analyzing aggregate properties of the large numbers of atoms to be found in samples of liquid or solid matter. Viewing the atoms as a system, each configuration, defined by the set of atomic positions, $\{r_i\}$ of the system is weighted by its Boltzmann probability factor, $exp(-E(\{r_i\})/k_BT)$, where $E(\{r_i\})$ is the energy of the configuration, k_B is Boltzmann's constant and T is temperature. The probability distribution of different configuration's energy is shown in Fig. 3.4. It has been shown that to have a configuration with lower energy state, the temperature should be low.

In practical context, low temperature alone is not a sufficient condition for finding ground states of matter. Experiments that determine the low-temperature state of a material, for example, growing a single crystal from a melt, are done by careful annealing, first melting the substance, then lowering the temperature slowly and spending a long time at temperatures in the vicinity of the freezing point. If this is not done, and the substance is allowed to get out of equilibrium, the resulting crystal will have many defects, or the substance may form a glass, with no crystalline order and only metastable, a locally optimal structure (local minimum state).

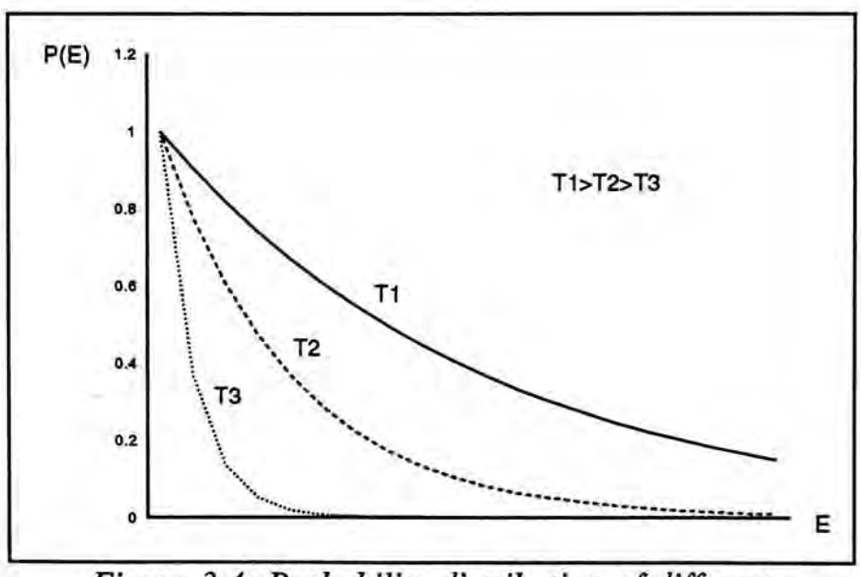

Figure 3.4: Probability distribution of different system configuration at different temperature

To apply the concept to solve Hopfield network's local minimum problem, a stochastic neural network model can be adopted and the neuron's transfer function will then be,

$$x_i^{new} = \begin{cases} 1-x_i^{old} & \text{if } \delta H < 0 \\ 1-x_i^{old} & \text{if } \delta H > 0 \ \& \ e < \exp(-\delta H/T) \\ x_i^{old} & \text{if otherwise} \end{cases}$$

where $\delta H = -\left[\sum_j T_{ij} x_j^{old} + I_i \right] (1-2x_i^{old})$,

T = annealing temperature

e = a random number between 0 and 1

T_{ij} = connection weight between i^{th} and j^{th} neurons

I_i = external bias for i^{th} neuron

x_i = state variable of ith neuron

At the very high temperature, the dynamic of the network will become almost like a random walk. As temperature decreases, the probability that the dynamic of the network follows the gradient information of the Hopfield energy profile increases. The network will change back to a gradient system when the temperature is sufficiently low. If the cooling schedule is slow enough, it can be proved that the global minimum state can be obtained in infinite time. However, the main disadvantage of Simulated Annealing is the extensive computational time. Using a faster cooling schedule will sacrifice the quality of the solution. Nevertheless, still a lot of researchers [Hua92] [WZX92] [TGD91] use it when optimal solution is wanted. The main reason most probably is its simplicity. If we can tolerate waiting for a really long time, setting a very slow schedule, then the system can converge to a solution very close to the global minimum state. To be a wiser user, the cooling schedule can be set wisely according to the problem, eg. starting the process at *critical temperature* when the network almost starts to be dominated by the gradient information instead of random walk. However, to estimate that critical temperature, analysis has to be performed and then the neural network is no longer an effective problem independent black box for optimization.

3.3.2 Mean Field Annealing (MFA) [BMMS89]

Mean Field Annealing is in fact a speed-up version of SA using mean field approximation, a simple approximation of the behaviour of system particles in equilibrium. For SA, at a particular temperature, we have to wait for a period long enough till an equilibrium is established and then the temperature is lowered again and another equilibrium has to be waited for. Using mean field approximation, this process can be approximated by a relaxation phase with the dynamic equation;

$$\frac{d\langle u_i\rangle}{dt} = \sum_{j=1}^{N} T_{ij} \langle v_i\rangle + I_i - \langle u_i\rangle$$

$$\langle v_i\rangle_{t+1} = [\ 1 + \exp(-\langle u_i\rangle_{t+1}/T\ )\ ]^{-1}$$

where $\langle x\rangle$ stands for equilibrium average of the entity x,

T is the annealing temperature,

$\langle v_i\rangle$ is the ith spin/particle state average of the system,

$\langle u_i\rangle$ is the mean field affecting spin v_i.

There are two interesting findings. Firstly, the relaxation phase is much faster than SA to come to equilibrium. That's why MFA is a speed-up version of SA. Secondly, the relaxation phase dynamic equation is exactly the same as that of continuous Hopfield network with gain varied by the T parameter. The analogy is revealed in Table 3.1.

MFA Algorithm	Hopfield Neural Network
The mean field	The neuron input
The spin average	The neuron output
The Boltzmann probability	The sigmoid amplifier
Temperature	Reciprocal of neuron gain

Table 3.1: A Comparison of quantities found in Hopfield Network and MFA Algorithm

So, the cooling schedule becomes a schedule for varying the neuron gain and the whole system can be implemented by a continuous Hopfield network. The concept of varying

the neuron gain to improve the quality of the solution obtained by Hopfield network has been mentioned by J.J. Hopfield in 1985 to solve TSP. But the relationship with mean field theory was firstly drawn in [BMMS89]. Also, Lee *et al.* [LS91a] suggested a scheme called Hardware Annealing, which is basically the same as MFA.

The MFA model mentioned above is the most original one. There is a variation of the model proposed in [PS89], which can enforce some exclusivity constraints without using penalty method in the problem formulation step. The modification is as simple as introducing normalization division operation among a group of neurons where only one of them is to be activated for a final valid solution. Besides, critical temperature can also be estimated to cut down the converging time by similar argument mentioned for SA. This algorithm has been applied to solve graph partition problem [BM90]. Although it has a lot of advantages, it remains to be an annealing based one. Cooling schedule determination and critical temperature estimation are all problem dependent, the same drawback of SA.

3.3.3 Adaptively changing neural network [XT91]

The intuitive idea of this algorithm is just filling up the local minimum valley. To have this being done, a higher-order term which will have a large value only at the local minimum state is added to the Hopfield energy. As it is a higher-order term, generalized Hopfield network model is necessary. Consider a network with ten neurons and the local minimum state to escape is {1010011001}. Then the energy term to be added is $kx_1x_3x_6x_7x_{10}$ where k is a large enough constant determined heuristically. The number of four-order connection needed will be of O(5!). So, the network's associated energy profile, and thus the structure is changing adaptively during the optimum searching process. As the number of connections to be added during the course of searching is large and unpredictable, this algorithm is only suitable for the type of the problem formulation used by Xu, where a prior knowledge on the type of local minimum states is used during the algorithm derivation. So, this is not a generic solution as the previous two.

3.3.4 Correcting Current Method [LS91]

Xu *et al.*'s adaptively changing neural network modifies the energy profile by higher-order term so that the local minimum valley can be filled. With a similar concept, Lee *et al.* [LS91b] suggested using correcting current method to fill the valley. Although the energy profile and thus the network architecture has been modified, its order remains the same. This simplicity makes this algorithm very attractive. However, to derive the correcting current terms, a prior knowledge on the local minimum states is again necessary. Although Lee *et al.* successfully built a neural-based AD converter hardware with global minimum guaranteed, for much more hard problem like TSP, where we have not much knowledge on the local minimum states, determination of the correcting current term will become very difficult. Lack of easy generalization of this method is its main shortcoming.

3.4 Conclusions

The procedure of mapping a problem to neural network formulation is discussed and the guidance on how to introduce hard constraints into the formulation to avoid converging to some invalid state is described. Moreover, we proposed a "reshaping strategy" which can be used to transform any neural network formulation to a proper one such that convergence can always be guaranteed. Lastly, Hopfield network's local minimum problem is addressed and some existing methods with the ability to escape from local minimum are discussed.

Among the suggested remedies for Hopfield network's defect in the literature, MFA is recognized as the most effective and successful one so far, due to its reduced computational time compared with SA and problem-solving generality. However, it has to be accompanied with the step of getting a proper cooling schedule and estimating critical temperature for a particular problem. As this requires some advanced knowledge on the topic, MFA is not so easily plug into an application if ones who want to use it are lack of annealing background. In the following chapter, we propose a novel neural network, named *Tunneling network* with capability of escaping from local minimum to seek the optimal solution. Instead of following the old annealing method, a well-known global optimization technique called *tunneling algorithm* is adopted. This is a completely new approach to solve the local minimum problem

of Hopfield network and no advance knowledge is necessary for using it, which makes it a very handy tool in our neural network toolbox.

Chapter 4 A Novel Neural Network for Global Optimization - Tunneling Network

In this Chapter, a novel neural network with seeking optimal solution capability is proposed. The intuitive idea is originated from a well-known global optimization technique called "tunneling algorithm", which escapes from the local minimum state by adding pole there so that the nearby barriers can be tunneled through to seek a better solution. We will first describe the original tunneling algorithm in Section 4.1 and then in Section 4.2, a proposed tunneling network, using pole shifting technique, will be described with analysis on network stability and global convergence property given in Section 4.3. The effect of varying the pole expression is studied and another Tunneling network, which is adaptive in nature, is proposed in the subsection 4.3.3.3. Simulation experiments are performed for the efficiency and effectiveness comparison with Simulated Annealing using TSP as the benchmark test. Promising results are obtained. Furthermore, for the enhancement of the proposed network's performance, a network decomposition strategy is attempted with two different partition schemes. All the results will be illustrated in Section 4.4. Lastly, suggestion on hardware implementation of the two proposed Tunneling networks are given in Section 4.5.

4.1 Tunneling Algorithm:

Tunneling algorithm is one of the algorithms using penalty methods for global optimization. 1-dimensional version of the algorithm was first proposed by A.V.Vilkov *et al.* [VZS75]. Later on, A.V.Levy *et al.* [LM85] named it "Tunneling" and generalized it to cover also the multidimensional case.

4.1.1 Description of Tunneling Algorithm

Tunneling algorithm consists of two phases, namely, **minimization** phase and **tunneling** phase. The two phases are used sequentially to approach the global minimizer of $f(x)$, the function to be optimized. In the minimization phase, for a given starting point x^0, any

minimization algorithm with a local descent property on $f(x)$ can be used to find a local minimum of $f(x)$, say at x^*. Then, in the tunneling phase, an auxiliary function $T(x)$ is defined, where $T(x)$, the **tunneling function**, is a scalar function with continuous first derivatives, whose zero-set coincides with the set where $f(x) = f(x^*)$. The objective of defining the function is to seek a new point x^0, starting at any point in a neighborhood of x^*, such that $T(x^0) \leq 0$.

4.2.2 Tunneling Phase

The minimization phase can be achieved by any existing local minimization schemes, such as, gradient descent, conjugate gradient or Newton's method. The most mystery part of tunneling algorithm is the tunneling phase.

After the minimization phase, assume x reaches x^* and $f(x^*)$ is the corresponding local minimum value. The tunneling phase will then start to find a point $x^0 \in \Omega$ such that

$$f(x^0) \leq f(x^*)$$

$$x^0 \neq x^*$$

where Ω is the domain of $f(x)$

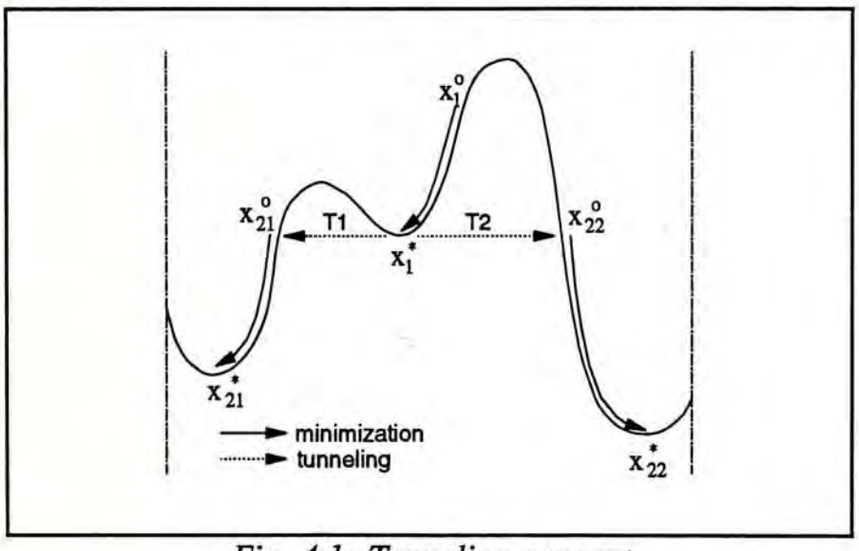

Fig. 4.1: Tunneling concept

To illustrate the concept, suppose the initial point is x_1^o as shown in Fig. 4.1. In minimization phase, the point will move until x_1^* is reached. Then, in tunneling phase, the tunneling path can be either T1 or T2 and { x_{21}^o, x_{22}^o } are the set of points targeted for.

When either of them is reached, switching back to minimization phase will move the point to x_{21}^* or x_{22}^*. And by repeating the process, a better minimum can always be found sequentially.

To perform the tunneling phase, a tunneling function is defined. Before coming to the tunneling function used in the original paper, a simple example can be used to illustrate the main idea. In fact, tunneling phase of tunneling algorithm is achieved by pole-adding scheme. Say, x^* is a local minimizer of $f(x)$. Adding a pole with a suitable strength to $f(x)$ at x^* can destroy the local minimum valley. Define an auxiliary function $T(x)$ such that

$$T(x) = \frac{f(x) - f(x^*)}{\left[(x-x^*)^T(x-x^*)\right]^{\lambda}} \qquad ..(4.1)$$

The numerator is defined such that whenever $T(x) < 0$ is detected, the tunneling phase is finished, i.e. a better local minimizer has been found. The denominator is a pole at $x = x^*$. For the pole strength λ, if $f(x)$ is polynomial, λ can be set deterministically to half of the order of $f(x)$. If $f(x)$ is a highly non-linear function, like exponential, λ determining step has to be done heuristically and it will be described later.

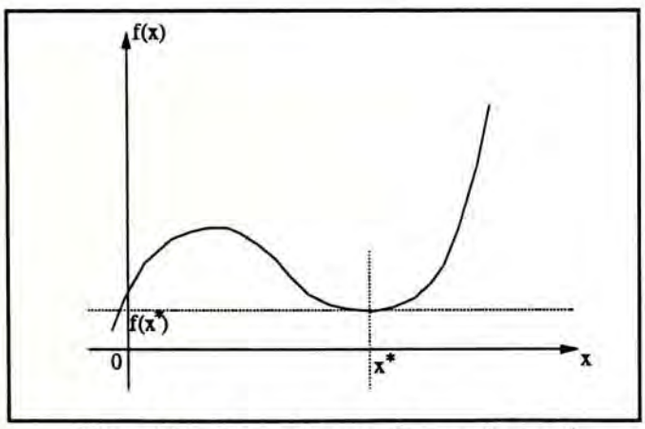

Fig. 4.2: Function to be optimized

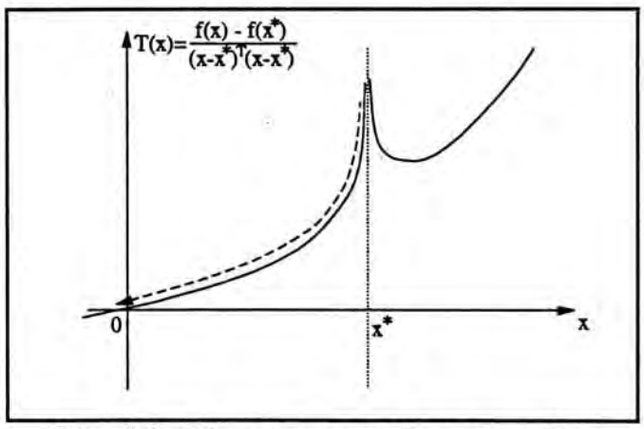

Fig. 4.3: The corresponding tunneling function of Fig. 4.2

After defining the key function for the algorithm, the basic idea of the algorithm can be well illustrated. Referring to Fig. 4.2 & Fig. 4.3, starting from a point in the neighborhood

of a local minimizer x^*, performing local minimization or zero-seeking step on $T(x)$ can "tunnel" through the neighboring barrier of $f(x)$ around x^*. That's why $T(x)$ is in fact named "tunneling function" and the algorithm called "tunneling".

In order to cope with more generalized and complicated function, A.V.Levy [LM85] defined a much more complicated tunneling function,

$$T(x) = \frac{f(x) - f(x^*)}{\left(\prod_{i=1}^{l} [\ (x-x_i^*)^T(x-x_i^*)\ ]^{\eta_i} \right) [\ (x-x_m)^T(x-x_m)\ ]^{\lambda_0}} \qquad ..(4.2)$$

The parameters to be determined for the tunneling function are [l, (x_i^*, i= 1, 2, ... l), (η_i, i= 1, 2, ... l), x_m, λ_0, $f(x^*)$]

i) $f(x^*)$: With this parameter, we accomplish our objective of tunneling below irrelevant local minima even if we do not know how many they are nor their locations.

ii) η_i and x_i^* : (i=1, 2, ... l)

1) assume there is one local minimum at x_1^*. Then the tunneling function is

$$T(x) = \frac{f(x) - f(x_1^*)}{[\ (x-x_1^*)^T(x-x_1^*)\ ]^{\lambda_1}} \qquad ,where\ x = x_1^* + \varepsilon \qquad ..(4.3)$$

2) The correct value of λ_1 is found iteratively. Starting from $\lambda_1 = 1$, it is updated iteratively until $T(x_1^*+\varepsilon)^T \cdot \Delta x < 0$,provided $\varepsilon^T \Delta x > 0$. ε is a random vector with $\|\varepsilon\| << 1$ and Δx is produced by the tunneling phase such that $T(x^0+\Delta x) \leq 0$. A.V.Levy used Restoration Algorithm to produce it.

If the above condition can't be satisfied, increase λ_1 by $\Delta\lambda_1$ until the above descent property is satisfied. Then, η_1 is given by eqn.(4.4):

$$\eta_1 = \begin{cases} 0 & \text{if } \gamma \geq 1 + \varepsilon_2 \\ \lambda_1 & \text{if } \gamma \leq 1 - \varepsilon_2 \\ (\frac{\lambda_1}{2})[(1+\frac{(1-\gamma)}{\varepsilon_2}] & \text{if } 1-\varepsilon_2 \leq \gamma \leq 1+\varepsilon_2 \end{cases} \quad ..(4.4)$$

where $\gamma = \| x - x^* \|$ and ε_2 is a small prescribed number.

For some functions, the denominator might become very large when $\|x - x^*\| >> 1$, forcing the tunneling function to become very flat and close to zero and thus slowing down the convergence of the tunneling algorithm. That's why the switching operation is incorporated so that the pole far away from the current position is switched off.

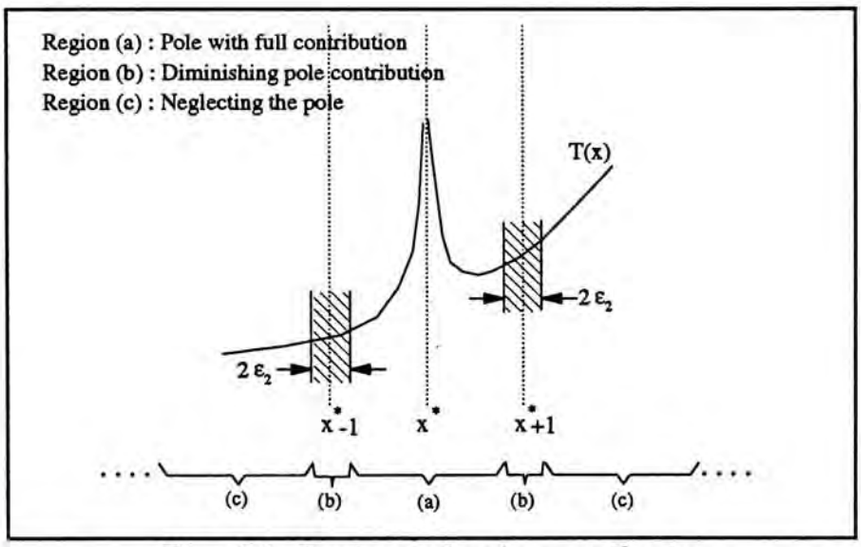

Fig. 4.4: Regions of pole contribution

3) l is the number of local minima found at the functional level equal $f(x^*)$. As long as a new local minimum found has a lower value than the previous one, it is reset to 1.

iii) Determination of x_m and λ_0:

The main function of these two parameters is to cancel out any undesirable relative minimum that the tunneling function $T(x)$ might have. When a local minimum is detected during tunneling (The detection is achieved by changing sign of the

gradient of consecutive paths), a moving pole x_m will be assigned by eqn.(4.5).

$$x_m = \begin{cases} \hat{x} & ,if \|\hat{x} - x\| < 1 \\ \xi\,\hat{x} + (1-\xi)x & ,if \|\hat{x} - x\| \geq 1 \end{cases} \qquad ..(4.5)$$

$$,where\ 0 < \xi \leq 1 \quad such\ that \quad \|x - x_m\| < 1$$

x : *present point*

$\hat{x}$: *previous point*

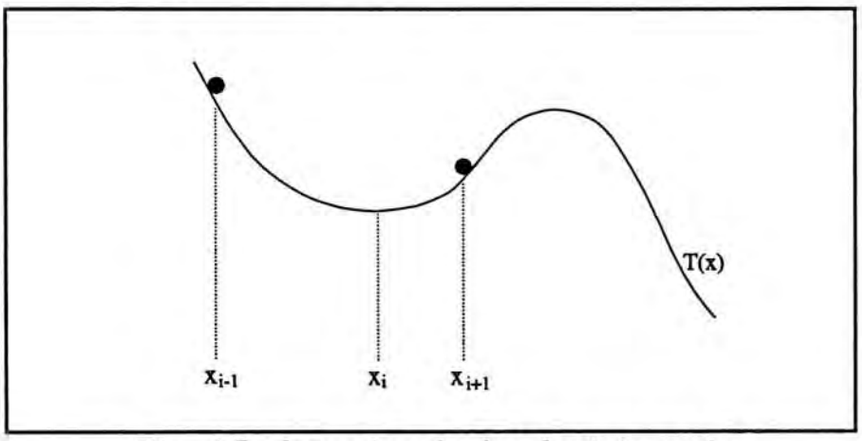

Fig. 4.5: determine the local minimum in tunnelling phase

For λ_0, which determines the strength of the pole $(x-x_m)^T(x-x_m)$, assume $x_{i-1} \rightarrow x_i \rightarrow x_{i+1}$. Define $u = (x_i - x_{i-1})^T \cdot (x_{i+1} - x_i)$. If $u > 0$, retain the value of λ_0. Otherwise, $\lambda_0 = \lambda_0 + \Delta\lambda_0$ until $u > 0$.

To ensure x will leave the attraction basin of the detected local minimum, a heuristic rule was proposed :

1) compute $\Delta x(\lambda_0)$ and $\Delta x(0)$ for current value of λ_0

2) $$\lambda_0 = \begin{cases} 0 & if\ \Delta x^T(0) \cdot \Delta x(\lambda_0) > 0 \\ \lambda_0 & if\ \Delta x^T(0) \cdot \Delta x(\lambda_0) \leq 0 \end{cases} \qquad ..(4.6)$$

The features of the tunneling function can be summarized by the following diagram:

- for zero crossing detection

$$T(x) = \frac{f(x) - f(x^*)}{\left(\prod_{i=1}^{l}\left[(x - x_i^*)^T(x - x_i^*)\right]^{\eta_i}\right)\left[(x - x_m)^T(x - x_m)\right]^{\lambda_o}}$$

- storage of local minimizers with functional value equal $f(x^*)$

- storage of current local minimizer encountered during tunneling and it's movable.

4.2 A Neural Network with tunneling capability --- Tunneling network:

To realize the tunneling algorithm, both minimization and tunneling phases have to be implemented. For minimization, generalized Hopfield network can do the job effectively until a local minimum state is converged. For tunneling phase, a new neural network model, which we name throughout the thesis "Tunneling network" is required. Before deriving the architecture of the Tunneling network, we have to specify clearly the network specifications.

4.2.1 Network Specifications:

S.1) Neuron model and the problem domain:

McCulloch and Pitts neuron model is adopted and each neuron can only have state value in the set $\{0,1\}$. If the size of network is n, the domain of the problem solved by the network is the corners of the unit-hypercube of n dimensional. Thus, that is, only combinatorial optimization problem is addressed.

S.2) Asynchronous updating:

There is only one neuron picked up randomly and updated at a time.

S.3) Hopfield network association and tunneling capability:

Each tunneling network is associated with a generalized Hopfield network. The former one minimizes the tunneling function defined corresponding to the Hopfield energy function of the latter so that tunneling network should be able to escape from local minimum state of the associated Hopfield energy function to approach to a lower energy state.

S.4) Hardware simplicity:

The derived architecture should be simple enough to make the hardware realization possible.

With the defined specifications, the architecture of the tunneling network, which is characterized by the network's dynamic equation (dynamic system theory) or updating rule (neural network theory), can be derived. For the updating rule of generalized Hopfield

network, as the required dynamic of the network is to minimize the corresponding Hopfield energy function, partial derivative information of the energy function is adopted. However, the convergence of the network can only be guaranteed with satisfaction of the conditions of $T_{ii} \geq 0$ and $T_{ij} = T_{ji}$ for a 1st order Hopfield network. A simple reshaping strategy, proposed in Chapter 3, Section 3.2, can be employed to modify the shape of the Hopfield energy function and guarantee network convergence. The strategy can also be extended to n^{th} order Hopfield network. However, analogous strategy is not so easily obtained for tunneling network. So, instead of partial derivative, energy difference between adjacent states is used as the information for updating rule derivation.

4.2.2 Tunneling function for Hopfield energy function and the corresponding updating rule:

According to the specifications stated in the last section, certain simplifications on the original tunneling function proposed by A.V.Levy *et al.* can be made.

C.1) Switching function of the original tunneling function for avoidance of slow convergence can be neglected as each updating step can only be one which is the result of S.1 and S.2.

C.2) To meet the specification of hardware simplicity, only the movable pole is left in the tunneling function. Whenever a local minimum state is found during the tunneling phase, simply shift the pole there.

Based on the above simplification, the tunneling function we use is,

$$T(x) = \frac{E(x)-E(x^*)}{\left(\sum_{i=1}^{m}\left(x_i - x_i^*\right)^2\right)^{\lambda}} \qquad ..(4.7)$$

where λ is the pole strength, E(x) is Hopfield energy function and x^* is the latest found local

minimum state.

Assume current value of x_i is x_i^c. Then, the other possible value of x_i is $1\text{-}x_i^c$. The tunneling energy difference due to the change in state value of the ith neuron equals,[1]

$$\Delta T(x)\,|_{x_i^c} = T(x)\,|_{1-x_i^c} - T(x)\,|_{x_i^c}$$

$$= \frac{E(x)\,|_{1-x_i^c} - E(x^*)}{\left(\sum_{k\neq i}\left(x_k - x_k^*\right)^2 + \left(1-x_i^c - x_i^*\right)^2\right)^\lambda} - \frac{E(x)\,|_{x_i^c} - E(x^*)}{\left(\sum_{k\neq i}\left(x_k - k_k^*\right)^2 + \left(x_i^c - x_i^*\right)^2\right)^\lambda}$$

$$= \frac{1}{D}\left[\left(\sum_{k\neq i}(x_k - x_k^*)^2 + (x_i^c - x_i^*)^2\right)^\lambda \left(E(x)\,|_{1-x_i^c} - E(x^*)\right)\right.$$

$$\left. - \left(\sum_{k\neq i}(x_k - x_k^*)^2 + (1-x_i^c - x_i^*)^2\right)^\lambda \left(E(x)\,|_{x_i^c} - E(x^*)\right)\right]$$

$$= \frac{1}{D}\left[\left(\sum_{k\neq i}(x_k - x_k^*)^2 + (x_i^c - x_i^*)^2\right)^\lambda \left(E(x)\,|_{x_i^c} - E(x^*) + \left.\frac{\partial E(x)}{\partial x_i}\right|_{x_i^c} \Delta x_i\right)\right.$$

$$\left. - \left(\sum_{k\neq i}(x_k - x_k^*)^2 + (1-2x_i^c)(1-2x_i^*) + (x_i^c - x_i^*)^2\right)^\lambda \left(E(x)\,|_{x_i^c} - E(x^*)\right)\right]$$

The validity of last step requires the assumption that $\frac{\Delta E}{\Delta x_i} = \frac{\partial E}{\partial x_i}$ which can be achieved by the reshaping strategy proposed in Section 3.2.

$$= \frac{1}{D}\left[\sum_k (x_k - x_k^*)^2 \left.\frac{\partial E(x)}{\partial x_i}\right|_{x_i^c} - (1-2x_i^*)\left(E(x)\,|_{x_i} - E(x^*)\right)\right]\cdot \Delta x_i \qquad ..(4.8)$$

assuming $\lambda = 1$, $\Delta x_i = (1-x_i^c) - x_i^c = 1-2x_i^c$

The updating rule for the tunneling network then can be obtained as:

---

1 $\Delta T(x)\,|_{x_i^c}$ *means* $\Delta T(x)$ *evaluated for* $x_i = x_i^c$

$$x_i = f_h\left[\frac{-\Delta T(x)}{\Delta x_i}\right]$$
$$= f_h\left[\sum_k (x_k - x_k^*)^2 \frac{-\partial E(x)}{\partial x_i} + (1-2x_i^*)\big(E(x) - E(x^*)\big)\right] \quad ..(4.9)$$

where f_h is hard-limiting function and $1/D$ is dropped as $D > 0$.

According to the rule eqn.(4.9), a Tunneling network can be defined. As we haven't put any constraints on the order of the Hopfield energy function $E(x)$, the network model is basically a generalized one in terms of network order. If a tunneling network is associated with an nth order Hopfield network, its order is defined to be also n.

The most important simplification we made here is assuming $\lambda = 1$. This simplification can make the updating rule simple enough for implementation,especially when hardware is taken into consideration. Albeit this may sacrifice its effectiveness, it's still a feasible way for escaping from local minimum state due to the moving pole.

In the following sections, we will investigated the stability and convergence properties of the tunneling network and also will discuss more on the effect of varying the pole strength, λ.

4.3 Tunneling network stability and global convergence property:

Before studying the performance of a newly proposed network, its stability has to be guaranteed first.

4.3.1 Tunneling network stability:

In this subsection, we are going to show that Tunneling network is stable and will converge to a local minimum state.

Let H^n be the set containing all the corners of the close hypercube in $\mathbb{R}^n$ defined by $H^n \equiv \{ x = (x_1, \dots x_n)^T \in \mathbb{R}^n : x_i \in \{0,1\}, i=1,\dots n \}$

Definition 4.1:

Neighborhood of a state, x, denoted by $N(x)$, is defined as, $N(x) \equiv \{ y = (y_1, \dots y_n)^T \in H^n : \| y - y' \|_1 = 1 \}$, where $\| a - b \|_1$ is the L_1 norm between vector a and b.

Definition 4.2:

x^* is a **local minimum** point of a function, $f(x):H_n \rightarrow \mathbb{R}$ if $\forall\ x \in N(x^*), f(x^*) < f(x)$.

Theorem 4.1:

Tunneling network with asynchronous updating rule as eqn.(4.9) and neuron model being McCulloch & Pitts' will converge to a stable state, which is a local minimum state of the associated tunneling function, $T(x):H_n \rightarrow \mathbb{R}$ defined by eqn.(4.7).

Proof:

To prove the theorem, we first show that $\frac{\Delta T(x)}{\Delta t} \leq 0$. Then, we prove that $T(x)$ is bounded below and the guarantee of network stability follows. For the stable state being local minimum state, we show it by contradiction.

Lemma 4.1: Show that $\frac{\Delta T(x)}{\Delta t} \leq 0$.

Proof:

Based on the asynchronous model assumption and without loss of generality, assume at a particular time, t, only i^{th} neuron with its state denoted by x_i is to be updated. Let the current state value of x_i equals x_i^c.

Then, the updating rule of the tunneling network is

$$x_i(t+1) = f_h\left(\frac{-\Delta T(x(t))}{\Delta x_i(t)}\right)$$

The timing parameter is introduced for the convenience of the proof. After updating based on the rule above, the change of x_i can be,

$$\Delta x_i(t) = \begin{cases} 0 & \text{if } x_i(t+1)=x_i(t) \quad \ldots\ldots \quad \textit{case I} \\ 1-2x_i(t) \text{ or } 2x_i(t+1)-1 & \text{if } x_i(t+1)\neq x_i(t) \quad \ldots\ldots \quad \textit{case II} \end{cases}$$

Case I: As x_i is not changed,

$$x_i(t) = x_i(t+1) = f_h\left(\frac{-\Delta T(x(t))}{\Delta x_i(t)}\right)$$

$$\Rightarrow \frac{\Delta T(x)}{\Delta t} = 0$$

Case II:

$$\frac{\Delta T(x(t))}{\Delta t} = \frac{\Delta T(x(t)}{\Delta x_i(t)} \cdot \frac{\Delta x_i(t)}{\Delta t}$$

$$= \frac{\Delta T(x(t)}{\Delta x_i(t)} \cdot \frac{2x_i(t+1)-1}{\Delta t}$$

$$= \frac{\Delta T(x(t)}{\Delta x_i(t)} \cdot \frac{2f_h\left(\frac{-\Delta T(x(t))}{\Delta x_i(t)}\right)-1}{\Delta t}$$

$$= \frac{\Delta T(x(t)}{\Delta x_i(t)\Delta t} \cdot \left[2\left(1 - f_h\left(\frac{\Delta T(x(t))}{\Delta x_i(t)}\right)\right)-1 \right]$$

$$= - \frac{\Delta T(x(t)}{\Delta x_i(t)\Delta t} \cdot sign\left(\frac{\Delta T(x(t))}{\Delta x_i(t)}\right)$$

$$= - \left|\frac{\Delta T(x(t)}{\Delta x_i(t)}\right| \cdot \frac{1}{\Delta t} < 0$$

Combining case I and II, it's proved that $\frac{\Delta T(x(t))}{\Delta t} \leq 0$. ■

Lemma 4.2: The tunneling function defined by eqn.(4.7) with bounded λ is bounded below.

Proof:

Consider the tunneling function $T(x)$,

$$T(x) = \frac{E(x)-E(x^*)}{\left(\sum_{k=1}^{n}(x_k-x_k^*)^2\right)^{\lambda}}$$

$$\geq \frac{\min\left(E(x)-E(x^*)\right)}{\max\left(\sum_{k=1}^{n}(x_k-x_k^*)^2\right)^{\lambda}}$$

As $E(x)$ is bounded below by,

$$-\frac{1}{n}\sum|T_{i_1i_2\ldots i_n}| - \frac{1}{n-1}\sum|T_{i_1i_2\ldots i_n}|\ldots\ldots\frac{1}{2}\sum|T_{i_1i_2}| - \sum|I_i|$$

which is a finite value. It follows immediately that $T(x)$ is bounded below. ■

By Lemma 4.1 and 4.2, thus $T(x)$ is a Lypunov function and by dynamic system theory, the network converging to a stable point can be guaranteed.

Lemma 4.3: Stable states of a tunneling network are equivalent to the local minimum states of the network.

Proof:

Assume a stable state, say x^{**} is not a local minimum state, ie. there exists some i such that $T(x)|_{x_i^c} > T(x)|_{1-x_i^c}$ or $\Delta T(x)|_{x_i^c} < 0$. As the network will move to an adjacent state with lower energy, state value of x_i will change from x_i^c to $1\text{-}x_i^c$ after updating, which contradicts the fact that x^{**} is a stable point. This completes the proof of Theorem 4.1. ■

4.3.2 Global Convergence property:

As tunneling is just an algorithm capable of escaping from local minimum state, convergence towards global minimum state can't be guaranteed within finite time. However, analogous to the annealing strategy [RV91], tunneling network can be proved to converge to global minimum in probability sense. Ahead of the proof, we need a stochastic model to represent the dynamic of the network.

4.3.2.1 Markov chain model for Hopfield network

Due to the fact that Hopfield network has the state set S with the states initially uniformly distributed and the transition property from one state to another is independent of the transitions beforehand, it can be represented by a markov chain as the stochastic model for analysis. [Ios80]

Let the number of nodes of the network be n and the number of possible states will be 2^n. Thus, the corresponding markov chain model (later named as Hopfield markov chain) will have 2^n states. Throughout the whole subsection, the state set is denoted as X. The construction of the transition probability matrix is based on the fact that the probability of choosing one of the neighbors is $1/n$ and the probability of acceptance of the updating depends on the energy difference. (See Fig. 4.6) It is defined in Definition 4.3.

Definition 4.3: Let $N(x_i) = \{ x_{j1}, \ldots , x_{jn} \}$ be the set of neighbors of state x_i.

Probability of transition from state x_i to x_j is defined as:

$$p(x_i , x_j) = \begin{cases} \dfrac{1}{n} & , \textit{if } E(x_i) > E(x_j) \textit{ and } x_j \in N(x_i) \\ 0 & , \textit{otherwises} \end{cases}$$

$$p(x_i , x_i) = 1 - \sum_{x_j \in N(x_i)} p(x_i , x_j)$$

where $E(x)$ is the Hopfield energy of state x.

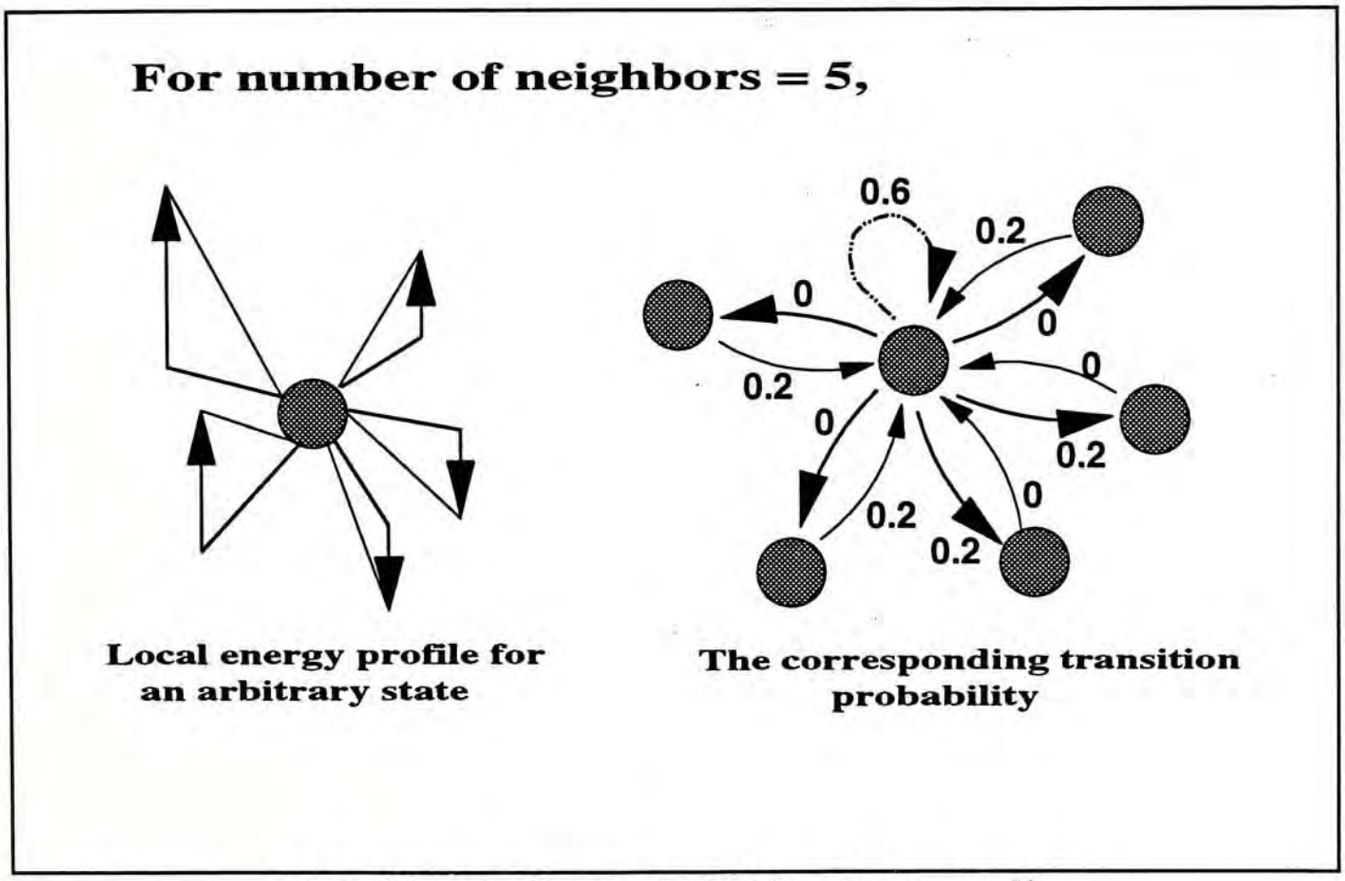

Fig. 4.6: state diagram derived from energy profile

4.3.2.2 Classification of the Hopfield markov chain

As it's known that whenever a local minimum state is reached in Hopfield network, it will get stuck at it. Let $X^* = \{ x_i^* \}$ be the set of local minimum states. The above fact can

be rewritten as:

$$p(x_i\ ,\ x_i) = 1 \qquad for\ x_i \in X^*$$

and thus the local minimum states are equivalent to the absorbing states of the Hopfield Markov Chain.

Existence of absorbing states implies that Hopfield markov chain is absorbing. As $\sum_i p(x_i,x_j) = 1$, the corresponding transition probability matrix is stochastic. Without loss of generality, number the states in such a way that the absorbing states are numbered first and then followed by the transient states in ascending order of Hopfield energy value. The transition probability matrix, denoted as P, will then be of canonical form:

$$\mathbf{P} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{R} & \mathbf{T} \end{bmatrix}$$

(M1)

where **I** is the identity matrix,
O is the zero matrix,
R is the matrix comprises the transition probability from transient states to absorbing states,
T is the matrix comprises the transition probability from transient states to transient states.

The global minimizers will then be at the top of the state list.

To calculate the probability matrix **A** = { $a(x_i,x_j^*)$ } which comprises the probability starting at transient state x_i ends up at absorbing state x_j^*, first generate the fundamental matrix, **N** of the associated markov chain as eqn.(4.10) and **A** matrix is then calculated by eqn.(4.11).

$$N = (\ I - T\ )^{-1} \qquad ..(4.10)$$

$$A = NR \qquad ..(4.11)$$

4.3.2.3 Markov chain model for tunneling network and its convergence towards global minimum

In the following, the perspective of viewing Tunneling network as another markov chain is adopted and a transition probability matrix for each tunneling phase in *averaging* sense is established. It reveals that tunneling phase can be modelled by an *ergodic* markov chain and the global convergence property can be investigated based on the model.

For Tunneling network, the lastly found local minimizer, say x_k^* (the kth element in the local minimum state list) in minimization phase will become a pole, ie. the transition probability matrix of the Hopfield markov chain will be modified as:

$$p(x_k^*\ ,\ x_k^*) = 0 \qquad \& \qquad p(x_k^*\ ,\ x_i) = \frac{1}{n} \qquad for\ x_i \in N(x_k^*) \qquad ..(4.12)$$

The transition probability matrix then becomes:

$$\mathbf{P_T} = \begin{bmatrix} \mathbf{I_k'} & \mathbf{O_k'} \\ \mathbf{R} & \mathbf{T} \end{bmatrix}$$

(M2)

where $\mathbf{I_k'}$ is equivalent to identity matrix **I**, except the kth diagonal element being zero

$\mathbf{O_k'}$ is equivalent to zero matrix **O**, except the kth row being not all zero.

The markov chain with transition probability matrix, P_T is describing the state transition phenomenon for a particular pole location. However, the pole is migrating during the tunneling phase. Thus, P_T will be varying. So, P_T is not a good stochastic model for describing the tunneling phase. However, we can use it to construct another markov chain to

model the whole tunneling process.

Consider a markov chain with the state set being X^*. As Transition from state x_i^* to x_j^* is only defined by $P_T|_{pole = x_i^*}$. From $A_T|_{pole = x_i^*}$ matrix, we can obtain the transition probability starting from x_i^* and terminating at x_j^*, when the pole is at x_i^*. Thus, based on the defined transition probability, a new markov chain is established. Whenever the network jumps a state, the pole will move to that state. So, this markov chain, named "tunneling markov chain", has the transition probability matrix, $\overline{P_T}$ defined by the formula,

$$\overline{p_T}(x_i^*,x_j^*) = a_T(x_i^*,x_j^*)|_{pole = x_i^*}$$

As tunneling markov chain with $\overline{P_T}|_{pole = x_i^*}$, $\{ x_j^* \in X^* \ \forall \ j \neq i \}$ are the only absorbing state,

$$\sum_{x_j \in X^*} a_T(x_i^*,x_j^*) = 1$$

Therefore, $\overline{P_T}$ is a stochastic matrix with only $\overline{p_T}(x_i^*,x_i^*) = 0$. All the other elements are positive.

It is easy to observe that for size of $X^* \geq 3$, $\overline{P_T}$ is a regular[2] stochastic matrix as all the elements of $\overline{P_T}^2$ are greater than zero. By the theory of markov chain[3] [Ios80], $\overline{P_T}^n$ will converge to a limiting probability matrix as n tends to infinity. Thus, tunneling markov chain is ergodic and all the states are recurrent states, ie. prob. that each state being reached for infinite time is 1. In other words, global minimum state being one of the recurrent states must be reached for sufficiently long time. Although the proof is in fact quite trivial, the tunneling

---

2 P is a regular matrix if there exists n st. $P^n > 0$, ie. every element of P^n is positive.

3 If P is a regular stochastic matrix, then P^n converges as $n \to \infty$ to a positive stable stochastic matrix $\Pi = e\,\pi^T$ where $e = (1,1,1...1)^T$, $\Sigma\,\pi_i = 1$ and $\pi_i > 0 \ \forall$ i.

markov chain framework can allow us to study the stochastic property of tunneling phase very effectively.

Lastly, the cases with size of $X^* \leq 2$ haven't been treated. For size of $X^* = 1$, tunneling is not necessary as global minimum state can be found by any local minimization technique. For size of $X^* = 2$,

$$\overline{P_T}^{2n+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \overline{P_T}^{2n} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Thus, it's oscillating. In fact, when the pole is added at one local minimum state, the probability that the only other local minimum state being reached is one. Therefore, global minimum state can be reached by just one tunneling transition without any pole moving required.

4.3.3 Variation of pole strength and its effect :

Variation of pole strength of tunneling network can affect the network's tunneling capability, or in other words, local minimum attraction basin destroying ability. To understand whether an added pole can completely destroy a local minimum attraction basin of the Hopfield network function, the profile of the energy function has to be investigated. Aiyer *et al.*[ANF90] and C.Chuan *et al.*[CMS91] performed analysis on the characteristics of the Hopfield energy profile for Travelling Salesman Problem(TSP) and Dynamic Programming(DP) respectively. However, there is an intrinsic difference between their analysis and the one we performed here. The solution space or the domain of the network studied by Aiyer *et al.* and C.Chuan *et al.* is the unit hypercube, H_c^n where $H_c^n \equiv \{ x = (x_1, \ldots x_n)^T \in \mathbf{R}^n : 0 \leq x_i \leq 1, i = 1, \ldots n \}$ while the space we are investigating is $H^n \equiv \{ x = (x_1, \ldots x_n)^T \in \mathbf{R}^n : x_i = \{0,1\}, i = 1, \ldots n \}$.

4.3.3.1 Energy Profile analysis:

For typical Hopfield energy function, $E(x)$, according to Section 3.1, it can be divided into two terms, one is the cost for optimization and the other is the cost for the constraints. This is just the penalty method well-adopted for transforming a constrained optimization problem to an unconstrained one.

Let $E1(x)$ denote the cost for the constraints,

$E2(x)$ denote the cost for optimization &

w_{hard} is the trade-off weighting for the two costs.

$$E(x) = E1(x) + w_{hard}\, E2(x)$$

Although, we can have unlimited forms for the constraints, the most common one used so far are:

T1) $\sum_i \sum_j x_i x_j$ - preventing more than one neuron being active for a group of neurons.

T2) $\left(\sum_i x_i - n \right)^2$ - ensuring n neurons being on for a group of neurons.

Constraints with the form *T1* must go with those of the form *T2* as the former ones just standing alone will very easily yield a solution with all zero for x. Furthermore, in fact, the constraint with the form *T1* is somehow reductant and can be replaced by *T2* form.

Consider the example by C.Chuan *et al.* [CMS91]. The cost for the constraints is

$$E1(x) = \sum_s \sum_i \sum_j x_{si} x_{sj} + \left(\sum_s \sum_i x_{si} - n \right)^2$$

where s is the index for stage and i,j are the index for nodes per stage.

The first term is to prevent more than one neuron being active for each stage of the DP network while the second one is to ensure only n neurons are active for the whole network. In fact, the whole statement can be rephrased to be simply only one neuron is active

for each stage of the DP network and the corresponding cost term can be,

$$E1(x) = \sum_s \left(\sum_i x_{si} - 1 \right)^2$$

which is summation of the costs with form *T1*. Thus, by the above argument, analysis of the constraints with only the form *T2* is sufficient and generic enough for most of the problems and thus will be the only form investigated throughout the analysis.

Definition 4.4:

V is the set of **valid states** and defined as $V \equiv \{ x=\{x_i, \ldots x_n\} \in H^n : \sum_i x_i = n \}$.

Thus, V is a subset of H^n and $H^n \backslash V$ will be the set of invalid states.

Definition 4.5:

L is the set of local minimum state and defined as $L \equiv \{ x=\{x_1, \ldots x_n\} \varepsilon H^n : x$ is a local minimum state[4] $\}$.

Proposition 4.1:

For sufficiently large w_{hard}, only the states, $x \in V$ are the local minimum states of $E(x)$, or $L \equiv V$.

Proof:

For sufficiently large w_{hard}, the energy profile of $E(x)$ is dominated by that of $E1(x)$. The major characteristics, eg. minimum states will be dominated by $E1(x)$.

Lemma 4.4: v is a valid state if and only if v is a global minimum state of $E1(x)$ in H^n.

Proof:

For every state vector $x \in H^n$,

[4] Refer to Definition 3.

$$E1(x) = \left(\sum_i x_i - n \right)^2 \geq 0$$

If $x \in V$, by the definition of valid state, it follows that $\sum_i x_i = n$. Thus, $E1(x) = 0$ and x is the global minimum state of $E1(x)$.

On the other hand, if x is a global minimum state of $E1(x)$, then $E1(x) = 0$. This implies $(\sum_i x_i - n)^2 = 0$ and $\sum_i x_i = n$. Therefore, v is a valid state. This completes the proof for the lemma. ■

Lemma 4.5: There are no local minimum states among the invalid states of $E1(x)$ in H^n.

For every invalid state vector $x \in H^n \backslash V$,

$$\left(\sum_i x_i - n \right)^2 > 0$$

it's obvious that there must exist i such that changing x_i to $1\text{-}x_i$ will result in decrease in $E1(x)$, ie. closer to one of the neighboring valid states. Thus, there are no local minimum states among the invalid states of $E1(x)$ in H^n. ■

By Lemmas 4.4 and 4.5, we can conclude that only the states $x \in V$ are the minimum states for $E1(x)$ and thus $E(x)$, ie. $V \equiv L$. This proves Proposition 4.1. ■

Before ending up this subsection, it is necessary to point out that the argument of E1(x) being dominating part is a very important fact for further analysis. Although a lot of Hopfield network applications are basically quadratic programming, there still exists a lot of problem which are of higher order. This means a higher order Hopfield energy function will result. Nevertheless, very likely cost for the constraints will remain generally the same, ie. $(\sum_i x_i - n)^2$. By the Proposition 4.1, the energy profile is dominated by the constraint term. This implies that the above analysis is still valid for general order Hopfield network and

furthermore it makes the existing analysis of standard Hopfield network still valid for higher-order one.

4.3.3.2 Size of attractive basin and pole strength required:

Again, by Proposition 4.1, the analysis is performed only for $E1(x)$ and expected that it applies to $E(x)$ at the same time. It's obvious that size of attraction basin for the local minimum state of $E1(x)$ is problem dependent or in fact constraint dependent.

Definition 4.6:

b is the corresponding size of **attraction basin** of a local minimum state, x^* of $E1(x)$ if $\exists\, x : \| x\text{-}x^* \|_1 \leq b$ and x can converge to x^* by the network dynamics.

Proposition 4.2:

The size of attraction basin of $E1(x)$ for every local minimum state is the same.

Proof:

As $E1(x)$ puts equal penalty according to distance measure from local minimum state and it is independent of the explicit local minimum location and the distance between adjacent local minimum states in L is constant, it's obvious that the attraction basin of $E1(x)$ should all be the same. ■

By this proposition, the size of attraction basin in fact equals to the half of the L_1 norm between two adjacent local minimum states. Let's consider two common examples.

Example 4.1: Constraint: For a *nim* matrix of neurons, there can only be one neuron active per row.

$$E1(x) = \sum_i \left(\sum_j x_{ij} - 1 \right)^2$$

Obviously, the L_1 norm between two local minimum states (or two adjacent valid state) is 2 and thus the size of the attraction basin is 1.

Example 4.2: Constraint: For a *nxm* matrix of neurons, there can only be one neuron active per row and column.

$$E1(x) = \sum_i \left(\sum_j x_{ij} - 1 \right)^2 + \sum_j \left(\sum_i x_{ij} - 1 \right)^2$$

An adjacent valid state can be obtained by swapping any two rows or columns, the L_1 norm difference is 4 and thus the size of the attraction basin is 2.

Definition 4.7:

Sequence $\{x(t)\}$ is an **uphill path** for E1(x) iff $\frac{\Delta E1(x)}{\Delta t} > 0$.

Definition 4.8:

Sequence $\{x(t)\}$ is a **downhill path** for E1(x) iff $\frac{\Delta E1(x)}{\Delta t} < 0$.

Proposition 4.3:

Along an uphill path within the local minimum basin of x^*, $\frac{\Delta \|x(t)-x^*\|_1}{\Delta t} > 0$.

Proof:

By Definition 4.7, along an uphill path,

$$\frac{\Delta E1(x)}{\Delta t} > 0$$

As $E1(x) = (\sum_i x_i - n)^2$, increasing t to t+1 yields

$$\begin{aligned} \Delta E1(x) &= \left(\sum_i x_i(t+1) - n \right)^2 - \left(\sum_i x_i(t) - n \right)^2 \\ &= \left(\sum_i x_i(t+1) - \sum_i x_i(t) \right) \left(\sum_i x_i(t+1) + \sum_i x_i(t) - 2n \right) > 0 \end{aligned}$$

$\Rightarrow$

Case I: $\sum_i x_i(t+1) > \sum_i x_i(t)$ & $\sum_i x_i(t+1) + \sum_i x_i(t) > 2n$

$$\therefore \ \Delta \ \|x(t+1)-x(t)\|_1 > 0 \qquad \forall\, t$$

When t=0, $x(t) = x^*$.

$$\Delta\|x(t)-x^*\|_1 = \Delta\|x(t)-x(t-1)\|_1 + \Delta\|x(t)-x(t-1)\|_1 + \ldots + \Delta\|x(1)-x^*\|_1 > 0$$

Case II: $\sum_i x_i(t+1) < \sum_i x_i(t)$ & $\sum_i x_i(t+1) + \sum_i x_i(t) < 2n$

Again $\therefore \ \Delta \ \|x(t+1)-x(t)\|_1 > 0 \qquad \forall\, t$

$$\Rightarrow \ \Delta \ \|x(t)-x^*\|_1 > 0$$

Thus, $\dfrac{\Delta\|x(t)-x^*\|_1}{\Delta t} > 0$ is proved. ■

Proposition 4.4:

Along a downhill path within the local minimum basin of x^*, $\dfrac{\Delta\|x(t)-x^*\|_1}{\Delta t} < 0$.

Proof:

The proof is very similar to that of Proposition 4.3 and is not repeated here.

According to Proposition 4.3 & 4.4, Fig. 4.7 shows a typical Hopfield energy profile from one local minimum state to an adjacent local minimum state with size of attraction basin = 2.

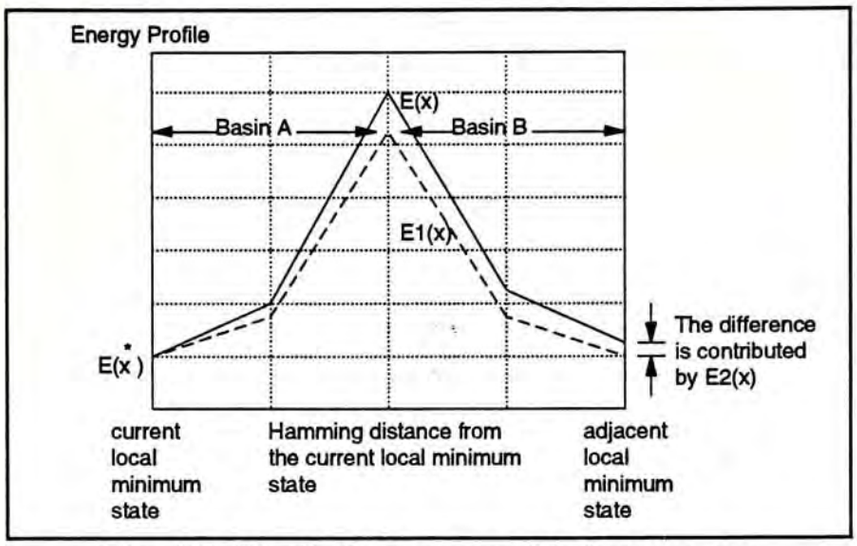

Fig. 4.7: Hopfield energy profile between two adjacent local minimum states for problem with attraction basin size = 2

Definition 4.9:

λ is said to be a currently **sufficient pole strength** of a tunneling function if the associated tunneling network always converge to a state beyond the current local minimum attraction basin.

For a sufficient pole strength, Basin A of Fig. 4.7 should be destroyed. As E1(x) is quadratic, the just sufficient pole strength should be greater than two[5]. For a more detailed illustration, assuming the constraint of example 4.2,

$$E1(x) = \sum_i \left(\sum_j x_{ij} - 1 \right)^2 + \sum_j \left(\sum_i x_{ij} - 1 \right)^2$$

which is an energy profile with size of attraction basin = 2.

[5] If it's intended to stick to power = 2, we can replace the pole by $\left(\sum_k (x_k - x_k^*)^2 + 1 \right)^\lambda$. Then, λ=2 will then become a sufficient pole strength.

Assume a trajectory from current local minimum state x^* to an adjacent local minimum state be the sequence $\{ x_1, x_2, x_3, x_4, x_5 \}$, where x_1 equals x^* and x_5 equals an adjacent local minimum state.

By the Proposition 4.2, 4.3, 4.4,

$$\|x_i - x^*\|_1 = i-1 \qquad \forall\, i \in \{ 1,2,3,4,5 \} \;\; \& \;\; E1(x_1) < E1(x_2) < E1(x_3) > E1(x_4) > E1(x_5).$$

Assume the corresponding tunneling function, $T(x)$ is:

$$\frac{E1(x) - E1(x^*)}{\left(\sum_k (x_k - x_k^*)^2\right)^\lambda} = \frac{E1(x) - E1(x^*)}{\left(\|x_i - x^*\|_1\right)^\lambda} = \frac{E1(x) - E1(x^*)}{\left(i-1 \right)^\lambda}$$

For $\lambda = 2$,

$T(x_1) = \infty$

$T(x_2) = 2$

$T(x_3) = 2$

$T(x_4) = 0.22$

$T(x_5) = 0$

For $\lambda = 2.1$,

$T(x_1) = \infty$

$T(x_2) = 2$

$T(x_3) = 1.867$

$T(x_4) = 0.199$

$T(x_5) = 0$

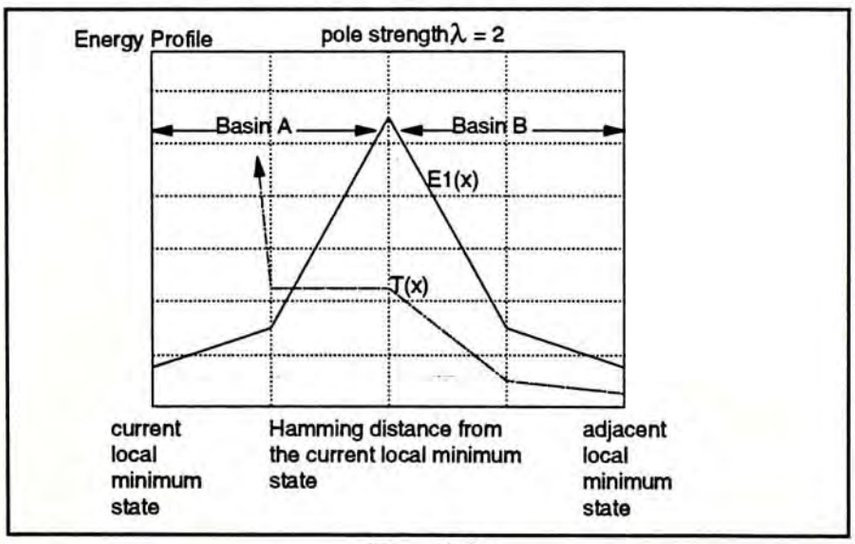

Fig. 4.8a

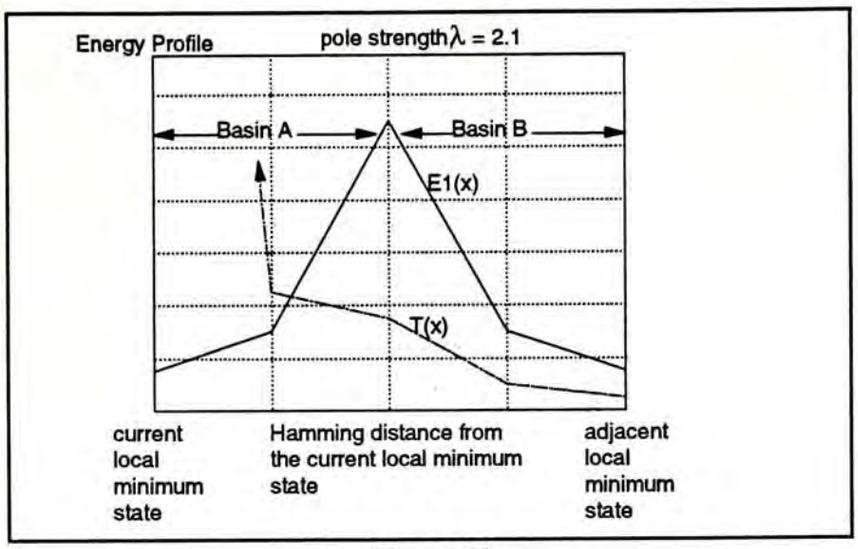

Fig. 4.8b

Fig. 4.8: Showing Hopfield energy profile and the corresponding Tunneling energy profile with pole strength equals a) $\lambda = 2$ and b) $\lambda = 2.1$

It's noted that for λ just being larger than 2, $T(x(t))$ becomes montonic decreasing as time proceeds. If λ is fixed to 1, the profile then becomes as Fig. 4.9. So, $\lambda = 1$ is not a sufficient pole strength for the problem with size of attraction basin = 2. But it doesn't mean that tunneling network doesn't work. The local minimum created within the current local minimum basin can be escaped by shifting the pole there. However, the efficiency of the network may be reduced.

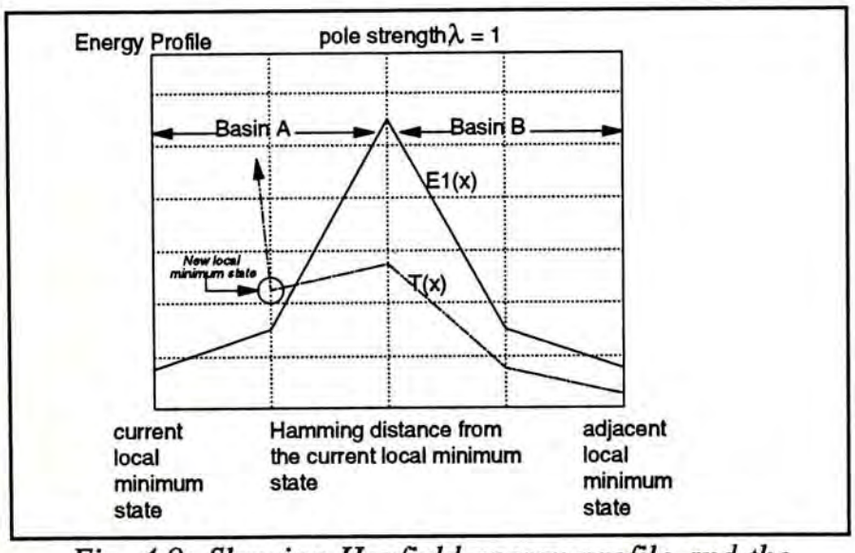

Fig. 4.9: Showing Hopfield energy profile and the corresponding tunneling energy profile with pole strength, $\lambda = 1$

Recall the examples of the constraints. Example 4.1 required $\lambda = 1$ as sufficient pole strength while example 2 required $\lambda = 2$ as sufficient pole strength.

Here, come up with an interesting argument. If we can increase the pole strength continuously, not only the current local minimum basin is destroyed, the other local minimum basins can also be destroyed, provided the energy value of the other local minimum states is not the same as the current one.[6] However, new set of local minimum states, B^n are created and defined as $B^n \equiv \{ x = (x_1, \ldots x_n)^T \in H^n : (x-x^*)^T (x-x^*) = n \}$. So, we are not interested in this extreme case. However, we can increase the pole adaptively during the course of tunneling and can escape from the local minimum encountered without the necessity of shifting pole. After playing with the fantasy in theoretical aspect, practical perspective have to be considered. An adaptive pole is not so easily implemented as the updating rule involves difference of two terms with the pole strength as the power.

4.3.3.3 A new type of pole eases the implementation problem :

Let the pole be of the form $K^{\sum_i (x_i - x_i^*)^2}$ where K is a prescribed constant. The updating rule based on that form of pole will be:

$$\Delta T(x) = \frac{1}{D'}\left[K^{\sum_i (x_i-x_i^*)^2}\left(E(x)\big|_{x_i} - E(x^*) + \frac{\partial E(x)}{\partial x_i}\Big|_{x_i}\Delta x \right) - K^{\sum_i (x_i-x_i^*)^2+(1-2x_i^*)(1-2x_i)}\left(E(x)\big|_{x_i} - E(x^*)\right)\right]$$

$$\text{where } D' = K^{\sum_i (x_i-x_i^*)^2}\, K^{\sum_i (x_i-x_i^*)^2+(1-2x_i^*)(1-2x_i)} > 0$$

$$= \frac{1}{D'}\cdot K^{\sum_i (x_i-x_i^*)^2}\left[(1-2x_i)(\,1-K^{(1-2x_i^*)(1-2x_i)})(E(x)\big|_{x_i} - E(x^*)) + \frac{\partial E(x)}{\partial x_i}\Big|_{x_i}\right]\Delta x$$

[6] If some local minimum states have energy value equal to that of the current local minimum states, the tunneling function there will remain zero no matter how large the pole strength is.

as $\Delta x = 1-2x_i$ & $(1-2x_i)^2 = 1$

Then,

$$x_i = f_h\left[\frac{\Delta T(x)}{\Delta x}\right]$$

$$= f_h\left[(1-2x_i)(1-K^{(1-2x_i^*)(1-2x_i)})(E(x)-E(x^*)) + \frac{\partial E(x)}{\partial x_i}\right]$$

To vary the pole strength, simple vary the value of K. And also, as the power of K is either 1 or -1, we don't really need power function at all. For a particular K, only two memory are needed, one for K and one for $1/K$.

The "adaptive pole tunneling network" is applied to TSP and it's shown that it can escape from local minimum and approach to the global minimum. Its effectiveness compared with the pole shifting version is manifested in the following section.

4.4 Simulation result and performance comparison:

Tunneling network is being simulated on DEC machine to verify its capability of performing global optimization. To compare its performance with some other existing optimization algorithms, including **Simulated Annealing, Discrete & Continous Hopfield network**, Travelling Salesman Problem is chosen to be the benchmark test and different algorithms are applied to find the optimal path. The testing set includes randomly generated ten 10-city TSP problems and ten 20-city TSP problems. Two converging properties are measured:

i) Optimal solution obtained after the optimization process stopped,

ii) Average convergence rate, r during the first half life, which is defined as:

$$r = \frac{Energy(t=0) - Energy(t=stopped)}{0.5*(half-life) + 0.25*(quarter-life)}$$

where half-life and quarter-life are respectively the time when half and quarter of the numerator is reached.

4.4.1 Simulation Experiment:

The problem we choose for the benchmark test is 10-city and 20-city Travelling Salesman Problem (TSP). So we first formulate TSP as a neural network formulation. As the most original TSP neural network formulation by J.J.Hopfield *et al.* [HT85] will converge to invalid solution very often, we choose the modified formulation by Aiyer *et al.* [ANF90]. The proposed neural network formulation is stated below;

$$\begin{aligned} T_{xi,yj} = & - A\,\delta_{xy}(1-\delta_{ij}) - A\,\delta_{ij}(1-\delta_{xy}) \\ & - 2\,A_1\delta_{xy}\delta_{ij} \\ & - C + \frac{2\,(An - A + A_1)}{n^2} \\ & - D\,d_{xy}\,(\delta_{j,i+1} + \delta_{j,i-1}) \\ I_{xi} = & \; Cn \end{aligned}$$

where n is the number of cities,

x,y and i,j are the city and time index respectively,

δ_{ab} are the delta function, which equals 1 only when a = b,

d_{xy} is the distance between city x and city y.

Through an eigenvalue analysis on the connection matrix, the weighting parameters A, A_1, C and D can set according to following deterministic rules.

$$\frac{A_1}{A} = 1 - \frac{n}{322} \qquad C = A/n \qquad D = An/80$$

Following is a table of the parameters calculated from the above rules used in our simulation.

No. of cities	A	A_1	C	D
10	8	7.75	0.8	1
20	8	7.75	0.4	2

Table 4.1: Parameters used for the TSP problem formulation

After the neural network formulation step, followings are the detailed descriptions of the implementation of the different optimization schemes that we adopted for the experiment.

a) Hopfield network:

Neuron model:	McCulloch & Pitt model
Updating sequence:	Asynchronous
Stopping rule:	All neurons remain unchanged after updating all of them
Initial state:	All neurons set to 1/no. of cities

---

do until stopping rule is satisfied

select a node x_i at random

$$x_i \Leftarrow \text{hardlimit}\left(-\frac{\partial E}{\partial x_i}\right)$$

---

b) Continuous Hopfield network:

Neuron model:	Sigmoid nonlinearity, $g(x) = [1+\exp(-x/u_o)]^{-1}$ where u_0 = 0.0001
Updating sequence:	Asynchronous
Network dynamic:	simulated by fourth order Runge-Kutta method
Stopping rule:	Changes of the output of the neurons are all less then 0.0001
Initial state:	All the neurons output, v_i set to 1/no. of cities while all the neurons input, u_i set to $-u_o ln$(no. of cities - 1) + small random noise. h = 0.00001 and decay constant for each neuron, $\tau = 1$

do until stopping rule is satisfied

select i^{th} node at random

$k_1 \Leftarrow h*f(u_i)$

$k_2 \Leftarrow h*f(u_i+0.5*k_1)$

$k_3 \Leftarrow h*f(u_i+0.5*k_2)$

$k_4 \Leftarrow h*f(u_i+k_3)$

$u_i \Leftarrow u_i + 0.167*(k_1+2k_2+2k_3+k_4)$

$v_i \Leftarrow g(u_i)$

* where $f(u_i) = -\dfrac{\partial E}{\partial x_i} - u_i/\tau$

c) Simulated Annealing:

Cooling schedule:

Fast schedule: Initial temperature, $T_o = 3$

Decreasing rate, $r = 0.95$

Slow schedule: Initial temperature, $T_o = 5$

Decreasing rate, $r = 0.99$

Stopping rule: - stopping rule for equilibrium being established at a particular temperature: all neurons updated ten times

- stopping rule for the cooling scheme: temperature is decreased to 0.01

$T \Leftarrow T_o$

while $(T>T_{min})$

do until equilibrium is established

select a node x_i at random

R = random(1.0)

if $(\exp(- \frac{\partial E}{\partial x_i}\frac{(1-2x_i)}{T}) > R)$

$x_i = 1-x_i$

$T \Leftarrow r*T$

---

d) Tunneling network:

Stopping rule: Best solution obtained remains unchanged for 200 iterations

Local minimum detection: All nodes remains unchanged after updating

---

$x^* \Leftarrow$ latest found local minimum state

select a node x_i at random; change it to $1-x_i$

do until stopping rule is satisfied

select a node x_i at random

$Td \Leftarrow L^1$ norm between x and x^*

$$x_i \Leftarrow \text{hardlimit}\left(T_d * \frac{\partial E(x)}{\partial x_i} - (1-2x_i^*)(E(x)-E(x^*))\right)$$

$$\text{if } x_i \text{ changed, } E(x) \Leftarrow E(x) + \frac{\partial E(x)}{\partial x_i}(2x_i - 1)$$

if local minimum encountered, $x^* \Leftarrow x$

---

e) Adaptive Tunneling network:

Stopping rule: Best solution obtained remains unchanged for 200 iterations

Local minimum detection: All nodes remains unchanged after updating

Initial pole strength, $K_o = 1$

Pole strength increment, $\Delta K = 0.05$;

$K \Leftarrow K_o$

$x^* \Leftarrow$ latest found local minimum state

select a node x_i at random; change it to $1-x_i$

do until stopping rule is satisfied

select a node x_i at random

$x_i \Leftarrow$ hardlimit(

$$(1-2x_i)(1-pow(K,(1-2x_i^*)(1-2x_i))(E(x)-E(x^*))+\frac{\partial E(x)}{\partial x_i}\ )$$

if x_i changed, $E(x) \Leftarrow E(x)+\frac{\partial E(x)}{\partial x_i}(2x_i-1)$

if local minimum encountered, $K \Leftarrow K+\Delta K$

4.4.2 Simulation result and discussion:

4.4.2.1 Comparisons on optimal path obtained and the converging rate

From Table 4.2 and 4.3, it's obvious that Tunneling network can obtain solution much better than Simulated Annealing with an even faster converging rate. And the algorithm still performs very well when problem size increased to 20 city, at which Simulated Annealing(SA)'s solution quality is far from satisfaction. Of course, if we further slow down

the cooling schedule, a better solution can be found by SA. However, the converging rate will drop very serious and will be simply too long for acceptance. Fig. 4.10 shows typical converging curves for the different algorithms and Fig. 4.11 show the best city maps that they got. It's observed that Adaptive Tunneling network(ATN) almost can get the optimal solution and Tunneling network(TN) is the second best. For ATN, the converging rate is comparable to the slow SA algorithm. However, a lot of iterations are in fact spent on the pole strength increment. If this limitation can be solved in the hardware implementation step, the converging rate can be greatly improved.

Besides the above advantages, Tunneling network is a generic solution for global optimization and there is no advanced knowledge needed for usage. Comparing with SA and some other existing algorithms, like mean field annealing, parameters, like the cooling schedule, have to be estimated before the algorithm can be applied effectively.

4.4.2.2 On Decomposition of Tunneling network

For any optimum searching algorithms including tunneling, the time required to get an even better solution is increasing exponentially as better and better solution is found. To improve the situation, we can decompose the network to different partitions and restrict the updating to particular partition at a time. By this method, the exploring power of tunneling network can be enhanced. We have tried two decomposition schemes; i) fixed (partition set of the network is fixed), ii) random (partition set of the network is time-varying in a random manner).

The simulation result is illustrated in Fig. 4.12 and Fig. 4.13. It is observed that small size partitioning scheme can result in converging to a better solution in general. Between fixed and random partitioning, the latter one appears to be a better choice as fixed partitioning unreasonably confines the searching space such that the whole solution space can't be covered.

Problem	SA(Fast)		SA(Slow)		DHop		CHop		TN		ATN	
	d^*	r	d^*	r	d^*	r	d^*	r	d^*	r	d^*	r
1	9.24	0.05	8.10	0.015	11.90	-	12.3	-	6.64	0.130	6.64	0.005
2	9.06	0.007	7.34	0.006	9.70	-	12.64	-	7.22	0.019	6.42	0.002
3	7.78	∞	7.28	0.002	7.78	-	8.94	-	6.56	0.003	6.50	0.002
4	7.38	0.216	7.24	0.028	12.14	-	11.56	-	6.04	1.53	5.10	0.020
5	6.74	0.184	6.12	0.118	8.96	-	8.22	-	5.94	0.025	5.68	0.109
6	6.64	0.007	5.94	0.005	7.12	-	5.96	-	5.10	0.006	3.70	0.010
7	6.18	0.024	5.98	0.010	7.52	-	10.04	-	4.94	0.004	4.88	0.008
8	6.52	0.070	6.36	0.015	9.88	-	7.04	-	5.18	0.390	5.02	0.054
9	9.26	0.023	9.24	0.006	11.14	-	15.0	-	7.18	0.180	6.66	0.012
10	10.8	0.077	8.86	0.039	14.5	-	13.7	-	7.86	0.368	6.62	0.018
mean of r	0.067		0.0243		-		-		0.266		0.024	

Table 4.2: Comparison on best path length, d^* obtained and converging rate, r between different algorithms on 10-city TSP
SA: Simulated Annealing, DHop & CHop: Discrete and Continuous Hopfield network,
TN: Tunneling Network, ATN: Adaptive Tunneling Network

Problem	SA(Fast)		SA(Slow)		DHop		CHop		TN		ATN	
	d^*	r	d^*	r	d^*	r	d^*	r	d^*	r	d^*	r
1	18.39	0.048	15.61	0.020	22.72	-	23.99	-	12.94	0.540	10.67	0.078
2	16.24	0.042	13.79	0.016	19.54	-	17.64	-	11.53	0.220	8.06	0.040
3	11.93	0.120	13.91	0.020	20.96	-	18.55	-	11.93	0.260	8.30	0.056
4	15.36	0.010	13.04	0.008	16.30	-	18.21	-	9.74	0.050	8.95	0.027
5	17.39	0.030	13.84	0.015	19.69	-	19.60	-	9.93	0.080	8.36	0.030
6	15.05	0.053	13.09	0.014	18.87	-	14.65	-	10.70	2.720	8.10	0.024
7	10.26	0.074	13.20	0.010	17.15	-	18.12	-	11.05	0.150	9.18	0.015
8	14.90	0.094	14.75	0.026	22.08	-	19.98	-	10.87	0.630	9.02	0.044
9	17.15	0.060	16.00	0.019	21.78	-	22.44	-	13.32	1.060	8.56	0.024
10	18.20	0.051	17.45	0.014	22.19	-	19.85	-	13.30	0.250	10.44	0.022
mean of r	0.0584		0.0162		-		-		0.596		0.036	

Table 4.3: Comparison on best path length, d^* obtained and converging rate, r between different algorithms on 20-city TSP
SA: Simulated Annealing, DHop & CHop: Discrete and Continuous Hopfield network,
TN: Tunneling Network, ATN: Adaptive Tunneling Network

Problem	pt = 4		pt = 6		pt = 8		pt = 10	
	d^*	r	d^*	r	d^*	r	d^*	r
1	6.30	0.039	5.98	0.036	6.64	0.125	6.64	0.130
2	6.64	0.019	6.72	0.009	6.74	0.025	7.22	0.019
3	6.46	0.011	6.44	0.003	6.30	0.004	6.56	0.003
4	5.94	0.240	5.62	0.090	5.88	1.040	6.04	1.53
5	5.78	0.047	5.58	0.015	5.64	0.018	5.94	0.026
6	5.36	0.029	4.84	0.002	5.18	0.013	5.10	0.006
7	4.88	0.029	5.62	0.023	5.38	0.019	4.94	0.004
8	5.18	0.062	5.70	0.130	5.16	0.067	5.18	0.391
9	7.52	0.009	7.50	0.007	8.32	0.015	7.18	0.180
10	8.08	0.190	7.94	0.120	7.70	0.150	7.86	0.370
mean d^*, r	6.21	0.068	6.20	0.032	6.29	0.148	6.27	0.266

Table 4.4 : Comparison on best path length, d^* obtained and converging rate between different size fixed partition decomposition of tunneling network on 10-city TSP. The fixed partition scheme is grouping consecutive rows of neurons together and the partitions are overlapping. "pt" stands for number of rows of neurons for each partition. pt = 10 means whole network forming one and only one partition.

Problem	pt = 4		pt = 8		pt = 12		pt = 16		pt = 20	
	d^*	r	d^*	r	d^*	r	d^*	r	d^*	r
1	14.26	0.090	13.68	0.150	12.12	0.060	11.61	0.550	12.94	0.540
2	11.59	0.210	11.30	0.060	10.35	0.150	9.82	0.060	9.80	0.070
3	11.68	0.100	11.25	0.092	11.45	0.210	10.26	0.046	11.92	0.260
4	11.53	0.088	9.34	0.036	10.09	0.480	9.87	0.230	9.74	0.051
5	13.84	0.202	10.96	0.110	10.76	0.297	12.38	0.280	9.93	0.079
6	9.65	0.071	8.61	0.076	10.12	0.194	10.10	0.460	10.70	2.720
7	10.17	0.104	10.51	0.041	9.88	0.025	9.47	0.044	11.08	0.152
8	9.86	0.256	10.50	0.212	9.70	0.303	11.40	1.340	10.86	0.63
9	12.54	0.048	10.79	0.145	11.87	0.117	11.66	0.147	13.32	1.058
10	13.10	0.103	14.13	0.448	12.58	0.081	12.21	0.071	13.31	0.254
mean d^*, r	11.82	0.128	11.11	0.137	10.89	0.197	10.88	0.323	11.36	0.627

Table 4.5 : Comparison on best path length, d^* obtained and converging rate between different size fixed partition decomposition of tunneling network on 20-city TSP. The fixed partition scheme is grouping consecutive rows of neurons together and the partitions are overlapping. "pt" stands for number of rows of neurons for each partition. pt = 20 means whole network forming one and only one partition.

Problem	pt = 4		pt = 6		pt = 8		pt = 10	
	d^*	r	d^*	r	d^*	r	d^*	r
1	5.98	0.030	6.22	0.045	7.06	0.403	6.70	0.065
2	6.94	0.024	6.98	0.075	7.42	0.230	7.16	0.022
3	6.22	0.008	6.62	0.007	6.56	0.006	6.98	0.004
4	5.10	0.019	5.10	0.503	6.32	0.243	6.08	1.510
5	5.66	0.016	5.76	0.012	5.76	0.011	6.30	0.058
6	4.42	0.013	5.06	0.004	5.10	0.063	5.36	0.019
7	4.88	0.002	5.92	0.009	5.48	0.113	5.64	0.009
8	5.02	0.044	5.04	0.052	5.26	0.085	5.18	1.173
9	6.66	0.042	7.66	0.053	6.92	0.023	7.40	0.156
10	6.62	0.080	7.90	0.366	8.14	0.117	7.52	0.580
mean d^*, r	5.75	0.028	6.23	0.113	6.40	0.128	6.43	0.359

Table 4.6 : Comparison on best path length, d^* obtained and converging rate between different size random partition decomposition of tunneling network on 10-city TSP. The random partition scheme is picking rows of neurons randomly and grouping them together at a time and thus the partition is time-varying in random manner. "pt" stands for number of rows of neurons for each partition. pt = 10 means whole network forming one and only one partition.

Problem	pt = 4		pt = 8		pt = 12		pt = 16		pt = 20	
	d^*	r	d^*	r	d^*	r	d^*	r	d^*	r
1	8.76	0.082	11.56	0.248	12.47	0.168	12.0	0.137	12.79	0.093
2	8.61	0.053	10.14	0.142	11.36	0.182	8.82	0.037	11.53	0.222
3	9.40	0.250	10.47	0.181	11.68	0.155	10.35	1.061	10.86	0.069
4	8.06	0.069	8.21	0.021	8.72	0.075	9.87	0.083	10.03	0.087
5	10.91	0.071	10.28	0.048	10.73	0.172	12.59	0.104	11.14	0.120
6	8.50	0.130	9.72	0.223	11.31	1.510	9.54	0.22	9.817	0.220
7	8.68	0.072	9.82	0.198	11.24	0.128	10.54	0.066	10.88	0.065
8	8.84	0.170	9.74	1.377	9.63	0.266	10.55	1.05	9.81	0.536
9	10.38	0.069	12.42	0.164	13.37	0.172	11.57	0.34	10.83	0.215
10	10.29	0.140	12.08	0.187	13.31	2.960	12.53	0.075	13.73	0.197
mean d^*, r	9.25	0.111	10.44	0.279	11.38	0.579	10.84	0.317	11.14	0.182

Table 4.7 : Comparison on best path length, d^* obtained and converging rate between different size random partition decomposition of tunneling network on 20-city TSP. The random partition scheme is picking rows of neurons randomly and grouping them together at a time and thus the partition is time-varying in random manner. "pt" stands for number of rows of neurons for each partition. pt = 20 means whole network forming one and only one partition.

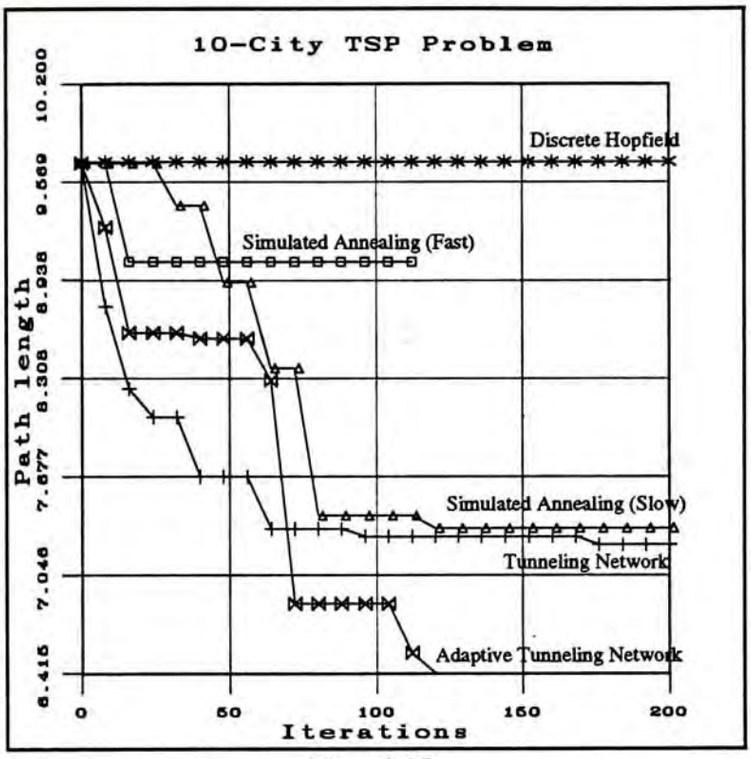

Fig. 4.10a

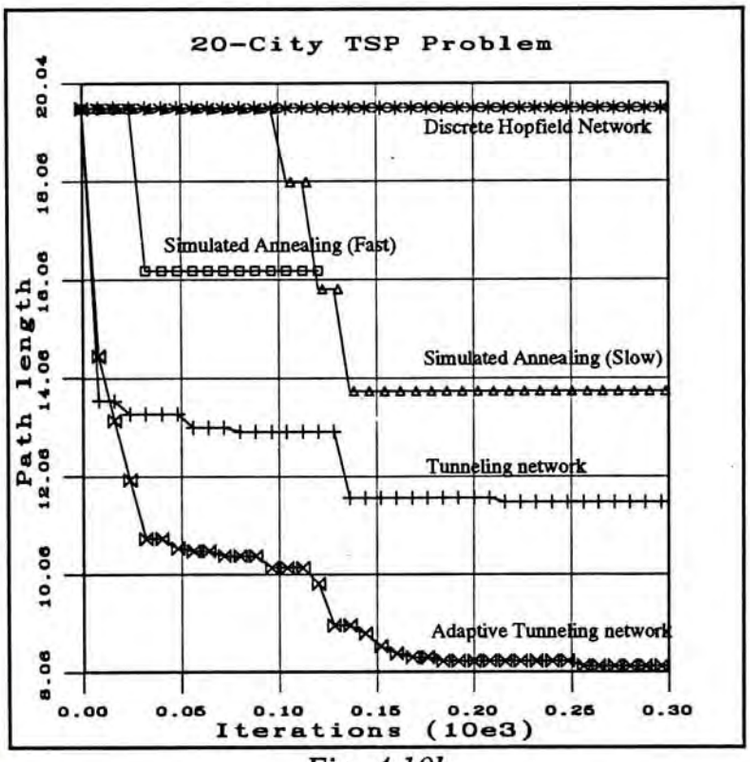

Fig. 4.10b

Fig. 4.10: Illustration of typical converging curve for different optimization schemes on a) 10-city and b) 20-city TSP

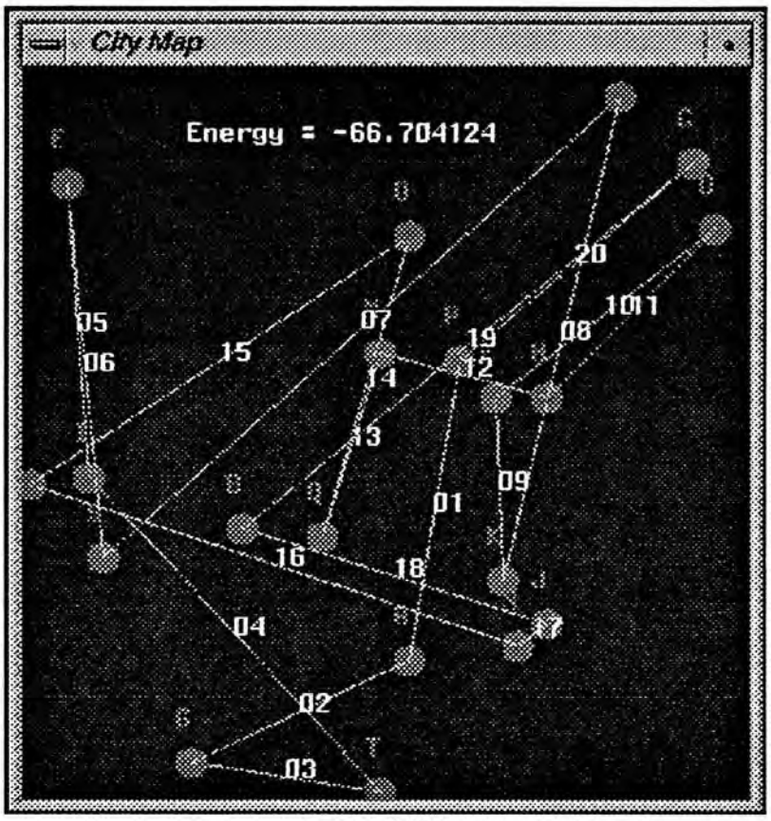

Fig. 4.11a

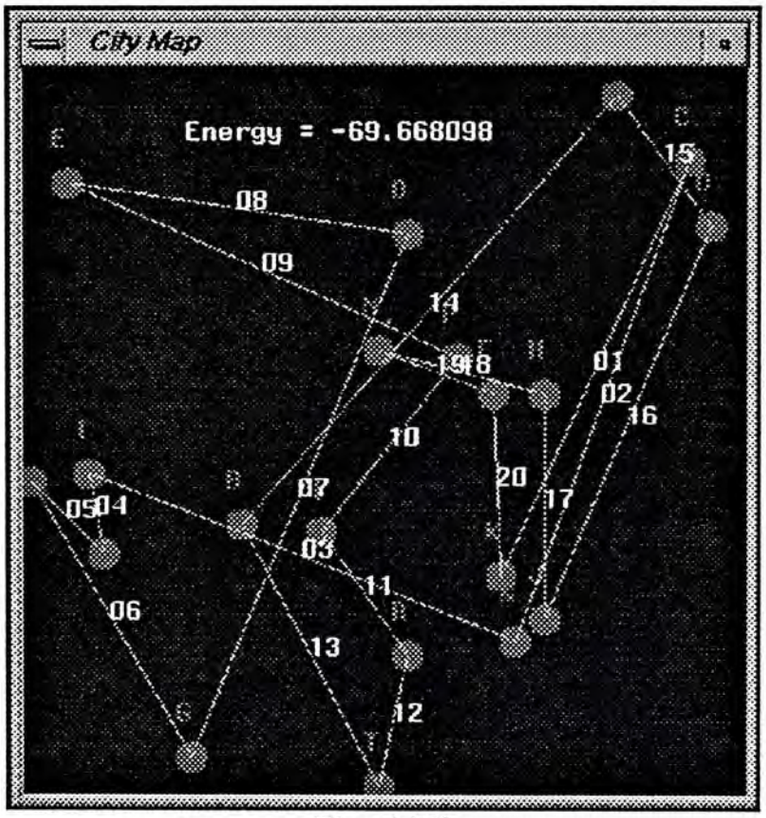

Fig. 4.11b

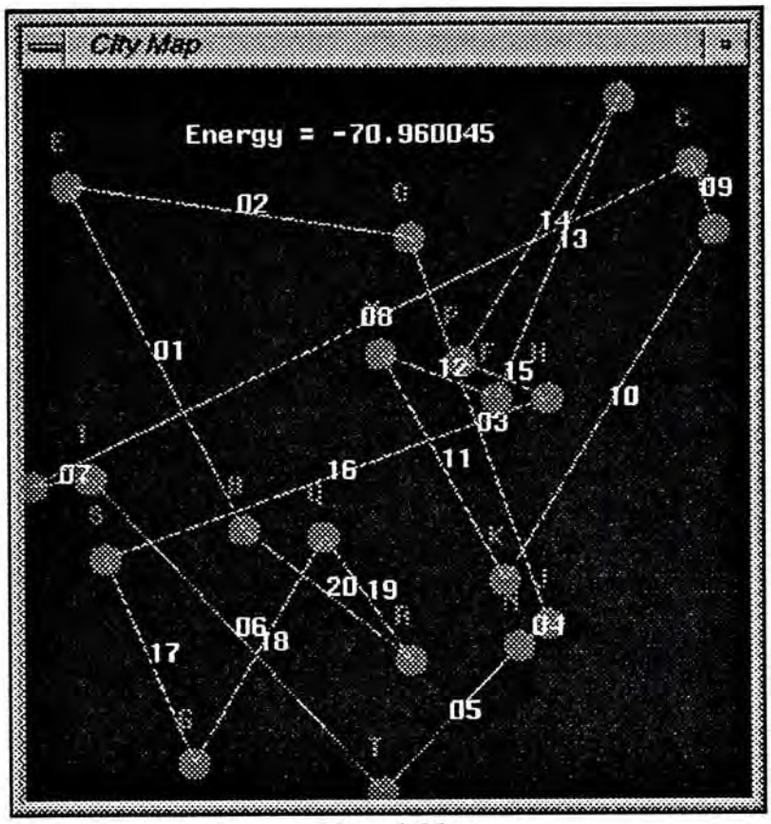

Fig. 4.11c

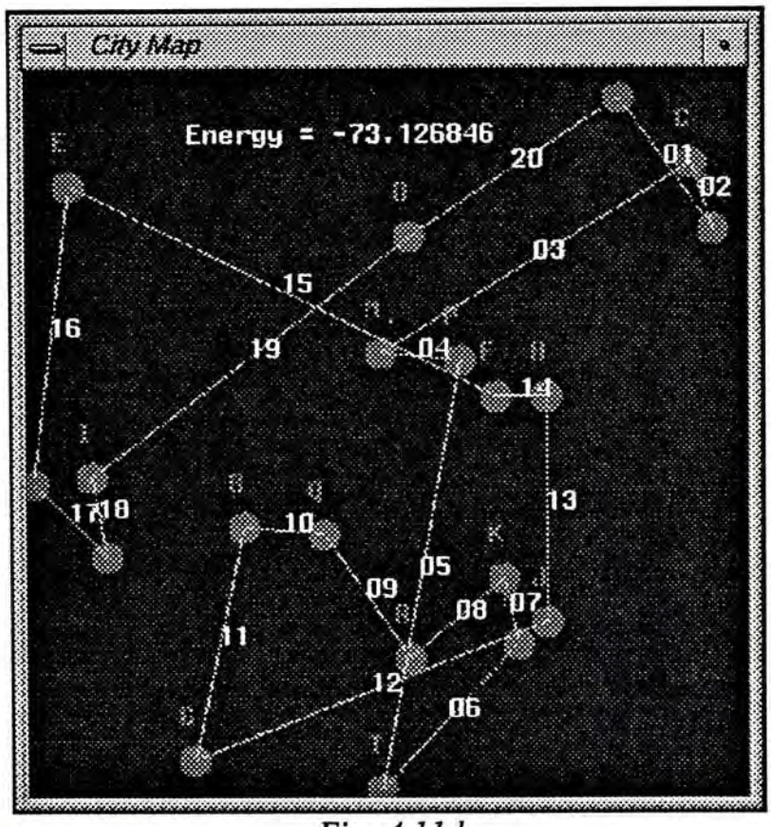

Fig. 4.11d

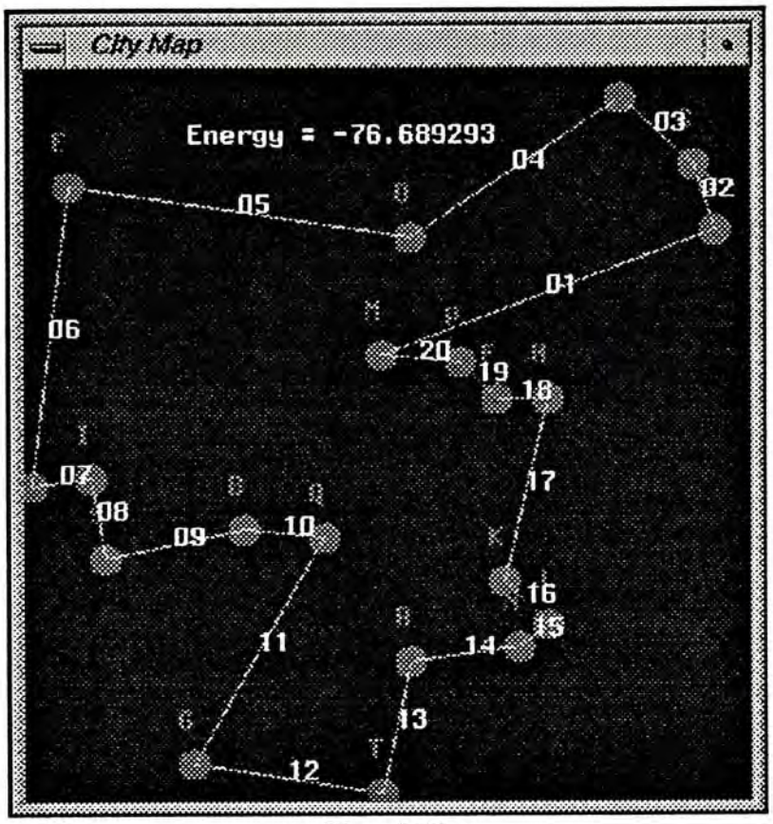

Fig. 4.11e

Fig. 4.11: Illustration of best result obtained by different optimization schemes on 20-city TSP

a) Discrete Hopfield network;
b) Simulated Annealing (Fast);
c) Simulated Annealing (Slow);
d) Tunneling network;
e) Adaptive Tunneling network.

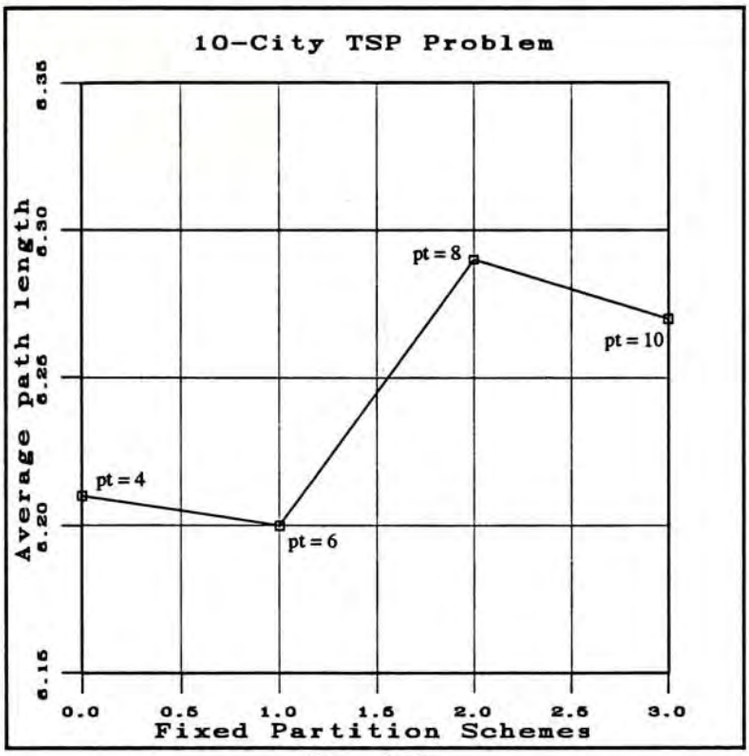

Fig. 4.12a

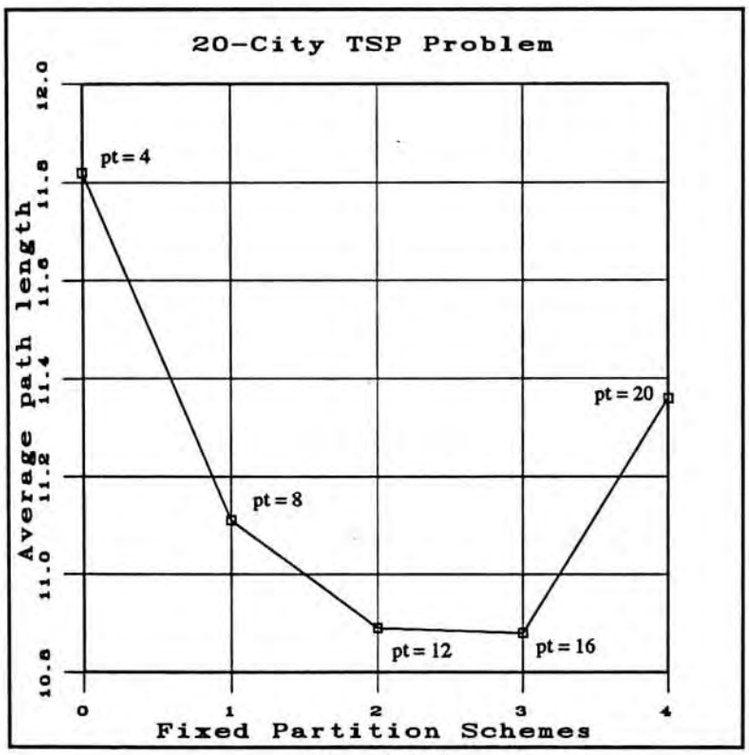

Fig. 4.12b

Figure 4.12: Effect of different partition sizes on the quality of solution obtained. a) 10-city TSP, b) 20-city TSP (Fixed Partition)

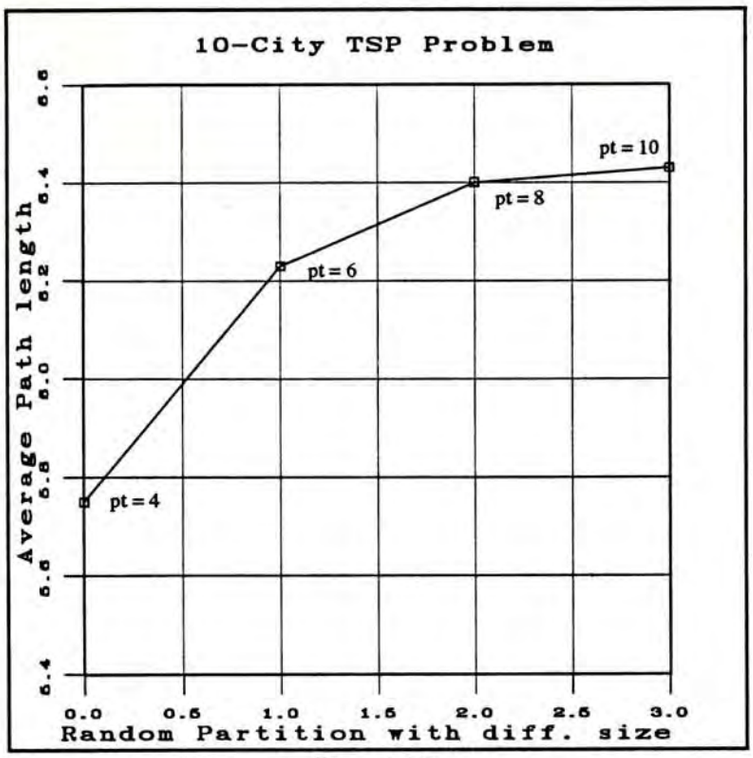

Fig. 4.13a

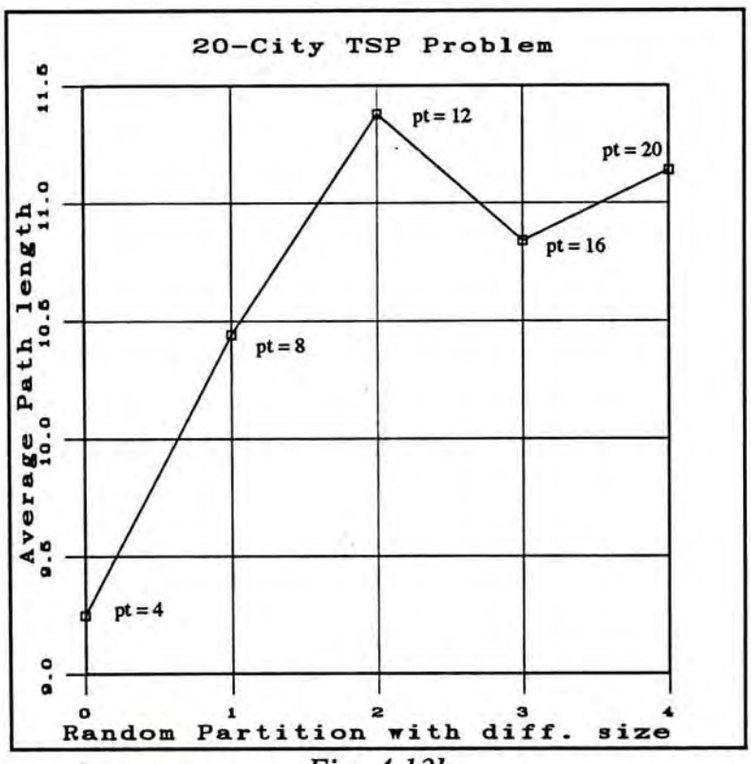

Fig. 4.13b

Figure 4.13: Effect of different partition sizes on the quality of solution obtained. a) 10-city TSP, b) 20-city TSP (Random Partition)

4.5 Suggested hardware implementation of tunneling network :

4.5.1 Tunneling network hardware implementation:

Based on the updating rule of tunneling network, a possible hardware realization is proposed. Although we have tried to make the hardware as simple as possible, we have no intention to claim that our suggestion is the best one but just to show the implementation possibility which is a very important fact for real-time application.

The updating rule of $(n\text{-}1)^{th}$ order tunneling network is restated here,

$$x_i = f_h\left[\sum_k (x_i - x_i^*)^2\left(\frac{\partial E(x)}{\partial x_i}\right) + (1-2x_i^*)(E(x)-E(x^*))\right]$$

i)

$$-\frac{\partial E(x)}{\partial x_i} = \left[\sum_{i_1}\sum_{i_2}\cdots\sum_{i_{n-1}} T_{i i_2 \cdots i_{n-1}} x_{i_1} x_{i_2} \cdots x_{i_{n-1}} + \cdots \sum_{i_1} T_{i_1 i} x_{i_1} + I_i\right]$$

This can be implemented simply by a Hopfield network.

ii)

$$\sum_k (x_i - x_i^*)^2$$

This can be implemented by a set of n EXOR gates and an analog summer. (Refer to Fig. 4.15)

iii)

$$E(x) - E(x^*)$$

This can be implemented by extracting information from Hopfield network with proper weighting. (Refer to Fig. 4.16 & Fig. 4.17)

The whole network structure is revealed in Fig. 4.14.

For the hardware implementation of Adaptive Tunneling network, we are also starting with the corresponding updating rule,

$$x_i = f_h\left[(1-2x_i)(1-K^{(1-2x_i^*)(1-2x_i)})(E(x)-E(x^*)) + \frac{\partial E(x)}{\partial x_i}\right]$$

i)

$$\boxed{(1-K^{(1-2x_i^*)(1-2x_i)})}$$

This can be implemented by a scaling component with gain switching between 1-K and 1-1/K while the switching is controlled by the value of x_i and x_i^*. (Refer to Fig. 4.18)

The whole network structure is revealed in Fig. 4.19.

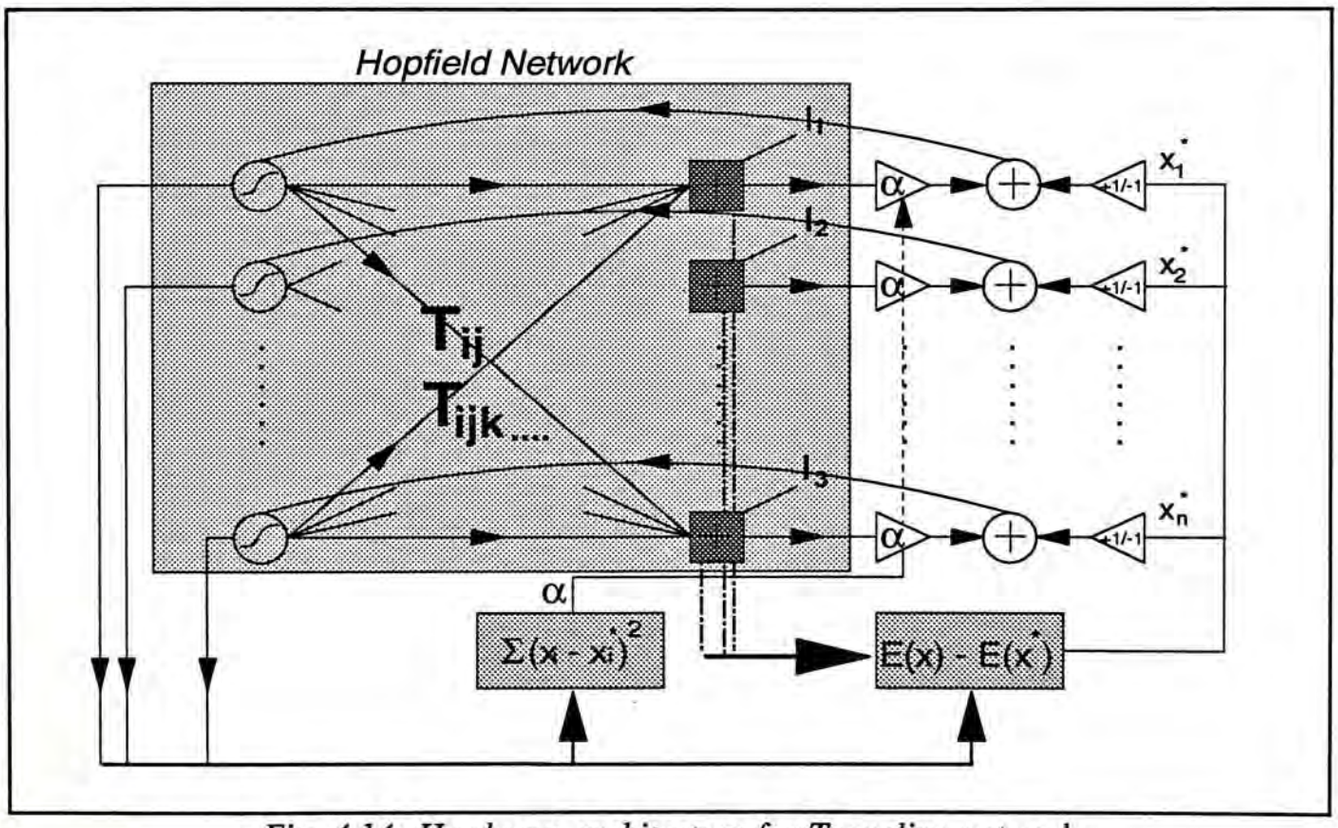

Fig. 4.14: Hardware architecture for Tunneling network

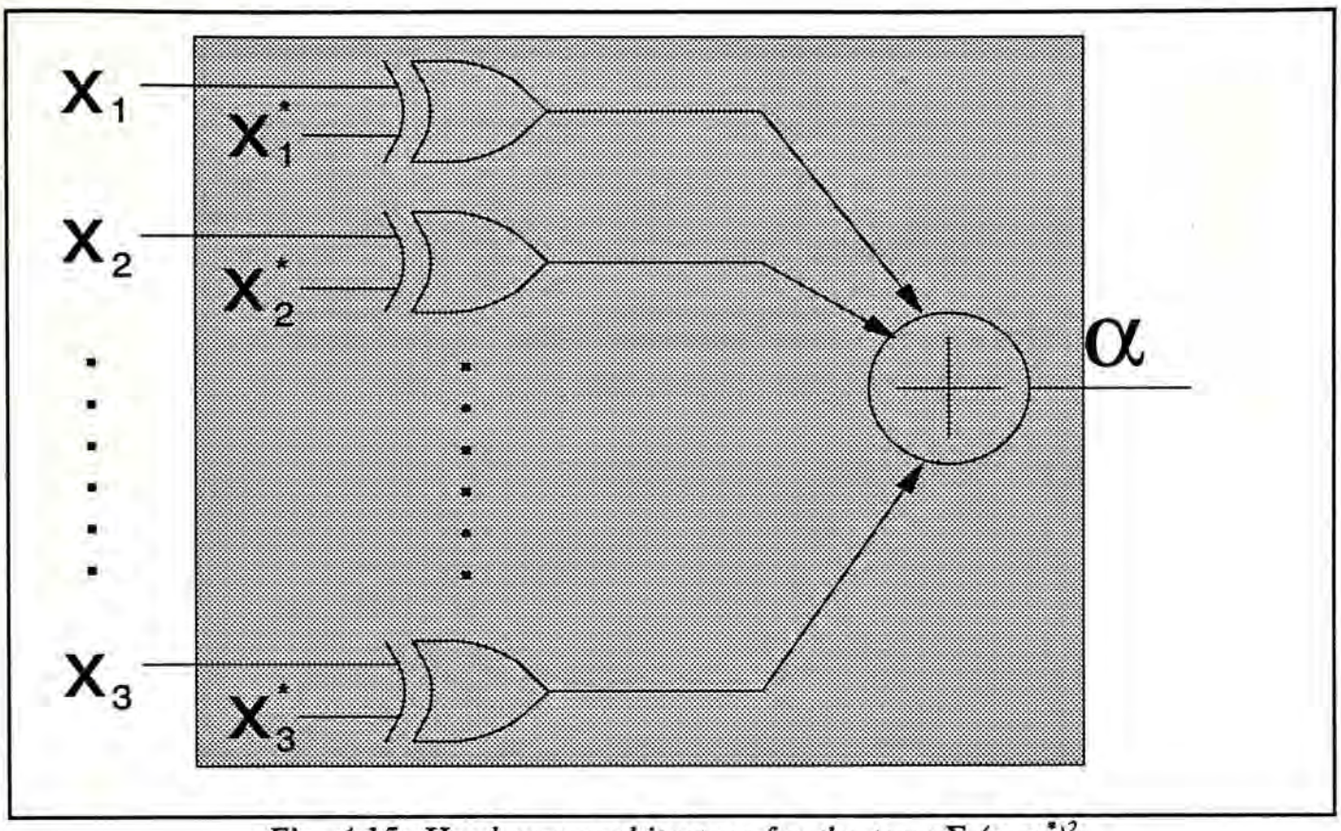

Fig. 4.15: Hardware architecture for the term $\Sigma\ (x_i - x_i^*)^2$

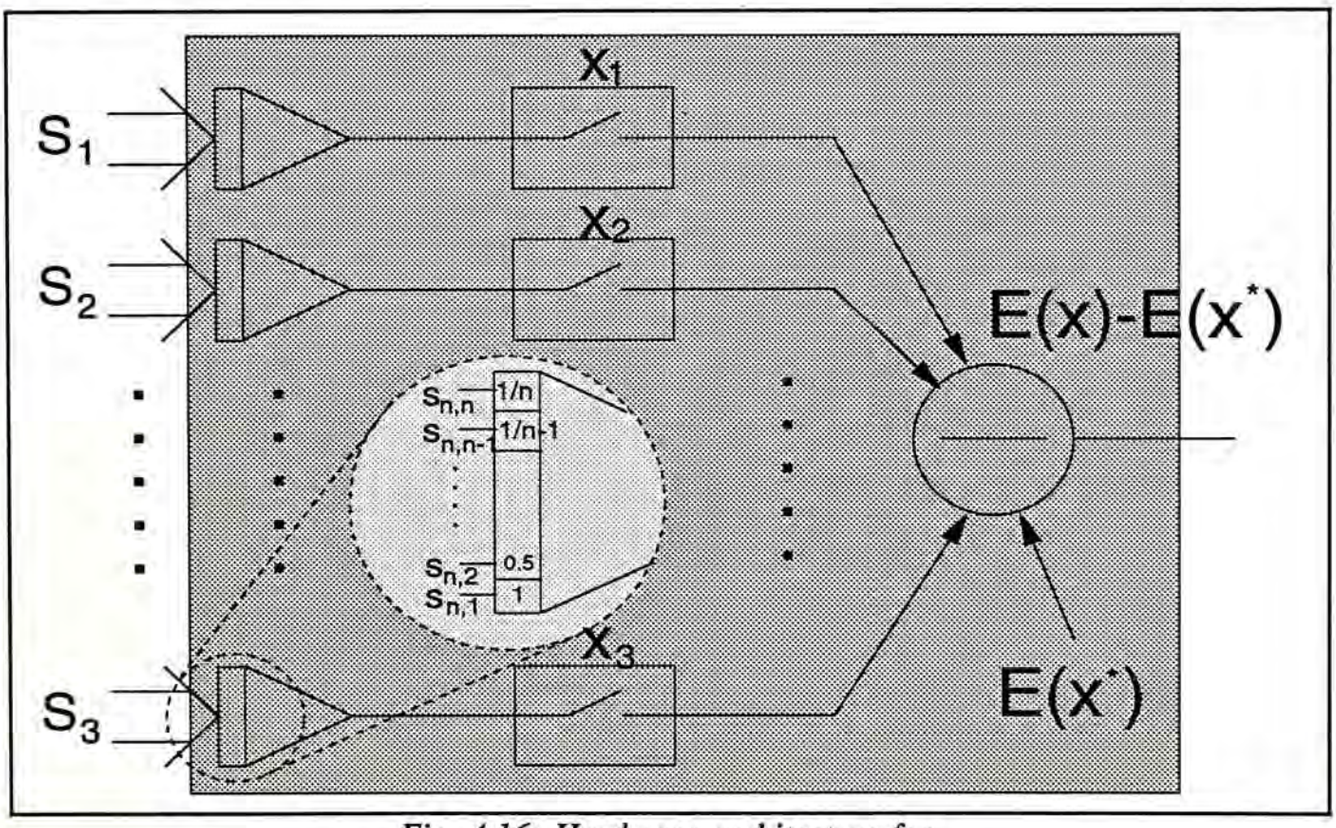

Fig. 4.16: Hardware architecture for the term $E(x)$-$E(x^*)$

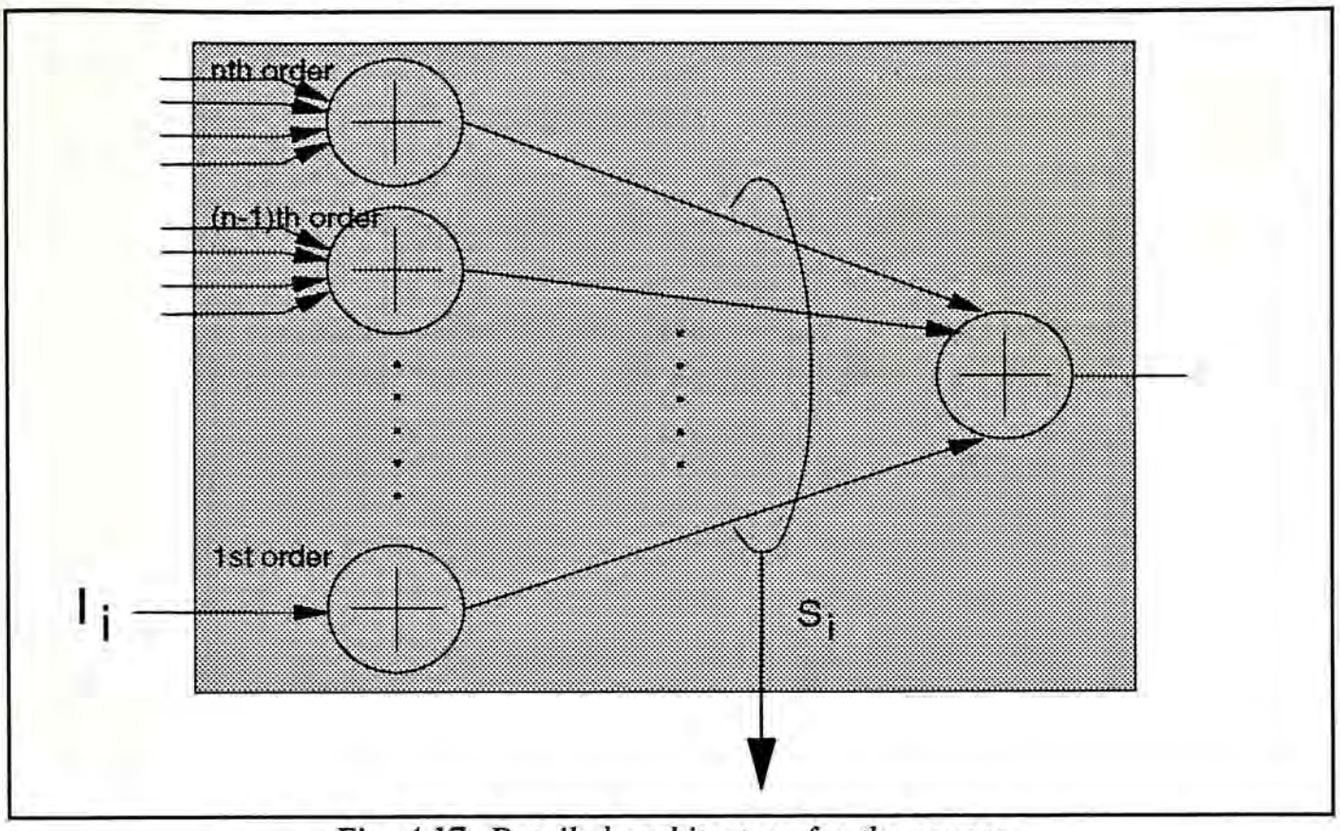

Fig. 4.17: Detailed architecture for the square summer in Fig. 4.19

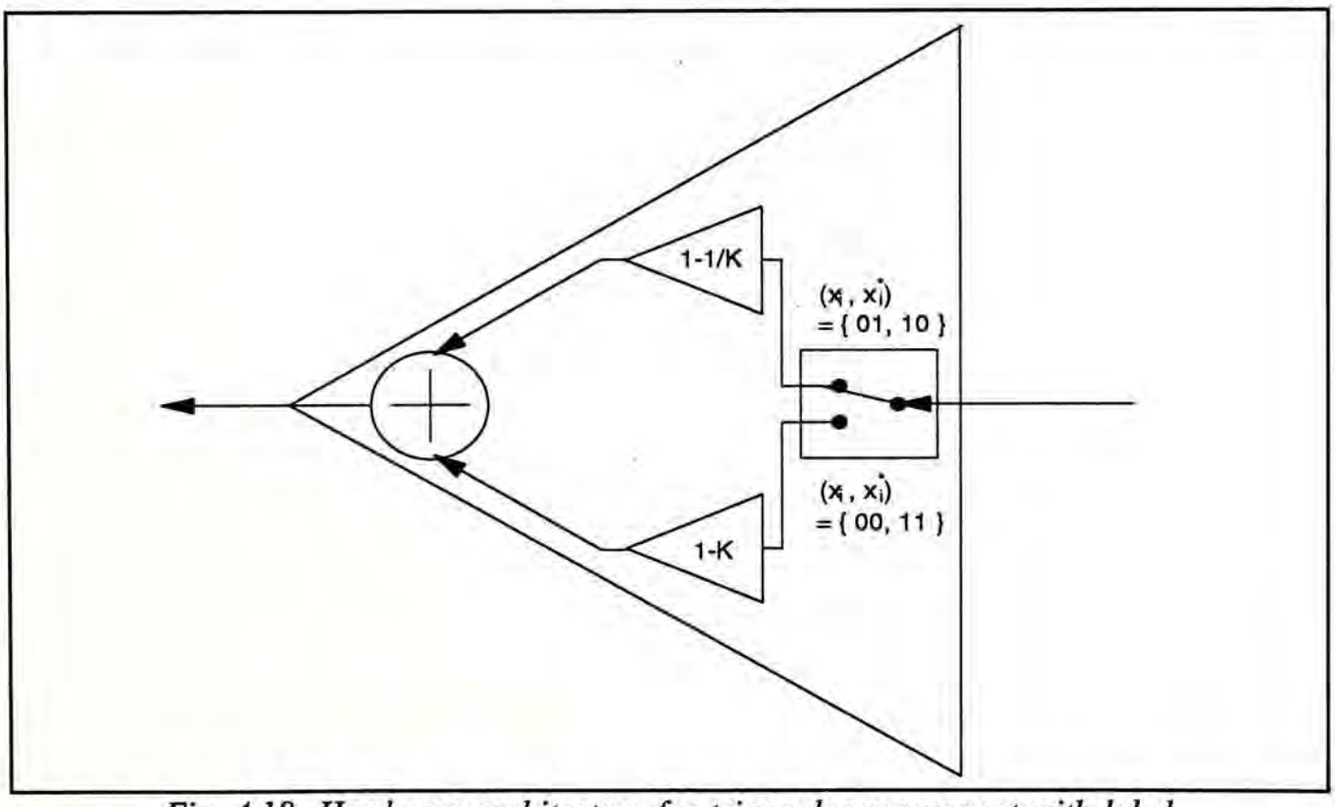

Fig. 4.18: Hardware architecture for triangular component with label K_i in Fig. 4.19

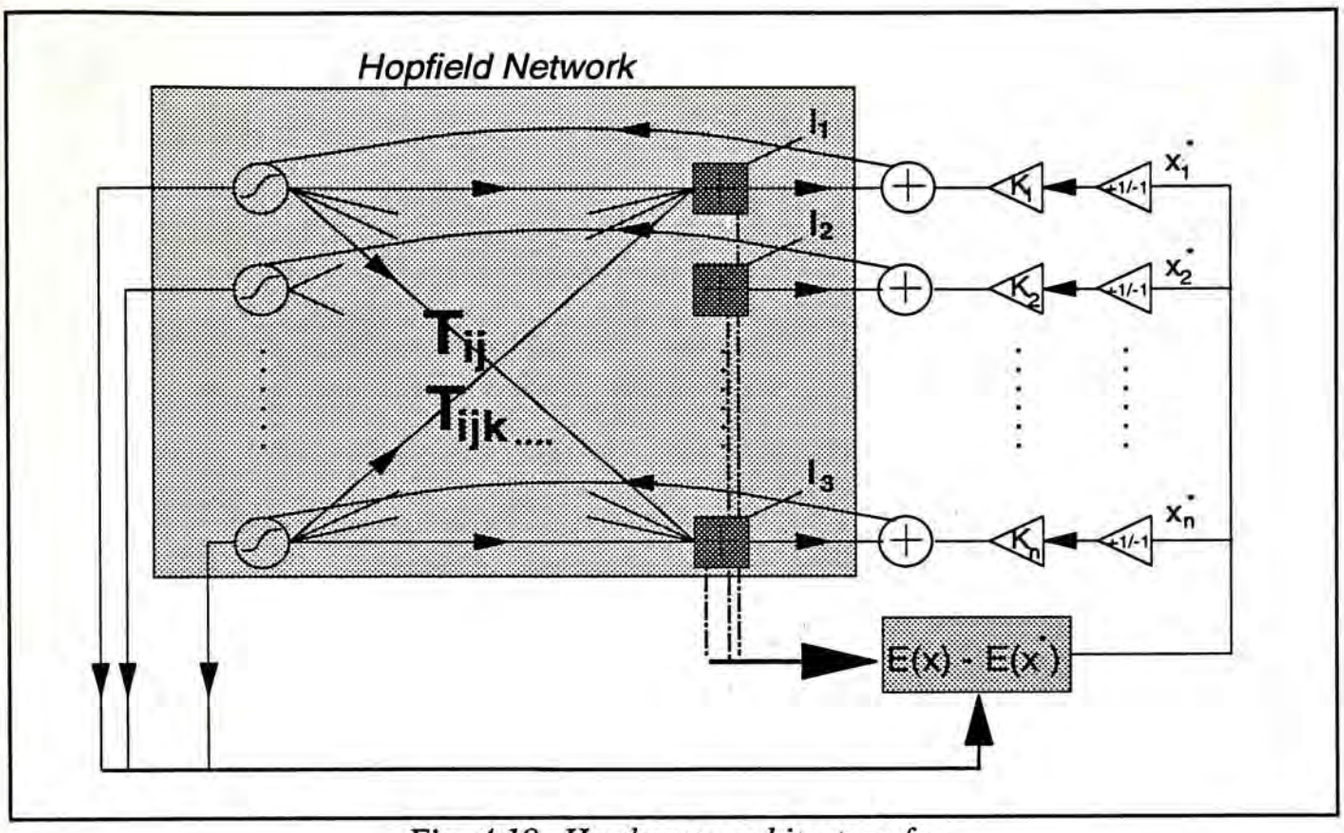

Fig. 4.19: Hardware architecture for Adaptive Tunneling Network

4.5.2 Alternative implementation theory:

With the objective of minimizing the hardware cost, we proposed a possible realization of tunneling network. The structure is obviously quite different from the standard Hopfield-like recurrent neural network. In this subsection, we are going to show that it is possible to realize the Tunneling network by the standard generalized Hopfield network. Same as before, the updating rule is the only thing to be manipulated. Assume the function to be optimized is of nth order and the connection weight symmetric property holds.

By expanding the updating rule of Tunneling network,

$$
\begin{aligned}
x_i = f_h\Bigg[& \sum_{i_1}\sum_{i_2}\cdots\sum i_{n-1}(1-2x_k^*)T_{i_1 i_2 \ldots i_{n-1} i}x_{i_1}\ldots x_{i_{n-1}}x_k + \sum_{i_1}\cdots\sum_{i_{n-2}}\sum_{k}(1-2x_k^*)T_{i_1\ldots i_{n-2}}x_{i_1}\ldots x_{i_k} + \ldots \\
& + \sum_{i_1}\sum_{k}(1-2x_k)T_{i_1 i}x_{i_1} + \sum_{k}(1-2x_k)I_i \\
& + (\sum_k x_k^*)\left(\sum_{i_1}\sum_{i_2}\cdots\sum_{i_{n-1}}T_{i_1 i_2\ldots i_{n-1} i}x_{i_1}\ldots x_{i_{n-1}} + \sum_{i_1}\cdots\sum_{i_{n-2}}T_{i_1\ldots i_{n-2} i}x_{i_1}\ldots x_{i_{n-2}} + \ldots + \sum_{i_1}T_{i_1 i}x_{i_1} + I_i\right) \\
& + (1-2x_i^*)\left(-\frac{1}{n}\sum_{i_1}\sum_{i_2}\cdots\sum_{i_n}T_{i_1 i_2\ldots i_n}x_{i_1}\ldots x_{i_n} - \frac{1}{(n-1)}\sum_{i_1}\cdots\sum_{i_{n-1}}T_{i_1\ldots i_{n-1}} - \ldots - \sum_{i_1}I_{i_1}x_{i_1} - E(x^*)\right)\Bigg]
\end{aligned}
$$

$$
\begin{aligned}
= f_h\Bigg[& \sum_{i_1}\sum_{i_2}\cdots\sum_{i_n}\left((1-2x_{i_n}^*)T_{i_1 i_2\ldots i_{n-1} i} - \frac{(1-2x_i^*)T_{i_1 i_2\ldots i_n}}{n}\right)x_{i_1}x_{i_2}\ldots x_{i_{n-1}}x_{i_n} \\
& + \sum_{i_1}\sum_{i_2}\cdots\sum_{i_{n-1}}\left((1-2x_{i_{n-1}}^*)T_{i_1\ldots i_{n-2} i} + (\sum_l x_l^*)T_{i_1 i_2\ldots i_{n-1} i} - \frac{(1-2x_i^*)}{(n-1)}T_{i_1\ldots i_{n-1}}\right)x_{i_1}\ldots x_{i_{n-1}} \\
& + \sum_{i_1}\sum_{i_2}\cdots\sum_{i_{n-2}}\left((1-2x_{i_{n-2}}^*)T_{i_1\ldots i_{n-3} i} + (\sum_l x_l^*)T_{i_1\ldots i_{n-2} i} - \frac{(1-2x_i^*)}{(n-2)}T_{i_1\ldots i_{n-2}}\right)x_{i_1}\ldots x_{i_{n-2}} \\
& \qquad \vdots \\
& + \sum_{i_1}\left((1-x_{i_1}^*)I_i + (\sum_l x_l^*)T_{i_1 i} - (1-2x_i^*)I_{i_1}\right)x_{i_1} \\
& + (\sum_l x_l^*)I_i + (1-2x_i^*)E(x^*)\Bigg]
\end{aligned}
$$

Therefore, we can build an nth order generalized Hopfield network with the neural network formulation being,

$$
\begin{aligned}
T_{i_1 i_2\ldots i_n i}^{new} &= (1-2x_{i_n}^*)T_{i_1 i_2\ldots i_{n-1} i} - \frac{(1-2x_i^*)T_{i_1 i_2\ldots i_n}}{n} \\
T_{i_1 i_2\ldots i_{n-1} i}^{new} &= (1-2x_{i_{n-1}}^*)T_{i_1\ldots i_{n-2} i} + (\sum_l x_l^*)T_{i_1 i_2\ldots i_{n-1} i} - \frac{(1-2x_i^*)}{(n-1)}T_{i_1\ldots i_{n-1}} \\
& \vdots \\
T_{i_1 i}^{new} &= (1-x_{i_1}^*)I_i + (\sum_l x_l^*)T_{i_1 i} - (1-2x_i^*)I_{i_1} \\
I_i^{new} &= (\sum_l x_l^*)I_i + (1-2x_i^*)E(x^*)
\end{aligned}
$$

Although the symmetric property of the connection weight is not satisfied, it can undergo the reshaping treatment as described in Section 3.2 so that the convergence of the network can be guaranteed. What we have done in this subsection can be concluded as follows: Any Tunneling network corresponding to an $(n\text{-}1)^{th}$ order generalized Hopfield network can be realized by a standard model of nth order generalized Hopfield network. However, in terms of hardware cost, this may not be a good choice because of the tremendous increase in number of connection weights. To have this implementation scheme possible, the advance in technology of implementation of higher-order Hopfield network has to be waited. In fact, recently, implementation of higher-order Hopfield network by optical neurocomputer is being studied and hopefully, their success can make this implementation scheme much more practical. More detailed discussion on this topic can be found in Section 2.4 of Chapter 2.

4.6 Conclusion:

A neural network named Tunneling network (TN), together with a modified adaptive version of it named Adaptive Tunneling network (ATN) are proposed and is shown to have optimal solution seeking capability through simulation. The network is proved to be stable and converge to global minimum state for infinite time. Through extensive performance comparison, TN and ATN are shown to be much more effective than simulated annealing. Between TN and ATN, the former one is found to have higher converging rate while ATN is attractive in terms of the quality of solutions it found. Also, hardware implementation suggestion for both TN and ATN are depicted. In particular, we have shown that $(n\text{-}1)^{th}$ order tunneling network can be implemented by an nth order generalized Hopfield network.

Chapter 5 Recurrent Neural Network for Gaussian Filtering

In this chapter, a recurrent neural network will be designed to perform **Gaussian Filtering**. This application is equivalent to solving a **Convex Quadratic Programming** problem, which belongs to the category of continuous optimization with one and only one local minimum state within the problem domain. In Section 5.1, an introduction on gaussian filtering and some existing neural network models with filtering capability will be given first. Then, the derivation of a recurrent neural network for gaussian filtering, using the problem formulation adopted from regularization theory, is described in Section 5.2. The derived network is simulated and its filtering property is investigated. All the simulation results are illustrated and discussed in Section 5.3.

5.1 Introduction

On extracting information from an image, it's obvious that an image carries information with different scales. Differentiating the information according to different scales can provide a better understanding of an image. For example, consider edge information from an image with a human face. At coarse scale, only the outline of the human face will be expected to be included. At fine scale, outline of the face and also that of the eyes, ears, nose, etc. will also be revealed. So, large-scale feature can be easily judged to be the human face while small-scale features will be the organs on the face. Although this example is posed in a very simplified way as there are a lot of practical considerations in real situation, this shows the main idea of the importance of *scale-space* concept.

This *scale-space* concept, speculated from psychological observation [MH80], has been widely accepted for a long time and proved to have a lot of applications in computer vision and image processing [Wit83] [Ter86b] [RC92]. In particular, Witkin, in 1983, proposed a scale-space filtering. By detecting the zero-crossings of the laplacian of the image filtered at different scale, a scale-space map is created as a representation of the image. The kernel used for the filtering is of gaussian shape. The scale of the kernel is equivalent to the standard derivation of its gaussian shape. It's proved that the kernel being of gaussian shape possesses a nice and unique property that zero crossings will not be created as the scale increased

[PT86] [BWBD86] [YP86]. Moreover, it has been proved that gaussian distributed kernel is an optimal smoothing filter in terms of the localization requirement in both spatial and spectral domains for an image [MH80]. Filtering using gaussian kernel is named as **"Gaussian Filtering"**.

To implement gaussian filtering, the simplest way is to convolve the image with a gaussian distributed weighting template. However, in order to have a precise gaussian template, a large template size will be necessary. This implies heavy calculation required (No. of multiplications needed = template size). Instead of this type of digital implementation, gaussian filtering can be achieved by an analog resister network. Using regularization theory [PTK85], Poggio *et al.* casted the filtering part of the ill-posed edge detection problem as a minimization problem where the energy is the deviation of the expected pixel value and the input pixel value regulated by the smoothness among the neighboring pixels. By minimizing the energy, the minimum state obtained will be a smoothed version of the input image. Besides, the regularization is controlled by a regularization parameter. Varying this parameter will vary the degree of smoothing. In fact, Poggio *et al.* showed that the obtained result is very similar to that obtained by convolving the image with gaussian template. Thus, in fact, the regularization with different regularization parameters can be interpreted as gaussian filtering with different scales. One of the main difference is that the former one has an analog type implementation.

Poggio *et al.* suggested that the minimization can be solved by an analog resistive network. H.Kobayashi *et al.*, in 1991, implemented an active resistive network for gaussian filtering on a silicon chip [KWA91]. Incorporating the regularization theory, they modified the implementation of "Silicon Retina" [MM88], which was proposed by C.A. Mead as an analog model of the first stage of retinal processing. This can be considered as the first time putting neural network model and gaussian filtering together (although gaussian filtering is in fact originated from psychological understanding on human visual system). In the following subsections, we will describe more on Silicon Retina and the active resistive network for gaussian filtering. Then, our implementation using Hopfield network will be illustrated.

5.1.1 Silicon Retina: [MM91]

Study on the characteristics of retina has been carried out by the psychophysiologists for several decades. According to the past knowledge on the structure of the retinas of higher animal, C.A. Mead *et al.* constructed an analog model of the first stages of retinal processing on a single silicon chip. The model, Mead called it "Silicon Retina", is the first neural network model proposed for the early vision processing. In the following, the analogy between biological retina and "Silicon Retina" will be described and the spatially smoothing property of the latter one will be explained.

Biological Retina and Silicon Retina:

The major divisions of the retina can be seen in the Fig. 5.1. Light is transduced into an electrical potential by the "photoreceptors" at the top. The primary signal pathway proceeds from the receptors through the "triad synapses" to the invaginating "bipolar cells", and thence to the ganglian cells.

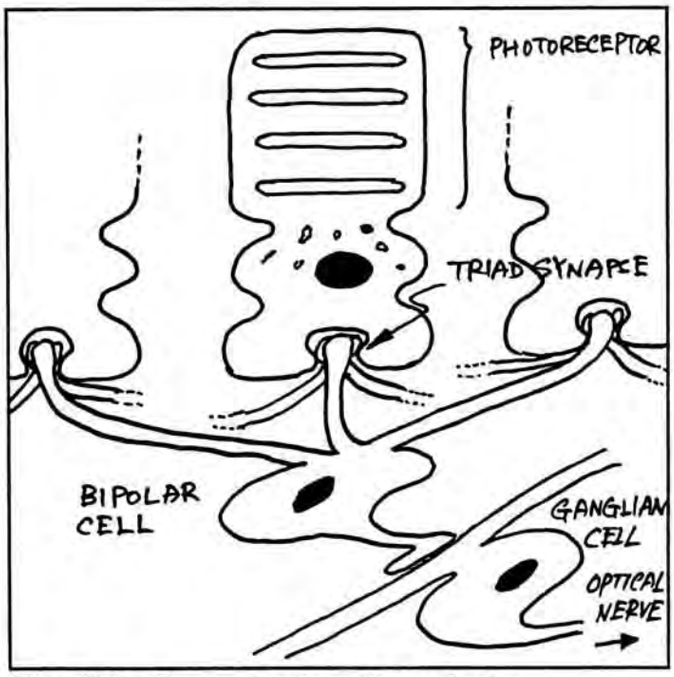

Fig. 5.1: Cross Section through the biological retina

The "Silicon Retina" model is given in Fig. 5.2. It mainly consists of photo sensors, conductance, G, an amplifier and a resistor network (Mead called it electrotonic spread). In Fig. 5.2, the original image is smoothed by the resistor network and the difference between

the image (at point B) and its smoothed version (at point A) is amplified as the network output. Comparing Fig. 5.1 and 5.2, the analogies between the biological model and the silicon model can be summarized in the following table.

Biological	Silicon Retina
Photoreceptors	Photo Sensors
Bipolar Cells	Resistor network
Triad Synapses	A conductance & an amplifier

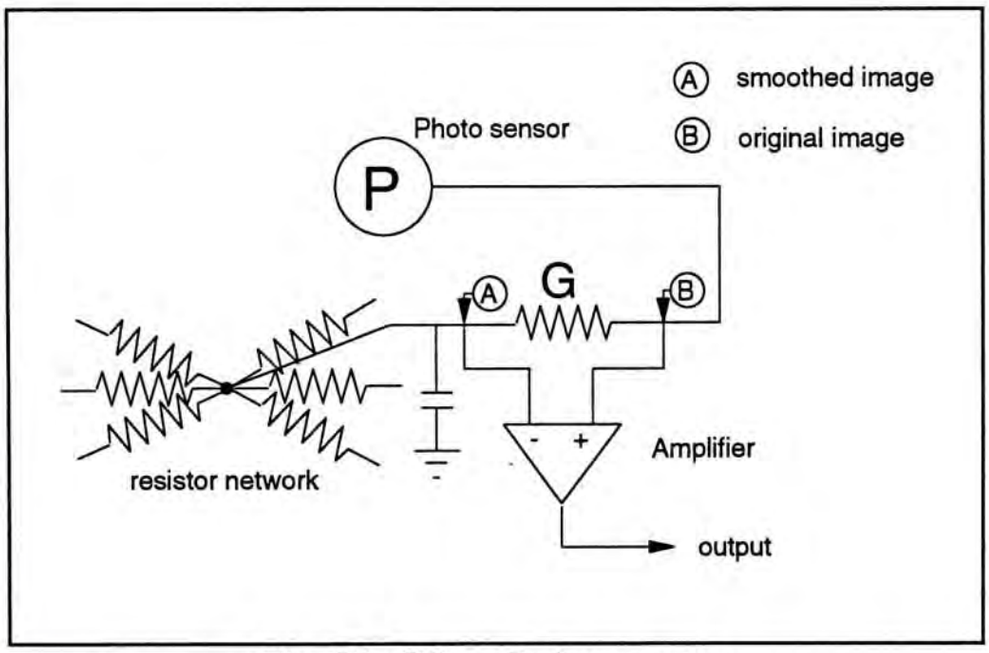

Fig. 5.2: Silicon Retina structure

Spatial Impulse Response of "Silicon Retina" :

To study the spatial impulse response of the network, the 1-D "Silicon Retina" model presented by Mead *et al.* is utilized and it is repeated here in Fig. 5.3a for reference. The corresponding spatial impulse response of the resistor sheet is given by the expression,

$$\exp(-nR_1/R_o)$$

where n is the number of nodes away from the excitation.

It is a decaying characteristics of the network due to the continuous leakage to ground.

The spatial impulse response can be viewed as the convolution kernel of the resistive network and it's illustrated in Figure 5.3b. From the filtering perspective, "Silicon Retina" is performing low-pass filtering/smoothing with exponential decaying kernel.

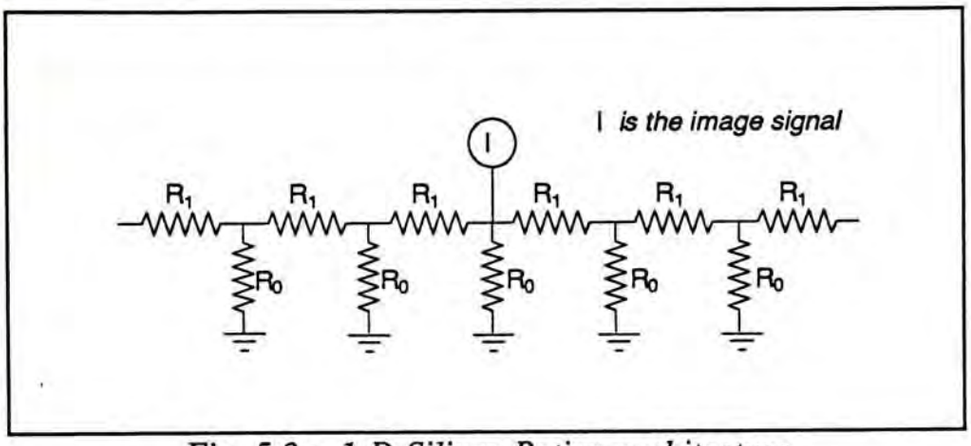

Fig. 5.3a: 1-D Silicon Retina architecture

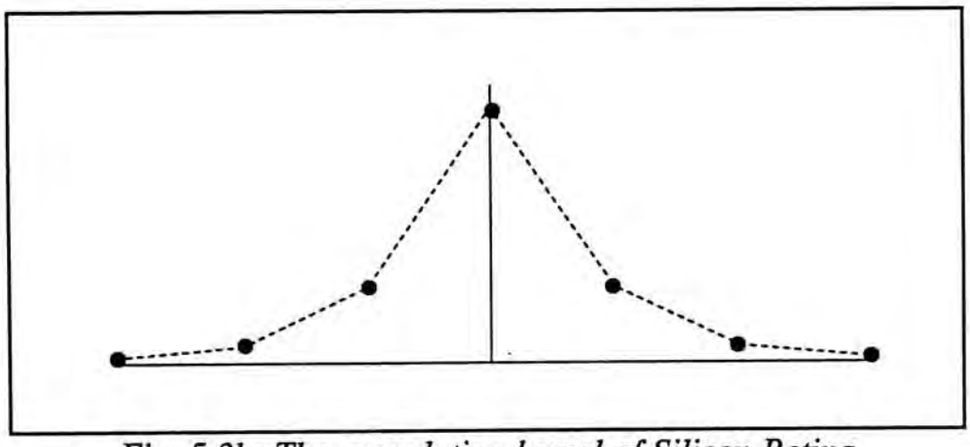

Fig. 5.3b: The convolution kernel of Silicon Retina

Mead *et al.* implemented the model on a silicon chip and compared the response of the chip with the data from the experimental measurement of the retina's response. The result is found to be qualitatively similar. Thus, the first stage of retinal processing involves smoothing is a rather confirmed fact. However, "Is the real model of retina so simple?" remains to be a question to answer.

5.1.2 An Active Resistor Network for Gaussian Filtering of Image: [KWA91]

H. Kobayashi et al. putted together the "Silicon Retina" model and the "nice property" of gaussian filter. He suggested a new resistive network with gaussian-like kernel. Its structure

is shown in Fig. 5.4a and its kernel is shown in Fig. 5.4b. The main difference of this network from "Silicon Retina" is that there are negative resistors between second nearest neighbours of the resistive network. Although there is no evidence obtained so far proving the fact that our retina is preforming gaussian filtering, the research on gaussian filter is partially originated from the psychophysiological observation [MH80]. Drawing their relationship together may not be just a coincidence.

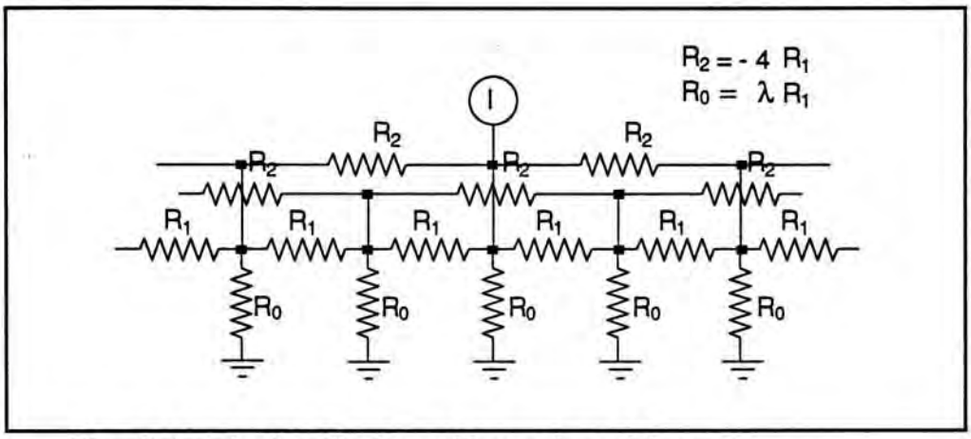

Fig. 5.4a: Active Resistor Network architecture

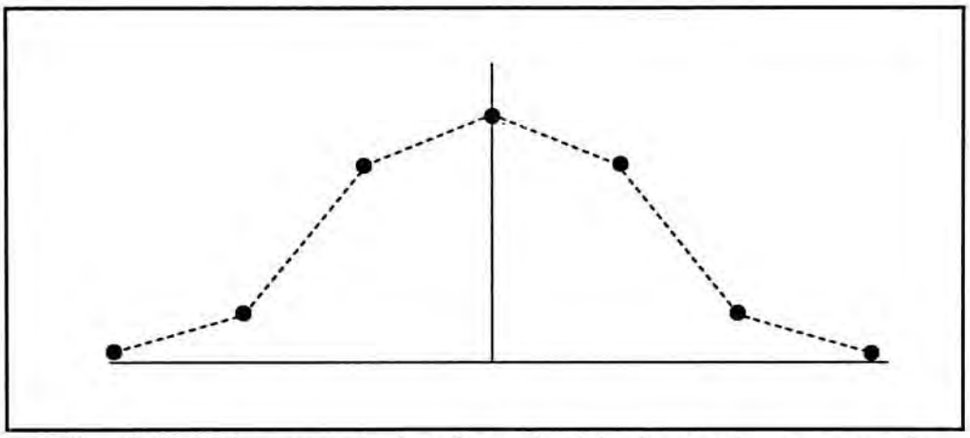

Fig. 5.4b: The convolution kernel of Active resistor network

Design the network using energy minimization approach:

H. Kobayashi utilized the result obtained by Poggio *et al.* [PTK85] [PT86] to design the resistive network. Poggio casted the gaussian filtering as a regularization problem, which can be solved by energy minimization approach. The energy function is defined as,

$$E = \sum_j \left(U(x=j) - V_j \right)^2 + \lambda \int \left(\frac{d^2U}{dx^2} \right)^2 dx \qquad ..(5.1)$$

The first term being the least-mean-square difference of the sample points V_j and the fitting function $U(x=j)$ is the data error or external error. The second term being the square of the second derivative of $U(x)$ is the internal error and can be viewed as the constraint that suppresses the excessive fluctuation of $U(x)$ and ensure the continuity of the first derivative of $U(x)$. λ, being the regularization parameter governs the degree of the constraint application. It had been shown [PTK85] that the solution of minimizing eqn.(5.1) is very similar to the result of convolving { V_j } with gaussian kernel. The spread of the gaussian kernel increases as λ increases.

To acquire the solution of minimizing eqn.(5.1), Haruo Kobayashi adopted the Kirchhoff's laws from circuit theory, which states that the constituent relations of the components drive a network to a state of minimum energy dissipation. He constructed the active network whose energy dissipation is described by eqn.(5.1).

Using a discrete estimate of the second derivative in eqn.(5.1), we get,

$$E = \sum_j (U_j - V_j)^2 + \lambda \sum_j (U_{j+1} + U_{j-1} - 2U_j)^2 \qquad ..(5.2)$$

where $U_j = U(x=j)$. This is a quadratic form, and therefore has a unique minimum where, $\frac{\partial E}{\partial V_j} = 0$ for all j, so,

$$0 = 2\ (U_j - V_j) + \lambda \frac{\partial}{\partial U_j} \sum_i (U_{i+1} + U_{i-1} - 2U_i)^2 \qquad ..(5.3)$$

Differentiating the terms in the sum and noting that $\frac{\partial U_i}{\partial U_j} = 0$ if $i \neq j$.

$$0 = (U_j - V_j) + \lambda \left(6U_j - 4(U_{j-1} + U_{j+1}) + (U_{j-2} + U_{j+2}) \right) \qquad ..(5.4)$$

This describes the node equation of a one-dimensional mesh in Figure 5.4a.

5.1.3 Motivations of using recurrent neural network:

In the following section, we are going to implement gaussian filter using a Hopfield network. Although there already exists an active resistive network for gaussian filtering, we got several motivation of the implementation.

Firstly, it can be considered as a representative of continuous convex quadratic programming problems which is characterized by possessing one and only one local minimum state situated in the interior of the unit hypercube. Issues involved in the implementation of it to a certain extend are generic to problem within this category, ie. the neuron model, boundary connection assignment, etc. which are going to be discussed in more details.

Secondly, recurrent neural network model is a much more general one compared to resistive network model. Increasing the order of the network can extend its application to an even wider range of tasks. This generalization suggested the possibility that the future recurrent neural chip may possess similar role as that of the general-purpose microprocessor of the nowadays conventional digital computer.

Thirdly, in Section 5.1.1, it has already been pointed that whether the real model of retina is as simple as "Silicon Retina" - a resistor network, remains to be a question. However as the idea of the resistor network model is originated from physiological observation, there is a strong reason that any suggested network model for replacement must be structurally similar to that of the resistive network model. Recurrent neural network, in fact, can also be viewed as a resistive network. From Fig. 5.5, the proposed recurrent neural network model has the local connection property as that of the resistor network model although the way of connection is different. So, using recurrent neural network to replace the resistive network model is not a totally new thing and the basis of the physiological observations which triggers the establishment of the resistive network model inherited to the recurrent neural network model we proposed.

5.1.4 Difference between the active resistor network model and recurrent neural network model for gaussian filtering

The active resistive network utilizes the network property of driving itself to a state of minimum energy dissipation. The term "energy" is physically meaning the electrical energy. However, for recurrent neural network, it utilizes the dynamics of the network which is brought by its feedback structure to minimize an "energy" function. The energy function is a theoretical concept and doesn't mean any physical form of energy. So, the two network models utilizes different network's characteristics to achieve the minimization task and that's the main intrinsic difference of the two.

5.2 From Problem formulation to Neural Network formulation

The procedure of implementing 1-D and 2-D gaussian filters using Hopfield network are shown as follows.

5.2.1 One Dimensional Case

Problem Formulation:

According to Poggio's work [PTK85] [PT86], the energy function corresponding to gaussian filter is given by,

$$E = \lambda \sum_j |\hat{d}_j - d_j|^2 \left(\frac{1}{\sigma_{ext}}\right) + (1-\lambda) \sum_j |\hat{d}_{j+1} - 2\hat{d}_j + \hat{d}_{j-1}|^2 \left(\frac{1}{\sigma_{int}}\right) \qquad ..(5.5)$$

where $\hat{d}_j$: the value of jth neuron

d_j : the value of jth sample point of the original data

σ_{ext} & σ_{int} : normalization constant for external error and internal error

Recurrent neural network model used:

The feedback architecture of first order Hopfield network is adopted and two different neuron models are incorporated, one being discrete and one being continuous. For the discrete one, Paik's neuron model is used and for the continuous one, shifted sigmoid function with

upper and lower limit being 255 and 0 is used[1]. Each neuron will be representing one single pixel. Detailed description of the neuron models can be referred to Section 2.1 in Chapter 2.

Neural Network formulation:

Taking partial derivative of E with respect to $\hat{d}_i$ to obtain the dynamic equation of the corresponding network,

$$\frac{\partial E}{\partial \hat{d}_i} = \frac{\lambda}{\sigma_{ext}} \frac{\partial}{\partial \hat{d}_i}\left(.. + |\hat{d}_i - d_i|^2 + ..\right) + \frac{1-\lambda}{\sigma_{int}} \frac{\partial}{\partial \hat{d}_i}\left(.. + |\hat{d}_{i+1} - 2\hat{d}_i + \hat{d}_{i-1}|^2 + |\hat{d}_i - 2\hat{d}_{i-1} + \hat{d}_{i-2}|^2 + |\hat{d}_{i+2} - 2\hat{d}_{i+1} + \hat{d}_i|^2 + ..\right)$$

$$= \frac{\lambda}{\sigma_{ext}}\left(2\hat{d}_i - 2d_i\right) + \frac{1-\lambda}{\sigma_{int}}\left(12\hat{d}_i - 8\hat{d}_{i+1} - 8\hat{d}_{i-1} + 2\hat{d}_{i+2} + 2\hat{d}_{i-2}\right) \qquad ..(5.6)$$

Therefore, from the dynamic equation eqn.(5.6), the neural network formulation is given by:

$$\boxed{\begin{aligned} &Let\ \alpha = \frac{\lambda}{\sigma_{ext}}, \quad \beta = \frac{1-\lambda}{\sigma_{int}} \\ &T_{ii} = -12\beta - 2\alpha \\ &T_{i+1,i} = T_{i-1,i} = 8\beta \\ &T_{i+2,i} = T_{i-2,i} = -2\beta \\ &I_i = 2\alpha d_i \end{aligned}} \qquad ..(5.7)$$

The network architecture can be represented by Fig. 5.5. Local connectivity characteristics of the network can be revealed.

[1] The definition of the sigmoid gain used in this chapter is just the recipical of the one used in Section 2.2.1.

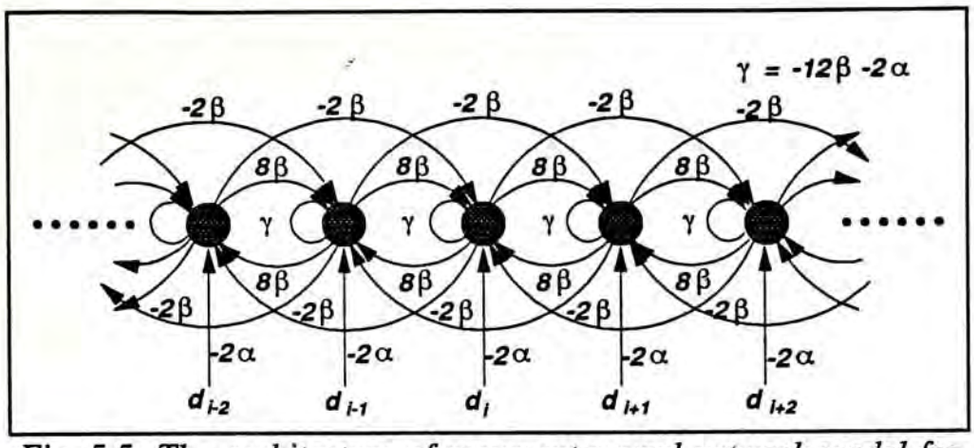

Fig. 5.5: The architecture of recurrent neural network model for gaussian filtering

Convergence of the network and boundary connection :

According to Theorem 2.3, the discrete recurrent neural network model we used is proved [PK92] to converge to a local minimum (or in other words, the corresponding energy function is a Lyapunov one) as long as the following two conditions are satisfied;

C.1 The connection weights are symmetric, ie. $T_{ij} = T_{ji}$

C.2 The value of self-feedback weight, T_{ii} must not be greater than zero.

For the second condition, by examining the neural network formulation of the network eqn.(5.7),

$$T_{ii} = \frac{-12(1-\lambda)}{\sigma_{int}} + \frac{-2\lambda}{\sigma_{ext}} \leq 0$$

Thus, C.2 is satisfied.

The problem left is the first condition C.1. Again referring to eqn.(5.7), the network's connection weight matrix remains to be symmetric until the boundary connections are encountered. The nature of the problem encountered is similar to the situation of setting the boundary condition in image processing. However, the consequence is comparatively more serious. Breaking the symmetric rule will incur nonconverging phenomenon of the network. So, the strategy for assigning the boundary connection has to take condition C.1 into consideration.

At the boundary, i.e. the first two neurons and the last two neurons, the energy to be minimized is different from eqn.(5.5) and thus the connection weights are different.

Let E_0 be the terms in E involving the first neuron

E_1 be the terms in E involving the second neuron

$$E_0 = \alpha\left(\hat{d}_0 - d_0\right)^2 + \beta\left(\hat{d}_2 - 2\hat{d}_1 + \hat{d}_0\right)^2$$

$$\begin{aligned}\frac{\partial E_0}{\partial d_0} &= 2\alpha\left(\hat{d}_0 - d_0\right) + 2\beta\left(\hat{d}_2 - 2\hat{d}_1 + \hat{d}_0\right) \\ &= 2(\alpha+\beta)\hat{d}_0 - 4\beta\hat{d}_1 + 2\beta\hat{d}_2 - 2\alpha d_0\end{aligned}$$

Therefore, the connections involving the first neuron are;

$$\begin{aligned} T_{00} &= -2(\alpha+\beta) \\ T_{01} &= 4\beta \\ T_{02} &= -2\beta \\ I_0 &= 2\alpha d_0 \end{aligned} \qquad ..(5.8)$$

$$E_1 = \alpha\left(\hat{d}_1 - d_1\right)^2 + \beta\left\{\left(\hat{d}_2 - 2\hat{d}_1 + \hat{d}_0\right)^2 + \left(\hat{d}_3 - 2\hat{d}_2 + \hat{d}_1\right)^2\right\}$$

$$\begin{aligned}\frac{\partial E_1}{\partial d_1} &= 2\alpha\left(\hat{d}_1 - d_1\right) - 4\beta\left(\hat{d}_2 - 2\hat{d}_1 + \hat{d}_0\right) + 2\beta\left(\hat{d}_3 - 2\hat{d}_2 + \hat{d}_1\right) \\ &= -4\beta\hat{d}_0 + (10\beta + 2\alpha)\hat{d}_1 - 8\beta\hat{d}_2 + 2\beta\hat{d}_3 - 2\alpha d_1\end{aligned}$$

Therefore, the connections involving the second neuron are;

$$\begin{aligned} T_{10} &= 4\beta \\ T_{11} &= -10\beta - 2\alpha \\ T_{12} &= 8\beta \\ T_{13} &= -2\beta \\ I_1 &= 2\alpha d_1 \end{aligned} \qquad ..(5.9)$$

The connections at the other boundary are assigned similarly and the connection matrix generated this way remains to be symmetric and thus the corresponding energy is still Lyapunov function.

For the continuous model, according to the energy function (5.5), as it is quadratic and always greater than or equal to zero for any $\hat{d}$, it's a convex quadratic function. Thus, the Hessian of E will be a positive semi-definite matrix and the connection matrix, which is negative of the Hessian of E will then be negative semi-definite. It's proved that a Hopfield network with negative semi-definite symmetric connection weight matrix will converge to a local minimum of E. The symmetric property of the connection weight matrix is just and thus the guarantee of the convergence of our continuous model of gaussian filter follows. Besides, the local minimum is unique for E, ie. it's the global minimum. However, the global minimum state can't be reached by the discrete model as quantization of the neuron state is applied. Thus, filtering using the discrete model will introduce a noise due to quantization.

5.2.2 Two Dimensional Case

As we are interested in applying the network to do gaussian filtering for image. Extending the 1-D network to 2-D counterpart is the natural flow of thinking. The extension we performed is rather straight forwards except that more attention have to be paid to the boundary connection of the network.

To obtain the energy function for 2-D case, we simply include the smoothness constraints of both vertical and horizontal direction in the regularizing term. The corresponding energy function for the 2-D network becomes;

$$E = \frac{\lambda}{\sigma_{ext}}\sum_x\sum_y |\hat{d}_{xy}-d_{xy}|^2 + \frac{1-\lambda}{\sigma_{int}}\sum_x\sum_y \left\{|\hat{d}_{x+1,y}-2\hat{d}_{x,y}+\hat{d}_{x-1,y}|^2 + |\hat{d}_{x,y+1}-2\hat{d}_{x,y}+\hat{d}_{x,y-1}|^2\right\} \quad ..(5.10)$$

The terms in E involving $\hat{d}_{xy}$

$$= \beta\left\{\left(\hat{d}_{x+1,y}-2\hat{d}_{x,y}+\hat{d}_{x-1,y}\right)^2 + \left(\hat{d}_{x+2,y}-2\hat{d}_{x+1,y}+\hat{d}_{x,y}\right)^2 + \left(\hat{d}_{x,y}-2\hat{d}_{x-1,y}+\hat{d}_{x-2,y}\right)^2 + \left(\hat{d}_{x,y+1}-2\hat{d}_{x,y}+\hat{d}_{x,y-1}\right)^2 + \left(\hat{d}_{x,y+2}-2\hat{d}_{x,y+1}+\hat{d}_{x,y}\right)^2 + \left(\hat{d}_{x,y}-2\hat{d}_{x,y-1}+\hat{d}_{x,y-2}\right)^2\right\} + \alpha\left(\hat{d}_{x,y}-d_{x,y}\right)^2$$

$$\frac{\partial E}{\partial \hat{d}_{x,y}} = \beta\left\{24\hat{d}_{x,y}-8\hat{d}_{x+1,y}-8\hat{d}_{x-1,y}-8\hat{d}_{x,y+1}-8\hat{d}_{x,y-1} + 2\hat{d}_{x+2,y}+2\hat{d}_{x-2,y}+2\hat{d}_{x,y-2}+2\hat{d}_{x,y-2}\right\} + 2\alpha\left(\hat{d}_{x,y}-d_{x,y}\right) \quad ..(5.11)$$

From this dynamic equation, the neural network formulation for the 2-D network is given by;

$$\begin{aligned} T_{x,y;x,y} &= -24\beta - 2\alpha \\ T_{x+1,y;x,y} &= T_{x-1,y;x,y} = T_{x,y+1;x,y} = T_{x,y-1;x,y} = 8\beta \\ T_{x+2,y;x,y} &= T_{x-2,y;x,y} = T_{x,y+2;x,y} = T_{x,y-2;x,y} = -2\beta \\ I_{x,y} &= 2\alpha d_{x,y} \end{aligned} \quad ..(5.12)$$

The idea of assigning the boundary connections is more or less the same as the 1-D case. As it is a 2-D network, the assignment is much more tedious. So, the mathematical details are included in Appendix I. To ease the tedious work, we succeeded in deriving formulae for the boundary connection assignment and proved that the connection weight matrix still satisfies the symmetric property. More details can be referred to Appendix II.

5.3 Simulation Results and Discussions

5.3.1 Spatial impulse response of the 1-D network

Spatial impulse responses of 1-D recurrent neural network for gaussian filtering with different regularization parameters' value are illustrated in Fig. 5.6a-d. They are obtained by feeding impulse signal to the network. Simulation is performed using both discrete and continuous model. The result obtained using either discrete or continuous model is consistent to the theoretical expectation, i.e. as the regularization parameter, r (= 1-λ. Refer to Section 5.2.1) increases, the variance of the convolving kernel increases accordingly.

However, it's noted that the convolving kernels of discrete model is not perfectly symmetric as the ones of continuous model. By checking the corresponding network's energy value, it's noted that the result obtained by continuous network is at a lower energy state. So, the performance of continuous model a bit outweighs that of discrete one. However, the simulation time is much longer using continuous model.

Besides, for continuous model, it's also observed that varying the neuron's gain will affect the finally converged state. The phenomenon is illustrated in Figure 5.7a-b. It is observed that for r = 0.5, the variation is not significant. The variation becomes more significant when r is increased to 0.95. To get a more clear picture, we try to compare ideal gaussian shape with the network's convolving kernels. The result is illustrated in Figure 5.8a-b. The y-axis, **mse(std)** is the minimal mean square error between gaussian shape with standard deviation equal **std** and the network's convolving kernels with regularization parameter varying from 0 to 0.95. It is observed that the continuous models with neuron gain = 0.01 and 0.1 approximate gaussian shape better than continuous model with neuron gain = 1 and discrete model. Also, the deviation from gaussian shape is very serious for very small value of **std**.

5.3.2 Filtering property of the 1-D network

A signal with information of different scales is applied to the network to illustrate its

filtering property. From Fig. 5.9a-f, it is observed that as the regularization parameter increases, features with smaller scale are removed. This illustrates the scale-space filtering property of the designed network.

To study its effect to a step edge, step edge signal is applied to the network. The result is presented in Fig. 5.10. There are two findings. Firstly, edge blurring is more serious as r increases. This result again is consistent to our theoretical expectation. Secondly, it's found that there is "overshoot" and "undershoot", ahead and behind a step edge respectively. The existence of negative portions of the filter kernel accounts for the phenomenon. We called it "step edge contrast enhancement effect". This is the characteristics owned by our network but not ideal gaussian filter, and in fact, is a defect of our network in terms of gaussian filtering application as the image filtered by the network will have several zero-crossings in the vincity of one step edge. Then, the accurate position of the edge will be difficult to locate. In order to remove the defect, further development have to be done at the very beginning step, the problem formulation, by introducing some more terms to reshape the network kernel to ideal gaussian-shaped one.

5.3.3 Spatial impulse response of the 2-D network and some filtering results

Similar to the 1-D case, we apply a 2-D impulse signal to 2-D recurrent neural network for gaussian filtering to obtain the spatial impulse response of the network. Again, the result is what we are expecting. As r increases, the spread of the 2-D convolving kernel increases and the result is clearly shown in Fig. 5.11a-e. Besides, it also possesses all the filtering property mentioned in the subsection 5.3.2. We have applied the network to 128x128 "Lenna" image and the smoothed versions with different values of r are shown in Fig. 5.12a-e.

5.4 Conclusions

A recurrent neural network is successfully designed for gaussian filtering. The spatial impulse response for both one-dimensional and 2-dimensional network as well as their filtering properties are investigated and it's observed that they are very similar to the gaussian

filter. Both discrete and continuous model are simulated and their difference is, in general, not very significant except that the former one wins in terms of simulation speed while the latter wins in terms of kernel's quality. Also, boundary connection assignment, which is analogous to the issue of assigning boundary value condition for a lot of image processing algorithms, is found to be the most tedious but important part during the network derivation. Some formula are derived to ease the problem and are given in the chapter.

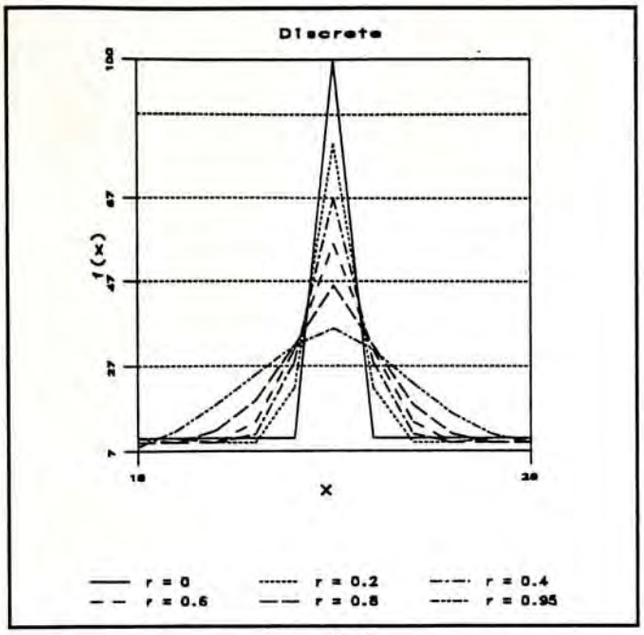

Fig. 5.6a

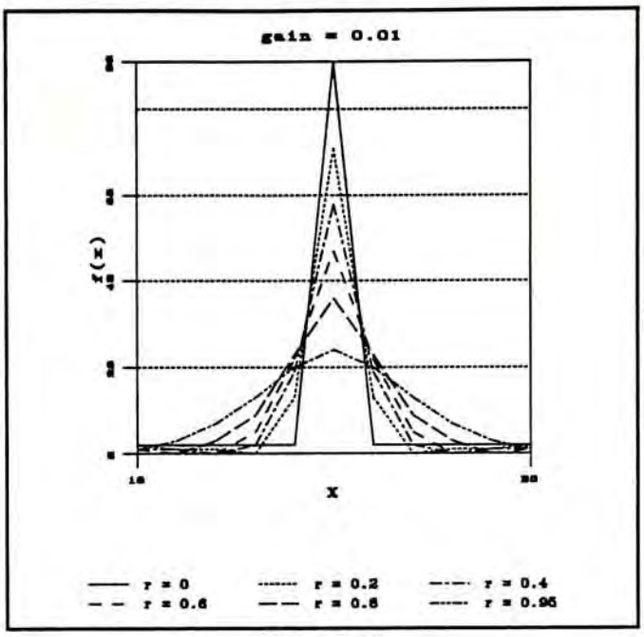

Fig. 5.6b

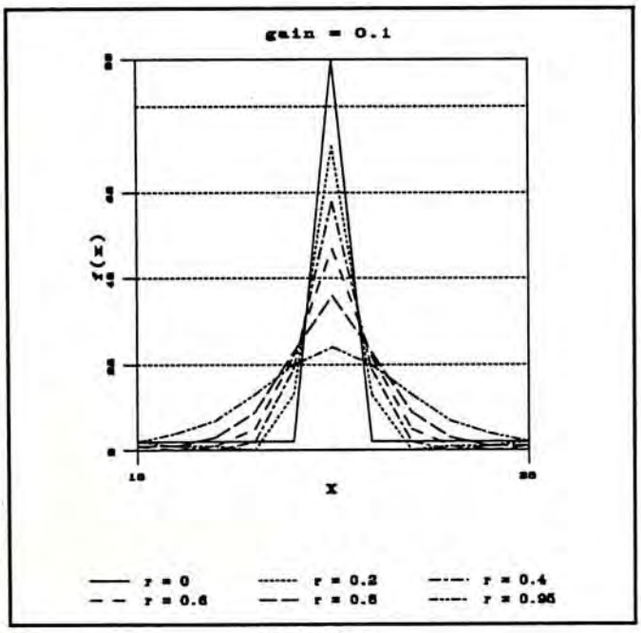

Fig. 5.6c

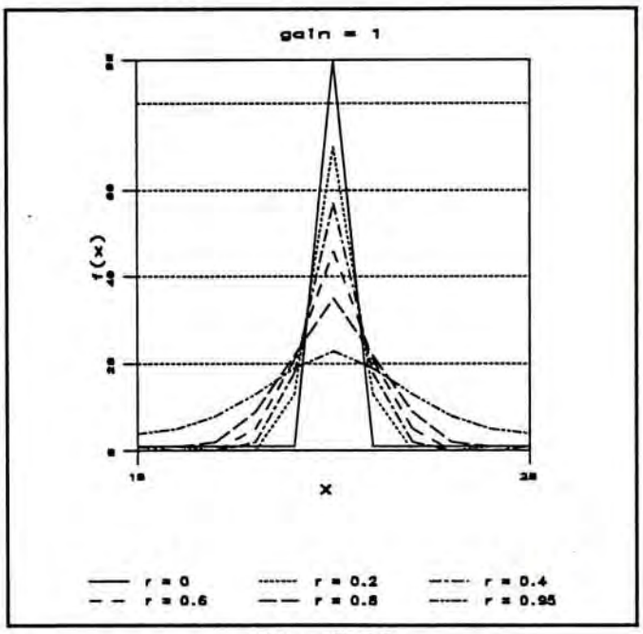

Fig. 5.6d

Fig. 5.6a-d: Spatial impulse response of 1D network with different regularization parameter and different neuron gain
a) discrete; b) gain = 0.01;
c) gain = 0.1; d) gain = 1

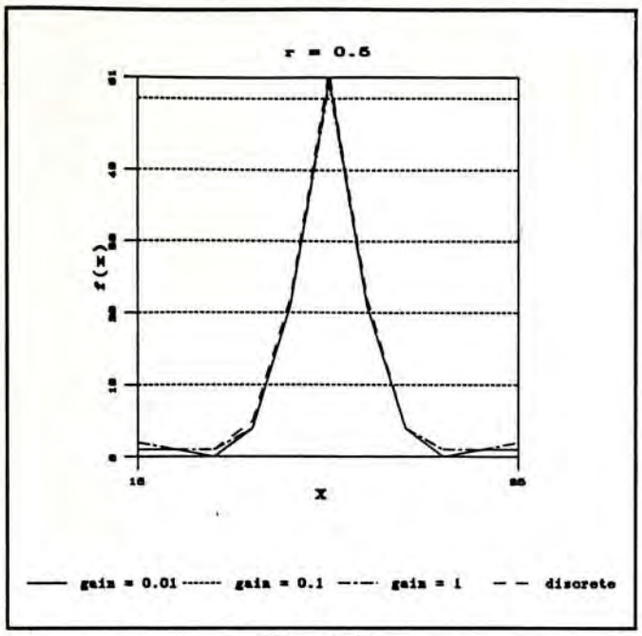

Fig. 5.7a

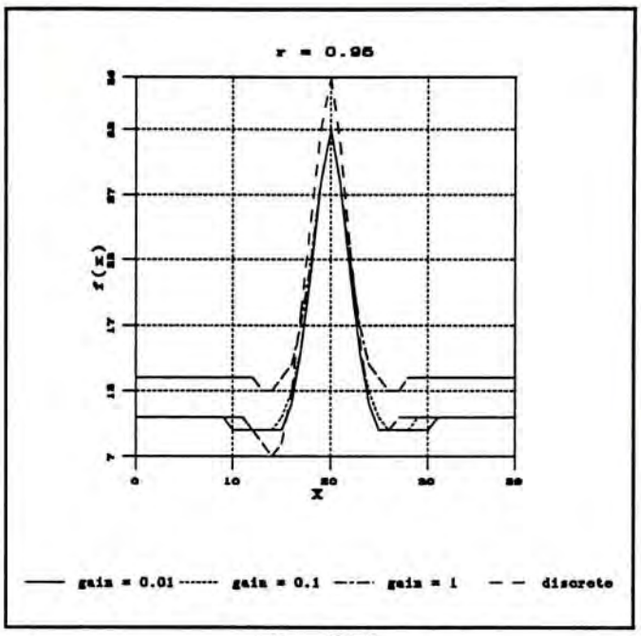

Fig. 5.7b

Fig. 5.7a-b: Comparing spatial impulse response of 1D network with different neuron gain and regularization parameter
e) r = 0.50; f) r = 0.95

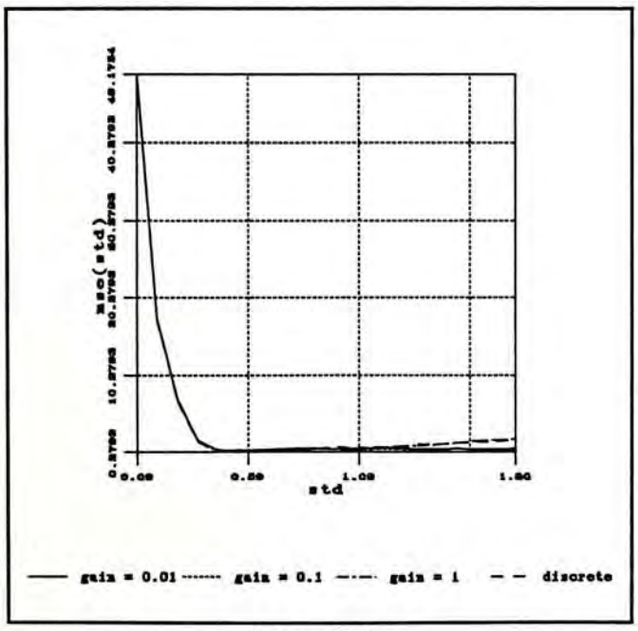

Fig. 5.8a

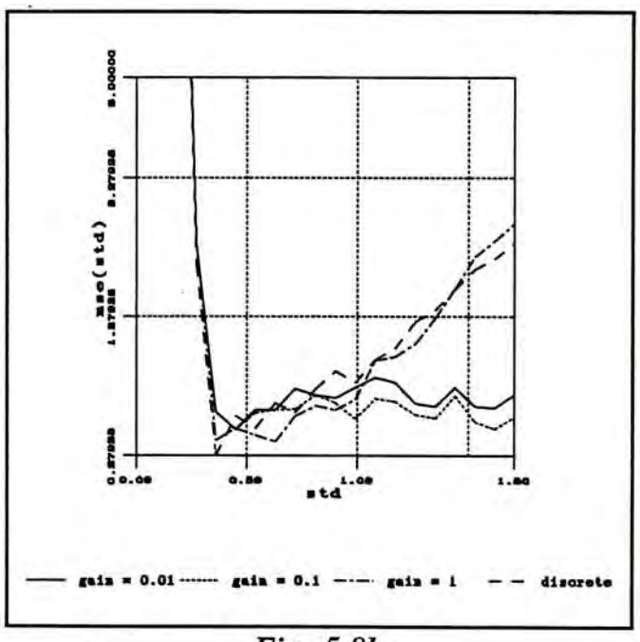

Fig. 5.8b

Figure 5.8a-b: Comparing the mse difference between ideal gaussian kernels of different variance with the kernels of the 1D network. b) is just locally enlarged version of a)

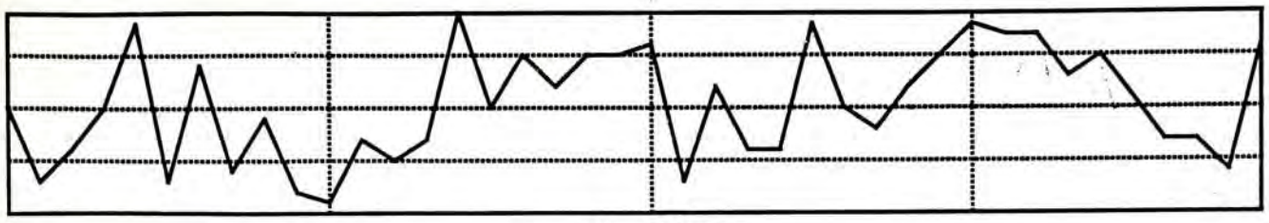

Fig. 5.9a

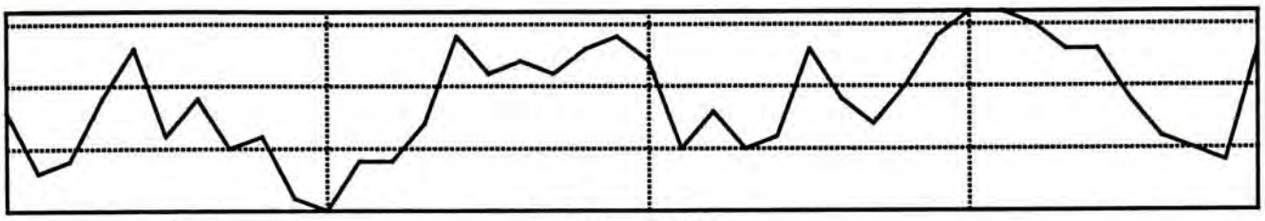

Fig. 5.9b

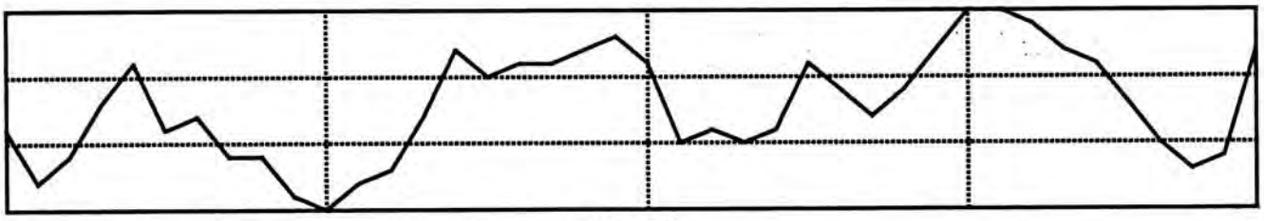

Fig. 5.9c

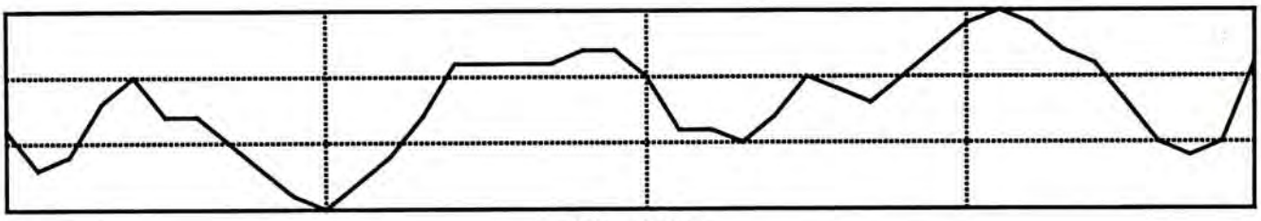

Fig. 5.9d

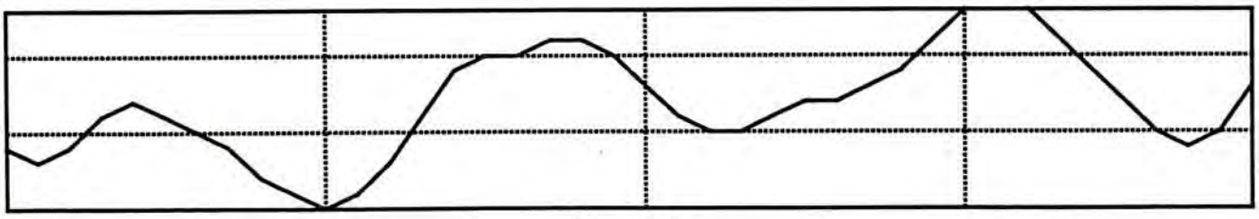

Fig. 5.9e

Figure 5.9a-e: *Filtering of signal with information of different scale. Note that the network with greater regularization parameter removes small scale information. a) r = 0.0; (original) b) r = 0.3; c) r = 0.5; d) r = 0.7; e) r = 0.9;*

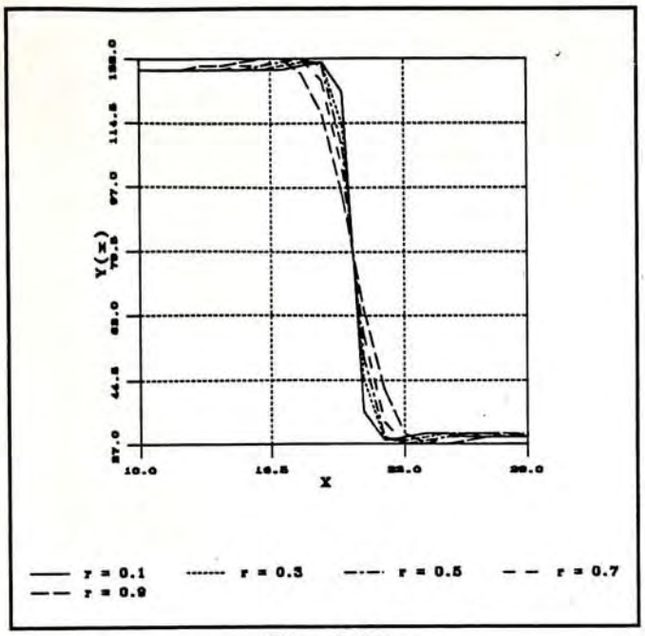

Fig. 5.10

Fig.5.10: *Filtering of step signal. Note that the network creates step edge constrast enhancement effect*

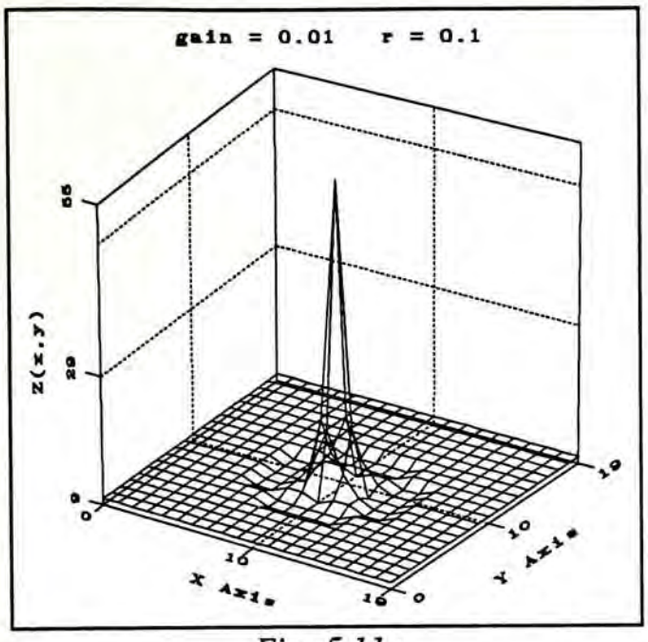

Fig. 5.11a

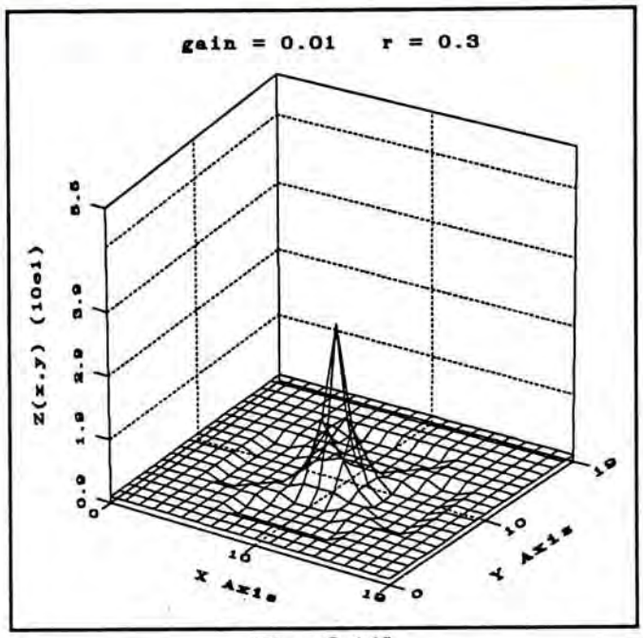

Fig. 5.11b

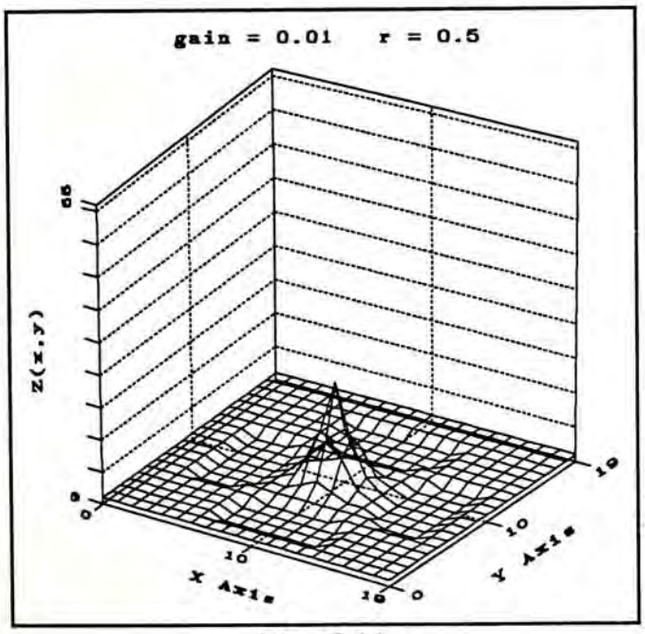

Fig. 5.11c

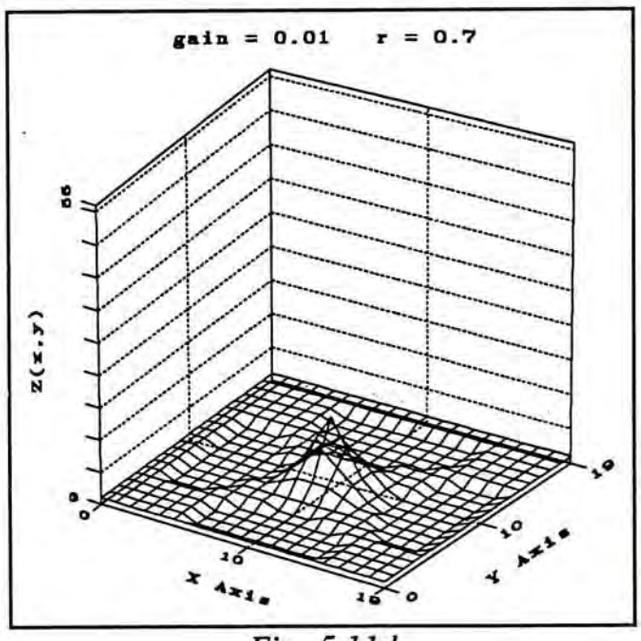

Fig. 5.11d

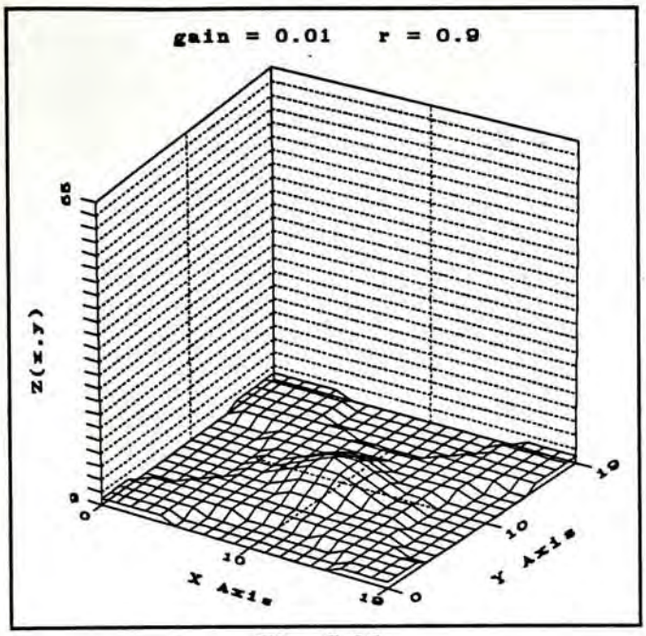

Fig. 5.11e

Figure 5.11a-e : Spatial impulse response of 2-D network with regularization parameter
a) 0.1; b) 0.3; c) 0.5;
d) 0.7; e) 0.9

Fig. 5.12a

Fig. 5.12b

Fig. 5.12c

Fig. 5.12d

Fig. 5.12e

Fig. 5.12: "Lenna" image smoothed by recurrent neural network.
a) r = 0.2; b) r = 0.4; c) r = 0.6; d) r = 0.8; e) r = 0.99

Chapter 6 Recurrent Neural Network for Boundary Detection:

In this chapter, the novel recurrent neural network model, named Tunneling network, derived in Chapter 4 is used to tackle an optimal boundary detection problem. The problem formulation is adopted from that of "active contour model - snake" and is in fact a 3rd order combinatorial programming. As Tunneling network has optimum seeking power, we expect the neural network implemented boundary detection scheme can detect a nearly optimal boundary. In Section 6.1, we will give an introduction on boundary detection using active contour model - snake. Then, the derivation of a recurrent neural network for boundary detection based on the problem formulation of the "Snake" is shown in Section 6.2. Simulation of applying derived network to detect boundaries on both synthesized and real image is performed and performance comparisons are made with existing boundary detection means using local minimization scheme. All the results are illustrated and discussed in Section 6.3.

6.1 Introduction

Extracting important features correctly from an image is a very important step in a computer vision system. Among others, edges and boundaries are the features commonly sought in an image. However, owing to noise, breaks in boundary due to nonuniform illumination, spurious intensity discontinuity, the extracting them from an image is not so straight forward. For boundary detection, most of the classical techniques can be divided into two categories; local analysis, like contour following vs. global analysis, like Hough Transform. They bear their own limitations as the former one, being totally data-driven, will break down when noise and breaks in boundary are significant while the latter one, being totally model-driven, can't capture the local features. In 1988, Kass *et al.* [KWT88] suggested using regularization technique to make a compromise between the two extrema and proposed a boundary detection scheme named "Active contour model - Snake".

A snake is a set of ordered points (named snaxels in the remaining text) with an associated energy consisting two terms;

i) an internal energy term which defines a deformable model constraining the shape of the snake, eg. smoothness and corner sharpness,

ii) an external energy term which pushes the snake to the desired location according to the underlying image data.

By minimizing the energy, the snake will jiggle on the image until it converges. The boundary will then be obtained by interpolating the set of points of the snake according to their orders. The main advantage of using snake to detect object boundary in an image is that it can locate a lot of problematic boundaries, including subjective boundary, which was demonstrated by Kass *et al.* [KWT88].

How to design the internal and external energy terms is one of the crucial steps determining how good a boundary can be detected. Besides the problem formulation, how to do the minimization is also a very important for the quality of the detected boundary and the corresponding computational complexity is important for the possibility of real-time application. Originally, variational calculus [KWT88] [BM90] [Coh91] was used to solve the minimization task and the computational complexity is $O(n)$, where n is the number of snaxels. Amini *et al.* [AWJ90] pointed out that variation calculus method suffers from numerical stability problem and also the uniqueness of the solution can't be guaranteed as external energy term is not convex most of the time. Therefore, they formulate the problem directly on the discrete grid and the minimization becomes a combinatorial one with computational complexity being $O(m^n)$, where m is the size of the searching window for each snaxel. By dynamic programming approach, the minimization problem was then decomposed and casted into a discrete multistage decision process and the computational complexity can be reduced to $O(nm^3)$. Within the area allowing the snake to move, which is determined by the value m, the optimal solution can be guaranteed by the principal of optimality. As m increases, the quality of the detected boundary will be improved. However, the high computational complexity is the cost to be paid. Although Williams & Shah [WS92] attempts to improve the speed by a greedy algorithm, it's a local minimization method and sub-optimal solution will result. Therefore, Amini *et al.*'s dynamic programming method remains to be the only means for optimal boundary detection. Instead of reducing the computational complexity by sacrificing the solution quality, neural network, which possesses a massive

parallel framework for computation ultimately possibly in real time, is an attractive alternative. In this chapter, we will describe how the snake formulation can be mapped to a generalized Hopfield network model and then correspondingly to a tunneling network so that the nearly optimal solution can be obtained. Then, results of applying neural network implemented snake to detect object boundary in both synthesized and real images are shown and compared with a modified greedy algorithm in Section 6.3.

6.2 From Problem formulation to Neural Network formulation:

6.2.1 Problem Formulation

For the problem formulation, we use Lai-Chin "snake" formulation [LC93]. The internal energy term is corresponding to constraints for equal spacing between snaxels and smoothness along the boundary. This defines the characteristics of the deformable model. The external energy term is to push the snaxels to latch on edges and corners. Mathematically, the formulation is:

Internal energy term at i^{th} snaxel:

$$E_{int}(v_i) = E_{eq}(v_i) + E_{smooth}(v_i) \qquad ..(6.1)$$

Energy term corresponding to the equal spacing constraint:

$$E_{eq}(v_i) = \left(\frac{d_\infty - \|v_i - v_{i-1}\|_\infty}{d_\infty} \right)^2 \qquad ...(6.1a)$$

Energy term corresponding to the smoothness constraint:

$$E_{smooth}(v_i) = \frac{1}{4} \left| \frac{v_{i-1} - v_i}{|v_{i-1} - v_i|} - \frac{v_i - v_{i+1}}{|v_i - v_{i+1}|} \right|^2 \qquad ...(6.1b)$$

External energy term i^{th} snaxel:

$$E_{ext}(v_i) = 1 - |\nabla I(v_i)| \left| \nabla I(v_i) \cdot \frac{v_{i-1} - v_{i+1}}{|v_{i-1} - v_{i+1}|} \right| \qquad ...(6.2)$$

The overall energy function is:

$$E_{total}(v,\lambda) = \sum_{i=1}^{n_s} \left(\lambda_{i_1} E_{ext}(v_i) + \lambda_{i_2} E_{eq}(v_i) + \lambda_{i_3} E_{smooth}(v_i) \right) \qquad ...(6.3)$$

where v_i *is the* i^{th} *snaxel coordinates*

d_∞ *is the average of the infinite norm of inter-snaxel spacing*

$\nabla I(v_i)$ *is the gradient vector at* i^{th} *snaxel normalized among the whole image*

λ_i *is the compromising weights of different energy terms at* i^{th} *snaxel and*

$$\sum_{j=1}^{3} \lambda_{i_j} = 1$$

n_s is the number of snaxels.

Lai *et al.* [LC93] proposed to use minimax rule to determine the set of λ so that choosing them wisely without human-interaction can be achieved. Experimental result showed that this rule can achieve the least mean-square-error among the cases with all possible globally constant λ under different noise levels. Incorporating the minimax rule, the overall energy function becomes:

$$E_{total} = \sum_{i=1}^{n_s} \max \left\{ E_{ext}(v_i) \ , \ E_{eq}(v_i) \ , \ E_{smooth}(v_i) \right\} \qquad ...(6.4)$$

6.2.2 Recurrent Neural Network Model used

We represent each snaxel and its neighbours on the image by a layer of neurons of McCullough-Pitts type and there are totally *n* layers, where *n* is the no. of snaxels. If a neuron is activated in a particular layer, the corresponding snaxel will take that location. For the neurons' connection, according to the overall energy function for the "Snake", each term is defined by combination of at most **three** consecutive snaxels. Thus, it's obvious that we have to deal with a higher-order combinatorial optimization problem. To cope with the situation, a second order Hopfield network becomes necessary. However, as mentioned in Chapter 3, the quality of the solution obtained by Hopfield network is very much limited. In order to seek the optimal solution, we use our novel Tunneling network, which is thoroughly described and analysed in Chapter 4 and shown to be able to approach to a solution much better than Hopfield network. Pole shifting version of Tunneling network is used instead of the adaptive

one as for the case of synthesized image, there are a lot of local minimum states having same energy value and the adaptive version can't tunneling through that type of problem.

6.2.3 Neural Network formulation

As mentioned in Chapter 4, Tunneling network uses the neural network formulation of generalized Hopfield network. To derive the neural network formulation, according to Section 3.1, we have to map the problem's cost function to generalized Hopfield energy with some penalty terms for the validity of the converged solution. The "Snake" problem of eqn.(6.4) can be written as a cost function of third order combination of neuron's state value together with a penalty as:

$$E=\sum_{l=1}^{n_s}\sum_{p=1}^{m}\sum_{q=1}^{m}\sum_{r=1}^{m}\max\{E_{eq}(p,q,r),E_{smooth}(p,q,r),E_{ext}(p,q,r)\}\ x(l-1,p)x(l,q)x(l+1,r) - w_{hard}\sum_{l=1}^{n_s}\left(\sum_{p=1}^{m} x(l,p) - 1\right)^2 \quad ..(6.5)$$

where l is the layer index
p, q, r are the index of the neurons at particular layer
m is the size of the neighborhood included for a snaxel
n_s is the number of snaxels
$x(l,p)$ is the state variable of the pth neuron at l^{th} layer.

The first part includes the complete snake formulation into the neural network formulation.[1] They can be interpreted as soft constraints. However, in order that the neural network will work properly, we need some hard constraints to ensure that the network is making a meaningful/valid combinatorial solution. For our problem, it's obvious that one and only one neuron has to be activated for each layer. As normal practice, we impose this hard constraint by penalty method, ie. adding a penalty term to the energy function as the second term of eqn.(6.5). It's noted that the weighting w_{hard} should be sufficiently large to ensure the satisfaction of the hard constraint. According to eqn. (6.5), the neural network formulation can be derived as eqn.(6.6).

---

1 Evaluation of $E_{eq}(p,q,r)$, $E_{smooth}(p,q,r)$ & $E_{ext}(p,q,r)$ are equivalent to that of $E_{eq}(v_i)$, $E_{smooth}(v_i)$ & $E_{ext}(v_i)$ simply by substituting the corresponding coordinates of neuron p, q & r to v_{i-1}, v_i & v_{i+1} respectively and thus they can be evaluated using eqn.(6.1-2).

$$T_{soft}(l-1,p)(l,q)(l+1,r) = -0.5 \max \{ E_{eq}(p,q,r), E_{smooth}(p,q,r), E_{ext}(p,q,r) \}$$

$$T_{hard}(l,p)(l,q) = -2\, w_{hard} \qquad ..(6.6)$$

$$I_{hard}(l,p) = 2\, w_{hard}$$

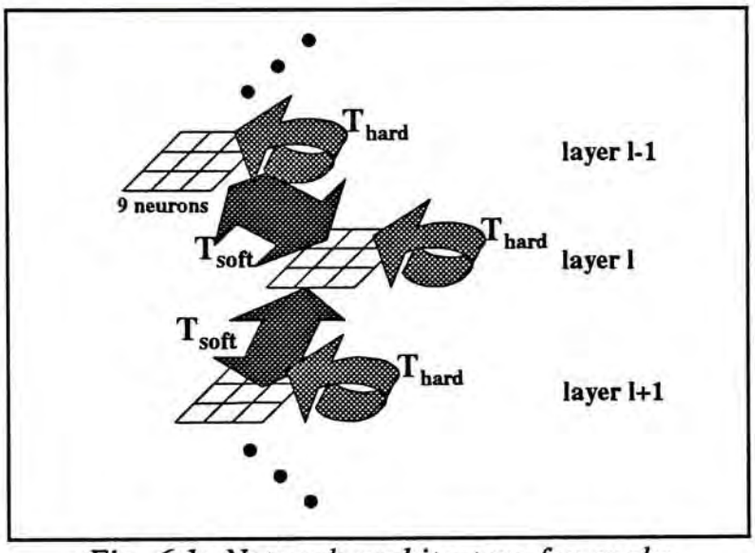

Fig. 6.1: Network architecture for snake application

It's noted that the T_{soft} governs the interlayer connection while T_{hard} governs the intralayer connection and the network architecture can be shown as Fig. 6.1. Based on eqn. (6.6), the Tunneling network can be defined accordingly.

Besides, it's to point out that the penalty term will become zero when the hard constraint is satisfied. So, the energy value of eqn.(6.5) will be the same as that of eqn.(6.4) at all the valid solutions, which is in fact the local minimum state of eqn.(6.5). Therefore, the local minima set of eqn. (6.4), denoted by L_n will be subset of that of eqn.(6.5), denoted by L_c. Thus, as the tunneling network possesses the ability to escape from L_c, its ability escape from L_n is implied obviously. So, it's expected that the neural network approach for "Snake" can detect a better boundary than the existing local minimization techniques' implementation.

6.3 Simulation Results and Discussions:

6.3.1 Feasibility study and Performance comparison

Based on the neural network formulation in Section 6.2.3, we apply Tunneling network implemented snake on synthesized 128x128 "Square" image (whose centre is at (64,64) and each side is of 64 pixel long) to illustrate the feasibility of this scheme. The initial snake is fixed to be a circle. The searching window is fixed to be 3x3. Then, an experimental sensitivity analysis of the proposed scheme towards initial snake position is performed and the result is compared to that of modified greedy algorithm, which is a modified version of Williams and Shah's greedy algorithm [WS92] for convergence assurance by Lai *et al.* [LC93]. For the experiment, we try to vary the initial snake position by i) rotating the initial snake, ii) changing the initial snake radius & iii) shifting the initial snake centre to test the quality of the converged snake. We measure the performance by three parameters, defined for an n-point snake as:

a) **Intersnaxel distance deviation**

$$= \frac{\sum_{i}^{n} \left(\textit{distance between } i^{th} \ \& \ i{+}1^{th} \textit{ snaxels} - \textit{average intersnaxel distance} \right)}{n}$$

b) **Corner deviation**[1]

$$= \frac{\sum_{i}^{n} \min\left(\left| \textit{distance between snaxels and the corner} \right| \right)}{n}$$

c) **No. of snaxels on the expected boundary**[2]

[1] As a square has four corners, we take average among the four corner deviation values.

[2] The boundary for the square is { $x = 33$, $33 \le y \le 96$ } or { $x = 96$, $33 \le y \le 96$ } or { $y = 33$, $33 \le x \le 96$ } or { $y = 96$, $33 \le x \le 96$ }

E.1) Experiment One:

Image for boundary detection: Synthesized "Square" image

Initial snake setting:
- Number of snaxels = 16
- Circular shape
- Centre of the snake = (64,64)
- Radius of the snake = 35

Initial snake variation: Rotating the initial snake with different angle

Result: The experiment result is presented in Table 6.1. It's observed that both Modified Greedy Algorithm (MGA) and Tunneling network (TN) have similar performance in boundary detection although the latter one is better in terms of intersnaxel distance deviation all the time.

E.2) Experiment Two:

Image for boundary detection: Synthesized "Square" image

Initial snake setting:
- Number of snaxels = 16
- Circular shape
- Centre of the snake = (64,64)
- Starting angle being fixed

Initial snake variation: Varying the initial snake's radius

Result: The experimental result is presented in Table 6.2, It's observed that the sensitivity of TN towards initial snake's radius is much lower than that of (MGA). For instance, when radius of the initial snake is 30, snake by MGA totally break down and can only have 2 snaxels finding the expected boundary while snake by TN can have all 16 snaxels correctly latching on it. The situation is shown in Fig. 6.3a-c.

E.3) Experiment Three:

Image for boundary detection: Synthesized "Square" image

Initial snake setting:
- Number of snaxels = 16
- Circular shape
- Radius of the snake = 35
- Starting angle being fixed

Initial snake variation:	The centre of the snake is shifted to the right from (64,64) to different extent

Result: From Table 6.3, we found that the sensitivity of TN towards initial snake shifting is also much lower than that of MGA. For example, when the initial snake centre is at (74,64), snake by MGA only have 5 snaxels on the expected boundary while snake by TN can have 15 snaxels correctly latching on it. Similar situation is shown in Fig. 6.2a-f.

6.3.2 Smoothing and Boundary Detection

In order to further improve the quality of the detected boundary, scale-space smoothing scheme can be introduced to the image from coarse to fine. At each scale, let the snake to converge and the converged snake is used as the initial snake for the next finer scale smoothed image. There are mainly two advantages.

i) The quality of the detected boundary becomes less sensitive towards the initial snake placement. Even if global optimization scheme is used, initialization of the snake is still a problem if the searching window for each snaxel is not large enough. For small window size, edges far from the snaxels can't be located. According to Fig. 6.5b, both MGA & TN failed in detecting the square's boundary as the snaxels of the initial snake are too far from the edges and local information gives no idea where to find the boundary. If the above smoothing scheme is applied as preprocessing step, the edge information will be dispersed out and the snake initialization sensitivity can be significantly improved. The result is clearly revealed in Fig. 6.5c-f. Although another obvious solution is to increase the searching window size, this method leads to tremendous increase in number of connection weights and is not a cost-effective mean.

ii) When applying snake on real image, even if the initial snake is placed near enough to the desired object, small local features due to the background and some other objects nearby can block the snake from locating the desired object boundary accurately. By coarse to fine scale smoothing, the undesirable local features can be escaped at coarse scale so that the snake can move near enough to the desired object boundary and at fine scale, the desirable local features on the object boundary can then be located accurately. Fig. 6.6 illustrates detection of the

girl's head in "Lenna" image with and without the scale-space smoothing scheme. It's obviously noted that at coarse scale, the snake can get the rough shape and the snake jiggles as the scale decreases to locate the boundary of the girl's head accurately.

6.3.3 Convergence improvement by network decomposition

In Section 4.4.2.2, it's shown that small-size decomposition of tunneling network can speed up a tunneling network such that it can yield a better solution within a fixed number of iterations. In this application, same strategy is introduced. However, we use fixed partitioning for consecutive layers of neurons instead of random partitioning as the neurons are only connected to that in the consecutive layers for this application. Coherent result is found and convergence towards global minimum state is faster for decomposition scheme using small-size partitioning. The result is obviously illustrated in Fig. 6.7.

According to the problem formulation, neurons with 2 layers between them will be independent. We can take advantage of the independence such that parallelism can be incorporated together with decomposition and all the independent partitions can be updated at a time. The idea is clearly shown in Fig. 6.8

6.3.4 Hardware implementation consideration

After proofing the feasibility, studying the effectiveness and efficiency of the algorithm, the hardware implementation has to be considered. Even if we can successfully build the tunneling network architecture, there still remains two problems required further investigation.

Firstly, the external energy term for the snake problem formulation involves complicated calculation between the input data/image and the possible snake positions. This implies that network connection has to be modified for different images and the possibility of implementing an image independent hard-wired chip for boundary detection can only be achieved by introducing dedicated hardware for the calculation.

Secondly, the huge number of network connection is another limitation. Number of connection needed can be found to be $m^3 x n x 6$ ($O(nm^3)$).

Window size, m	Connection No.
3x3	131.22k
5x5	2.18M
7x7	21.18M

This great amount of connection requirement is inherited from the higher-order architecture and hinders the possibility of building the hardware on a single chip. For more detailed discussion, interested readers can refer to Section 2.4.3 in Chapter 2.

6.4 Conclusions

The boundary detection problem using "Snake" has been successfully mapped to a generalized higher-order Hopfield network and then to a Tunneling network, a recurrent neural network with a massively parallel platform. Performance comparison through a sensitivity analysis between the existing local minimization technique, Modified Greedy Algorithm and the neural network alternative proved that the neural network technique can yield a better solution due to its capability in seeking global minimum state and is much less sensitive towards initial snake position. Furthermore, the converging rate of the neural network is found to be increased by using network decomposition technique described in Section 4.4.2.2 and the performance of the boundary detection scheme can be further enhanced by using scale-space filtering as preprocessing step.

Rotated Angle (in radian)	MGA			TN		
	δ_d	δ_c	n	δ_d	δ_c	n
0.00	7.25	0.00	16	3.18	2.00	16
0.05	7.25	0.00	16	2.88	2.00	16
0.10	7.25	0.00	16	2.81	1.75	16
0.15	4.47	2.00	16	3.94	3.30	16
0.20	5.17	4.50	16	3.67	3.50	16
0.25	5.19	4.75	16	3.50	3.00	16
0.30	5.00	3.00	16	3.38	2.00	16
0.35	5.33	1.25	16	0.375	0.00	16
0.40	5.92	0.50	16	0.375	0.00	16
0.45	7.25	0.00	16	2.75	1.00	16

Table 6.1: *Sensitivity analysis towards detecting boundary of "Square" image by rotating circular initial snake (Centre at (64,64); radius = 35). MGA stands for Modified Greedy Algorithm and TN stands for Tunneling network.*

i) δ_d is the intersnaxel distance deviation,
ii) δ_c is the average corner deviation,
iii) n is the number of snaxels on the expected edge.

Radius of Initial Snake	MGA			TN		
	δ_d	δ_c	n	δ_d	δ_c	n
35	7.25	0.00	16	2.88	2.0	16
34	5.97	0.50	16	1.78	0.75	16
33	5.91	4.25	11	0.94	0.0	16
32	6.38	4.50	11	1.78	0.75	16
31	0.75	10.50	3	1.78	0.75	16
30	0.84	10.50	2	3.41	2.00	16
29	1.25	11.50	0	1.25	11.50	0

Table 6.2: Sensitivity analysis towards detecting boundary of "Square" image by varying the radius of circular initial snake (Centre at (64,64); Fixed starting angle). MGA stands for Modified Greedy Algorithm and TN stands for Tunneling network.
i) δ_d is the intersnaxel distance deviation,
ii) δ_c is the average corner deviation,
iii) n is the number of snaxels on the expected edge.

Shifted pixels of Initial Snake	MGA			TN		
	δ_d	δ_c	n	δ_d	δ_c	n
2	7.25	0.00	16	2.88	1.50	16
4	6.38	2.00	15	2.60	1.75	16
6	5.813	7.25	10	4.38	3.50	15
8	6.86	6.50	9	4.06	2.00	16
10	3.56	9.50	5	2.88	1.50	15

Table 6.3: Sensitivity analysis towards detecting boundary of "Square" image by shifting the initial snake to the right (Radius = 35; Fixed starting angle). MGA stands for Modified Greedy Algorithm and TN stands for Tunneling network.
i) δ_d is the intersnaxel distance deviation,
ii) δ_c is the average corner deviation,
iii) n is the number of snaxels on the expected edge.

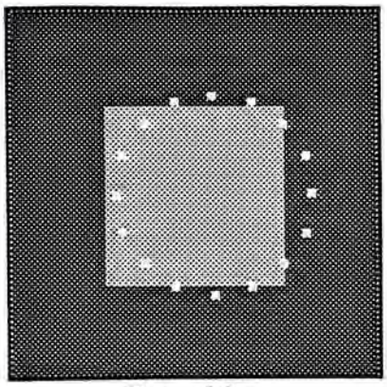

Fig. 6.2a

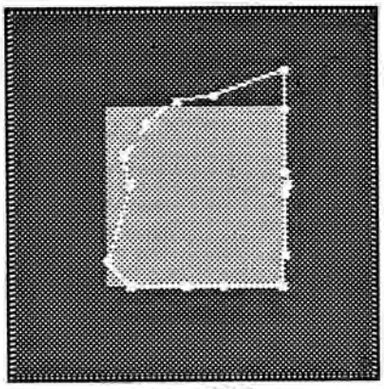

Fig. 6.2b

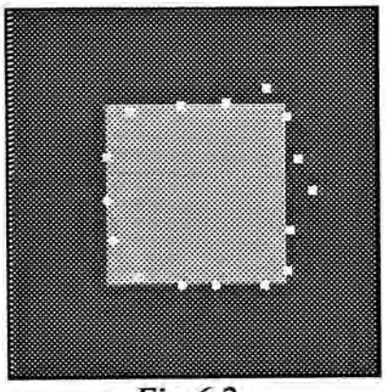

Fig 6.2c

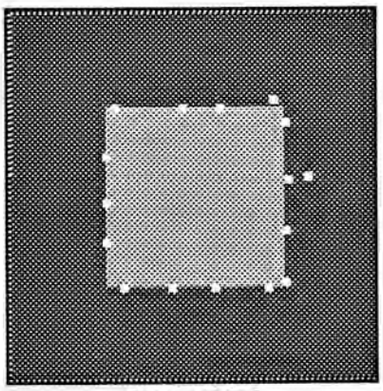

Fig. 6.2d

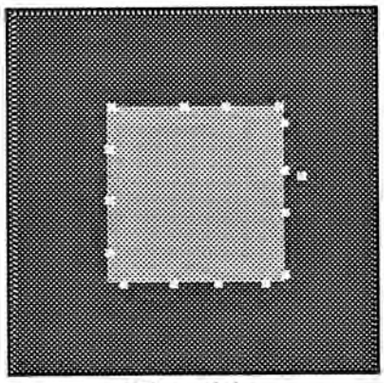

Fig. 6.2e

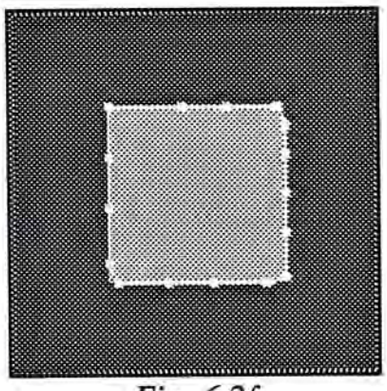

Fig. 6.2f

Fig. 6.2: *Comparison of modified greedy algorithm and tunneling network implemented snake on "Square" image with the centre of the initial circular snake shifted to (71,64). a) Initial snake position; b) Converged Snake by modified greedy algorithm; c-e) Snake position after 10, 20, 30, 40 tunneling iteration; f) Converged Snake by tunneling network.*

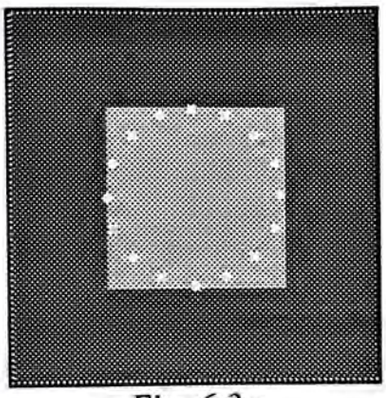

Fig. 6.3a

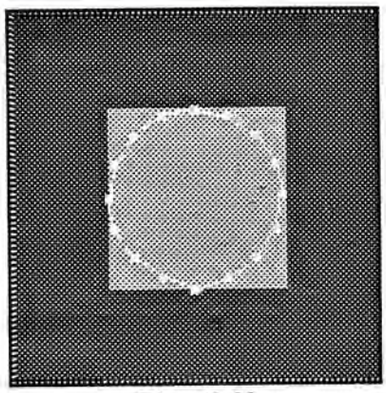

Fig. 6.3b

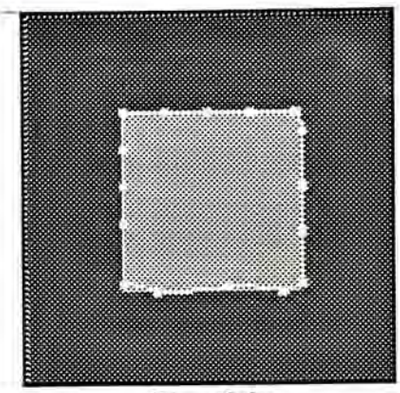

Fig. 6.3c

Fig. 6.3: *Comparison of modified greedy algorithm and tunneling network implemented snake on "Square" image with the radius of the inital circular snake equal 30. a) Initial snake position; b) Converged Snake by modified greedy algorithm; c) Converged Snake by tunneling network.*

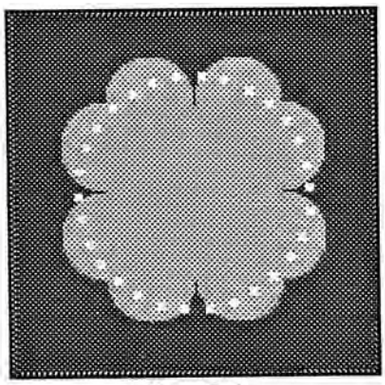

Fig. 6.4a

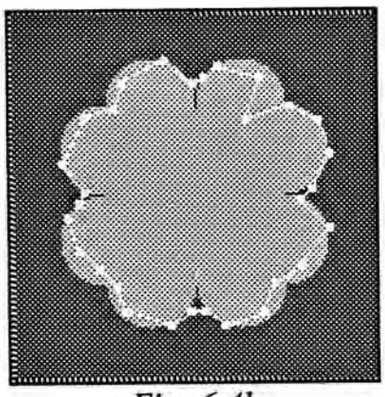

Fig. 6.4b

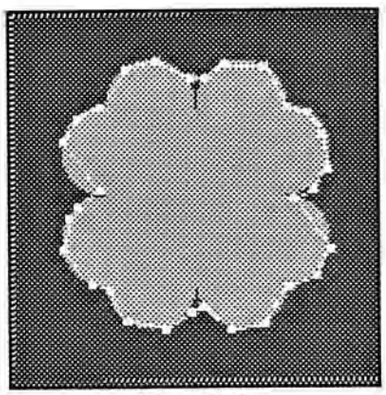

Fig. 6.4c

Fig. 6.4: *Comparison of modified greedy algorithm and tunneling network implemented snake on "Flower" image. a) Initial snake position; b) Converged snake by modified greedy algorithm; c) Converged Snake by tunneling network.*

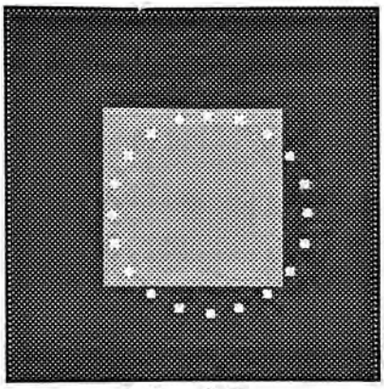

Fig. 6.5a

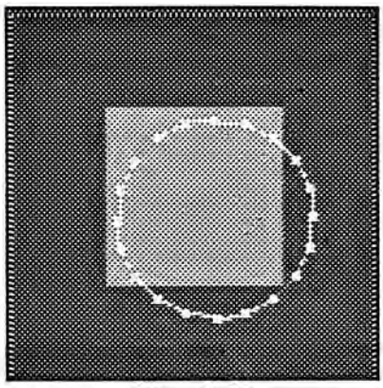

Fig. 6.5b

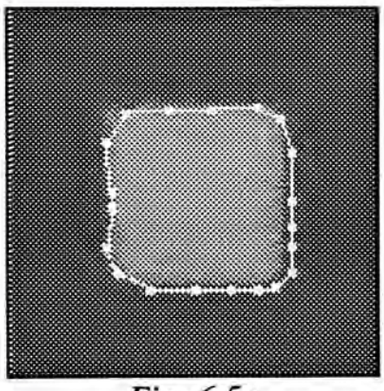

Fig. 6.5c

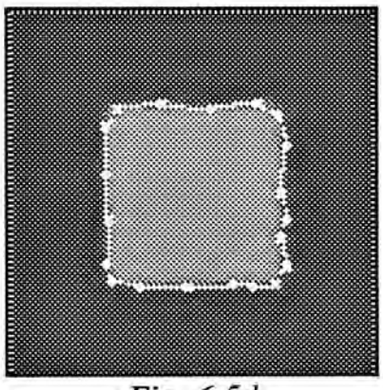

Fig. 6.5d

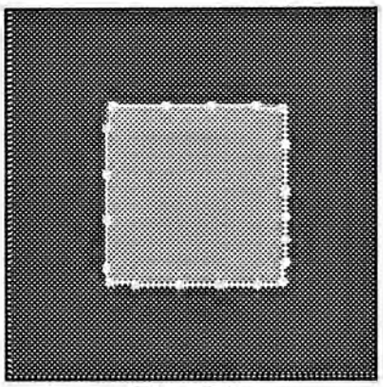

Fig. 6.5e

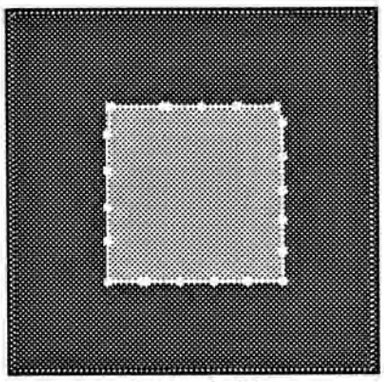

Fig. 6.5f

Fig. 6.5: *Figures to illustrate how coarse to fine scale-space filtering can be used to further relax the sensitivity of initial snake position. a) Initial snake position on "Square" image; b) Converged snake without applying any smoothing for both MGA & TN; c-f) Converged snake for coarse to fine scale smoothed image. The converged snake at a particular scale is used as the initial snake position for the next finer scale. It's noted that the snake is brought to the right position at coarse scale and latches on the edge accurately at fine scale.*

Fig. 6.6a

Fig. 6.6b

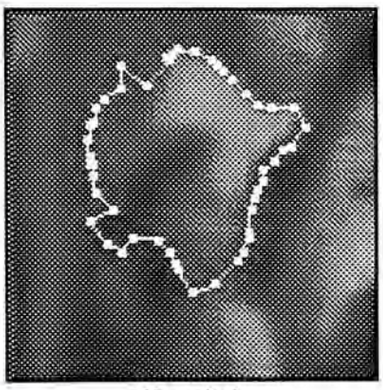

Fig. 6.6c

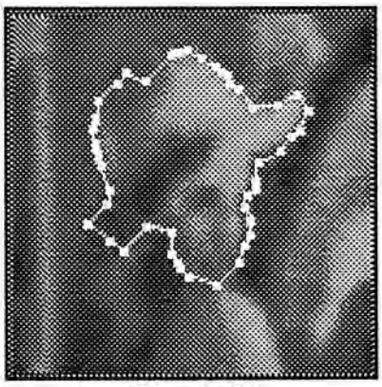

Fig. 6.6d

Fig. 6.6e

Fig. 6.6f

Fig. 6.6: *Figures to illustrate how coarse to fine scale-space filtering can be used to detect boundary of real image. TN is used for the minimization. a) Initial snake position on "Lenna" image; b) Converged snake without applying any smoothing; c-f) Converged snake for coarse to fine scale smoothed image. The converged snake at a particular scale is used as the initial snake position for the next finer scale. It's noted that the snake captures the outline of the girl's head at coarse scale and latches on the local features accurately at fine scale.*

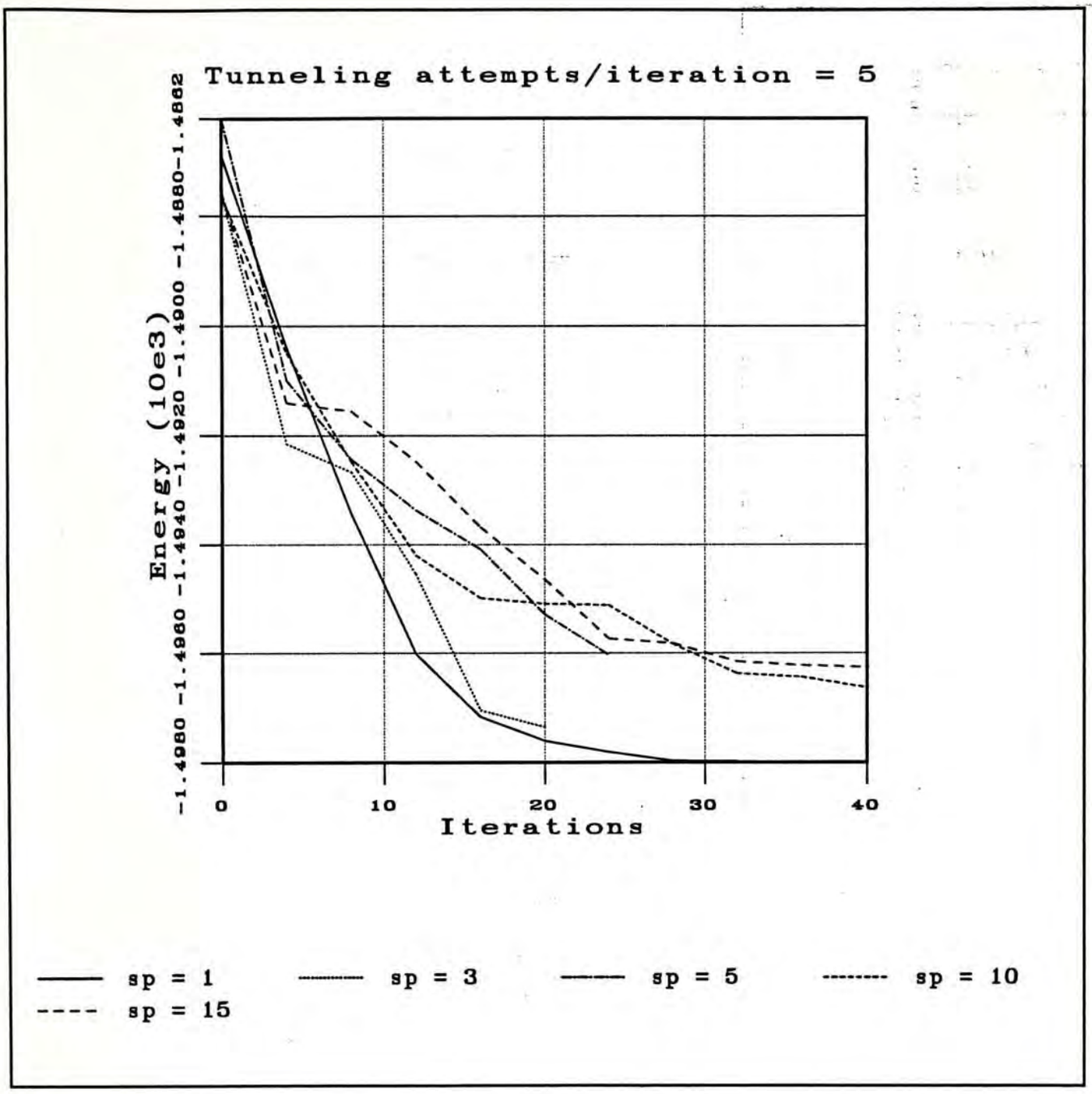

Fig. 6.7: *Converaging rate comparison by different size network decomposition. The problem consists of 15 snaxels to detect "Square" image's boundary and the variable, "sp" used in the legend means the number of consecutive layers grouped at a time for updating. For this simulation, 5 tunneling attempt per iteration is adopted, ie. all the neurons within the current updating group are checked and updated accordingly for five times during each iteration. It's noted that the cases, with smaller number of neurons grouped at a time, converge faster and are able to obtain a state with lower energy value.*

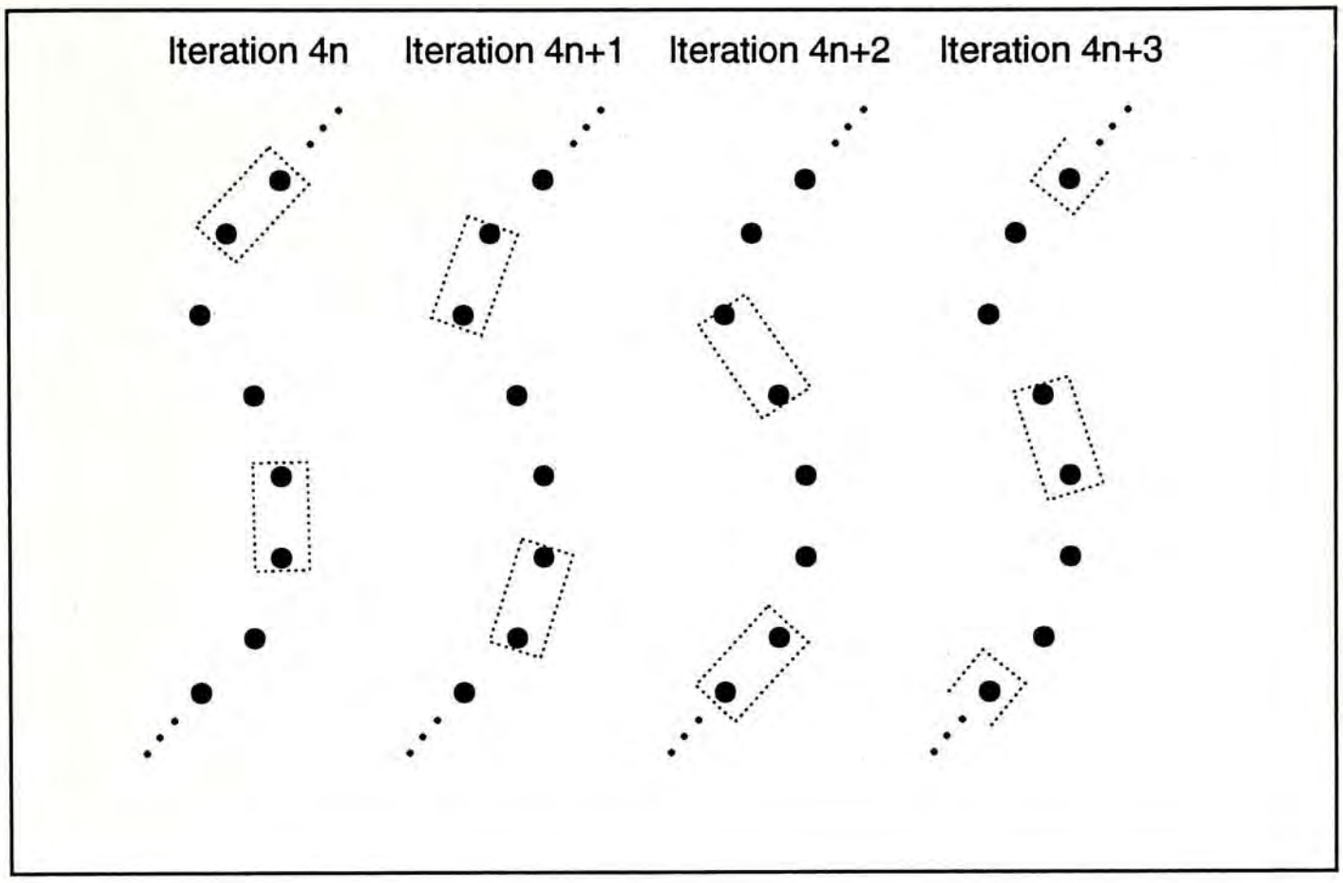

Fig. 6.8: *Figure to illustrate how parallelism can be incorporated into the decomposed recurrent neural network for snake so that all the neurons can be updated at least once within 4 iterations. The neurons within the boxes with dotted line are meant to be updated. Boxes are separated by two neurons as neurons with that separation are independent. Simultaneously updating them won't give rise to conflicts in determining the neurons' state.*

Chapter 7 Conclusions and Future Researches

Through our research, different limitations arisen within the procedure of utilizing generalized Hopfield network for optimization are addressed. Suggested solutions are then provided accordingly, including a novel neural network model for global optimization. Finally, we succeed in providing neural network solutions for two computer vision problems, gaussian filtering and boundary detection. In this chapter, we will first summarize our contributions and draw some conclusions based on them, then, limitations of our work and suggested future researches are followed.

7.1 Contributions and Conclusions

Discussions on the procedure of utilizing generalized Hopfield network model for optimization are provided from the very beginning of problem formulation to the end of network stability assurance. They have been verified to be useful during our derivation of the recurrent neural networks for the applications, gaussian filtering and boundary detection. Although we can't prove that this is a universal procedure as there are so many different problems in the world, it can be used as an *application reference* for some domain expertises who are new in this field but may be interested in neural network problem-solving paradigm.

Within the procedure, one of the important steps is to assure the formulated network's convergence. From a geometric perspective, we have shown that all generalized Hopfield networks with their formulations having non-zero self-reinforcement terms will not be stable. In view of the fact that this condition frequently happened in practice, a "*reshaping strategy*" is thus proposed to reformulate (reshape) the neural network formulation such that the "reshaped" network can always converge. So, by this proposed strategy, whenever there is a problem mappable to generalized Hopfield network, a corresponding neural network formulation with convergence guaranteed can always be found. This is a simple yet important step when generalized Hopfield network is considered for combinatorial optimization.

For contribution in improving the quality of the solution obtained, a novel neural network model - *Tunneling network* is proposed and derived from the marriage of generalized Hopfield network and a global optimization technique called tunneling algorithm. The tunneling process is achieved by a pole shifting scheme. The stability and global convergence property of the newly proposed network are both investigated. It's proved to converge to a local minimum state for a fixed pole and to the global minimum state in infinite time during the whole tunneling process. This provides the concrete backup of the *global optimization capability* possessed by Tunneling network.

Through modifying the pole expression of the Tunneling network associated energy function, different versions of Tunneling network can be derived. In this thesis, by considering hardware implementation simplicity, two variations are proposed. Besides the pole shifting version, an adaptive version is also posed. To solve TSP problem, simulation results reveal that adaptive version is a much better one in terms of the quality of solutions obtained. However, under the situation that the current local minimum state surrounded by some other local minimum states with same energy value, the adaptive Tunneling network can't tunnel through nearby barrier and will be stuck at the state forever. Thus, this version will not always guarantee convergence to the global minimum. In such a situation, the pole shifting version becomes the only choice as it doesn't have this defect. Therefore, modifying the Tunneling network's pole is a way to *adjust* its tunneling power and generality of its problem-solving capability.

In considering average performance of Tunneling network in terms of the quality of solutions obtained and the convergence rate, both versions, through simulations, are found to overwhelm simulated annealing, a popular global optimization technique very often accompanying with Hopfield network when an optimal result is desired. For further enhancement, a network decomposition strategy is proposed by grouping certain number of neurons at a time such that the solution space is confined according for more thorough optimum searching by tunneling. Simulation result manifested that small-size grouping of neurons with random partitioning scheme can converge to a better solution. All these

simulation results prove that Tunneling network is a very *promising* means for global optimization.

As revealed in Section 1.3, computer vision problems can be categorised mainly into convex quadratic programming and combinatorial programming. To evaluate our neural network methodology in computer vision domain, gaussian filtering and boundary detection are chosen from the two categories respectively. They are successfully mapped to neural network formulation and solved by using different recurrent neural networks. In the gaussian filtering application, the impulse response of the derived recurrent neural network is found to be very much like a gaussian-shape. Deriving this recurrent neural network is rather straight forward but the most tedious but important part is the boundary connection assignment. Mis-assignment may lead to error arisen at the boundary. The error will propagate to the interior part and the performance of the network will be greatly deteriorated. In fact, similar problem was noted by Zhou *et al.* [ZCVJ88] during their establishment of recurrent neural network for image restoration. Ringing effect is resulted due to the boundary error. For this particular application, we derived some formula to cope with the situation. Thus, the *boundary connection assignment* can be concluded as the step requiring the most attention in deriving recurrent neural network for image processing algorithms which requires boundary value consideration.

In the boundary detection application, we adopted active contour model - snake as our problem formulation and our novel Tunneling network is chosen as the recurrent neural network model. Due to its optimal boundary seeking capability, the derived network is found to be able to detect a much better object boundary than some existing implementations of active contour model. Moreover, its performance is much less sensitive to initial conditions. Thus, through this application, the tunneling network has once again demonstrate its value in seeking optimal solution for practical problems of combinatorial type. Moreover, as it's a blind-searching method with no a prior knowledge on the problems' solution being required, it can be easily plugged into other applications. This reveals the fact that our Tunneling network is in fact a very *generic* solution for a large group of combinatorial problems.

7.2 Limitations and Suggested Future Researches

i) Necessity of general-purpose neurocomputer

Despite the impressive results obtained by Tunneling network for both benchmark and practical problems, lengthy computation time for simulation will disappoint ones who intend to use it. As all the neural network algorithms, including our Tunneling network, are designed based on a parallel distributed structure, to simulate its dynamic using digital sequential computer will definitely take a very long time. Also, due to the requirement of huge amount of memory for the storage of the connections, substantial memory swapping will further lower the computational speed. To testify whether a neural network algorithm can fulfil its duty in real-time and furthermore, to speed up the development, instead of building dedicated hardware, a general-purpose neurocomputer with architecture specially designed for neural network algorithms will be of great help inevitably. Although, in the market, there are some models available, their capability is very limited. More fundamentally, in fact, the specification of a neurocomputer eligible for general development is not yet well discussed by the neurocomputing expertises. So, researches in development of a general-purpose neurocomputer will be of great value in triggering further development of neurocomputing paradigms.

ii) Further work on Tunneling network

As mentioned in Section 7.1, modifying the pole expression of Tunneling network associated tunneling function can affect its tunneling power and generality in problem-solving capability. In our researches, we only provide two possible variations. Not much effort is specially put to optimize the pole expression for network performance. So, it's believed that further adjustment or using different functions for the expression may come out with an even better version.

For evaluation, only simulated annealing is compared and found to be inferior to

proposed Tunneling network. Recently, mean field annealing (MFA) and genetic algorithm (GA) are found to be two very promising methods for blind-searching. For MFA, it's in fact a speedup version of SA. Using MFA involves the cooling scheduling determination and critical temperature estimation which complicates its comparison with other algorithms. Preliminary simulation is performed using a slow enough schedule and starting the annealing at a high enough temperature and the solutions so obtained are also inferior to the ones obtained by Tunneling network (not presented in this thesis). For genetic algorithm, it possesses a very attractive property - learning during searching, which is not possessed by our Tunneling network. However, so far it remains to be an algorithm based on programmed computing paradigm instead of neurocomputing's. To correctly rank them according to their performance, a more thorough performance comparison between MFA, GA and our Tunneling network can be a topic for further investigation. However, it is not necessary to put them into competitive relationship. There exist some hybrid models of GA, eg. GA + gradient descent, which can perform better than the individuals. So, integrating GA and Tunneling network may be able to establish a hybrid-model of Tunneling network with learning capability during the tunneling phase because during the study of our Tunneling network, it's observed that lots of local minimum states are revisited a number of times. If it's possible to direct the tunneling trajectory to the promising region using the knowledge learnt during the previous tunneling phases to avoid revisits, the time for locating the global minimum state can be reduced and the performance will undoubtedly be improved.

iii) <u>Parameter-free recurrent neural network model</u>

Other than further enhancement of Tunneling network, a general parameter-free recurrent neural network model for problem-solving is another possible research direction. A typical problem formulated as mathematical programming model may consist of soft constraints together with hard constraints, like our problem formulation of boundary detection application. Different sets of weighting for the soft constraint terms and the hard constraint terms will affect the quality of the solutions obtained. So far, the researchers assign the weighting's values experimentally. This parameter tuning process is known to be a non-

desirable one. In fact, there exist algorithms to determine the optimal set of weighting. For the soft constraints, min-max rule [GY88] can be used to flee the parameter tuning process for the optimal set of weighting. For hard constraints, Lagrange multiplier method can be used to calculate the required hard constraints' weighting. In fact, some research works have been started to implement a Lagrange Programming neural network [ZC92a] and Minimax neural network [CU92]. If the optimal set of weighting for both soft and hard constraints can be determined by the network dynamics itself, we then can have a really parameter-free recurrent neural network model, possessing the advantage of fully autonomy in problem-solving.

iv) Further research direction on applying Neural Network to Computer Vision

Including our work, most of the research works on applying neural network to Computer Vision are implementing different low-level or intermediate-level visions. Zhou *et al.* in the book *Artificial neural network for computer vision* pointed out that a very challenging research direction in the future is to establish a unified neural network model which can utilize different cues, including edges, texture, stereo, motion information obtained for recognition. In fact, not only in the neural network field, such a conceptual unified model is not much investigated in the computer vision field. Recently, Manjunath and Chellapa [MC93] start working in this direction and a conceptual unified approach to boundary perception is studied.

To realize such a model using neural network, the already established recurrent neural networks for different low-level visions, including our gaussian filtering and boundary detection ones, can be considered as building blocks of a feature extraction layer of the model. Then, the converged state of those blocks can provide cues to another layer, which can be responsible for the interaction between the different cues in order to merge the information extracted from different perspectives, (For example, edges from texture and edges from intensity.) and transform them to some more meaningful representation. Finally, at the toppest layer, some neural networks with classification capability, eg. Backpropagation network, can be used to make the final decision of recognizing an object in the scene.

Although development of such a neural network system is really worth taking, it requires a very substantial system support, including speedy processing system and huge amount of memory for storage all the connections of such a large network, and this may hinder the emergence of the system in the coming few years.

References:

[Als89] J. Alspector, "Neural-Style Microsystems that learn," *IEEE Communication Magazine*, Vol. 27, No. 11, Nov. 1989, pp. 29-36

[Ama72] S. Amari, "Learning patterns and pattern sequences by self-organizing nets of threshol elements," *IEEE Trans. on Computer*, Vol. C-21, Nov. 1972, pp. 1197-1206

[AKH92] S. Abe, J. Kawakami and K. Hirasawa, "Solving Inequality Constrained Combinatorial Optimization Problems by the Hopfield Neural Networks," *Neural Networks*, Vol. 5, 1992, pp. 663-670

[ANF90] S.V.B.Aiyer, M.Niranjan and Frank Fallside, "A Theoretical Investigation into the Performance of the Hopfield Model," *IEEE Trans. on Neural Network*, Vol. 1, No. 2, June 1990, pp. 204-215

[ANF90] S.V.B.Aiyer, M.Niranjan and Frank Fallside, "On the optimization properties of the Hopfield model," *IEEE INNC 90*, Vol. 1, pp. 245-248

[AWJ90] A.A. Amini, T.E. Weymouth and R.C. Jain, "Using Dynamic Programming for Solving Variational Problems in Vision," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 9, Sept 1990, pp. 855-867

[BG88] J. Bruck and J.W. Goodman, "A Generalized Convergence Theorem for Neural Network," *IEEE Trans. on Information Theory*, Vol. 34, No. 5, Sept. 1988

[BM90] David E. Van Den Bout and Thomas K. Miller, "Graph Partitioning Using Annealed Neural Networks," *IEEE Trans. on Neural Network*, Vol. 1, No. 2, June 1990, pp. 192-203

[BMMS89] G.Bilbo, R.Mann, T.K.Miller, W.E.Snyder, D.E.Van den Bout and M.White, "Optimization By Mean Field Annealing," in *Advances in Neural Information Processing System 1*, CA:Morgan Kaufmanu, 1989, pp. 91-98

[BT90] L.Bedini and A. Tonazzini, "Neural network use in Maximum Entropy image restoration," *Image and Vision Computing*, Vol.8, No.2, May 1990, pp. 108-114

[BT92] L. Bedini and A. Tonazzini, "Image restoration preserving discontinuities: the Bayesian approach and neural networks," *Image and Vision Computing*, Vol. 10, No. 2, 1992, pp. 108-118

[BWBD86] J. Babaud, A.P. Witkin, M. Baudin and R.O. Duda, "Uniqueness of the Gaussian Kernel for Scale-Space Filtering," *IEEE Trans. on Patten Analysis and Machine Intelligence*, Vol. 8, No. 1, Jan 1986, pp. 26-33

[CMS91] Chinchuan Chiu, Chia-Yiu Maa and Michael A. Shanblatt, "Energy Function Analysis of Dynamic Programming Neural Networks," *IEEE Trans. on Neural Network*, Vol. 2, No. 4, July 1991, pp. 418-426

[CL92] K.W.Cheung and T.Lee, "Neural Network for Global Optimization," *RNNS/IEEE Syposium on Neuroinformatics and Neurocomputers*, Vol. I, Oct 1992, pp. 53-63

[CS92] J. Chen and M.A. Shanblatt, "Improved Neural Networks for Linear and Nonlinear Programming," *International Journal of Neural Systems*, Vol. 2, No. 4, pp. 331-339, 1992

[CU92] A. Cichocki and R. Unbehauen, " Neural Networks for Solving Systems of Linear Equations - Part II: Minimax and Least Absolute Value Problems," *IEEE Trans. on Circuits and Systems*, Vol. 39, No. 9, Sept 1992, pp. 619-633

[FMM92] M. Forti, S. Maretti and M. Marini, "A Condition for global convergence of a class of symmetric Neural Circuit," *IEEE Trans. on Circuits and Systems*, Vol. 39, No. 6, June 1992

[GG84] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 6, No. 6, Nov. 1984

[GPP91] A. Ghosh, N.R. Pal, and S.K. Pal, "Image segmentation using a neural network," *Biological Cybernetics*, 66, 1991, pp. 151-158

[GY88] M.A. Gennert and A.L. Yuille, "Determining the Optimal Weights in Multiple Objective Function Optimization," *Proceedings of Second International Conference on Computer Vision*, 1988, pp. 87-89

[Hop82] J.J.Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci. USA*, Vol. 79, April 1982, pp. 2554-2558

[Hop84] J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci. USA*, May 1984, pp. 3088-92

[Hua92] C.L. Huang, "Parallel image segmentation using modified Hopfield model," *Pattern Recognition Letters*, 13, 1992, pp. 345-353

[Hul91] M.M.V. Hulle, "A Goal Programming Network for Mixed Integer Linear Programming: A Case Study for the Job-Shop Scheduling Problem," *International Journal of Neural Systems*, Vol. 2, No. 3, 1991, pp. 201-209

[HKP91] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Redwood City, California:Addison-Wesley, 1991

[HSA84] G.F. Hinton, T.J. Sejowski and D.H. Ackley, "Boltzmann machine: Constraint satisfaction networks that learns," *Carnegie Mellon University Technical Report # CMV-CS-84-119, Carnegie Mellon University*, Pitts-burge PA, May, 1984

[HT85] J.J.Hopfield and D.W.Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics*, 52, 1985, pp. 141-152

[Ios80] Marius Iosifescu, *Finite Markov Processes and Their Applications*, Romania:John Wiley & Sons, 1980

[IM91] D. Ingman and Y. Merlis, "Local Minimum Escape Using Thermodynamic Properties of Neural Networks," *Neural Networks*, Vol. 4, 1991, pp. 395-404

[Jai89] Anil K. Jain, *Fundamentals of Digital Image Processing*, New Jersy:Prentice-Hall, 1989

[JB91] Jayadeva and B. Bhaumik, "Comments on "A Naive Approach to Global Optimization," *IEEE Trans. on Circuits and Systems*, Vol. 38, No. 7, July 1991, pp. 818-820

[Kli92] C. Klimasauskas, "Neural Networks: An Engineering Perspective," *IEEE Communication Magazine*, Vol. 30, No. 9, Sept 1992, pp. 50-53

[Kob91] Y. Kobuch, "State Evaluation Functions and Lyapunov Functions for Neural Networks," *Neural Networks*, Vol. 4, 1991, pp. 505-510

[KC88] M.P. Kennedy and L.O. Chua, "Neural Networks for Nonlinear Programming," *IEEE Trans. on Circuits and Systems*, Vol. 35, No. 5, May 1988, pp. 554-562

[KGV83] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol 220, No. 4598, May 1983, pp. 671-680

[KLKH91] K.H. Kim, C.H. Lee, B.Y. Kim and H.Y. Hwang, "Neural optimization network for minimum-via layer assignment," *Neurocomputing*, 3, 1991, pp. 15-27

[KWA91] H. Kobayashi, J.L. White and A.A. Abidi, "An Active Resistor Network for Gaussian Filtering of Images", *IEEE Journal of Solid-state Circuits*, Vol. 26, No. 5, pp.738-748, May 1991

[KWT88] M. Kass, A. Witkin and D. Terzopoulos, "Snakes: Active Contour Models," *First International Conference on Computer Vision*, 1987, pp. 259-269

[LC93] K.F.Lai and R.Chin, "On Regularization, Formulation and Initialization of the Active Contour Models (Snakes)", submitted *IEEE Trans. on Pattern Analysis and Machine Intelligence* for publication

[LLTL91] W.C. Lin, F.Y. Liao, C.K. Tsao and T. Lingutla, " A Hierarchical Multiple-View Approach to Three-Dimensional Object Recognition", *IEEE Trans. on Neural Networks*, Vol. 2, No. 1, pp.84-92, January 1991

[LM85] A.V. Levy and A. Montalvo, "The tunneling algorithm for the global minimization of functions," *SIAM. J. Sci. Stat. Comput.*, Vol. 6, No. 1, January 1985, pp. 15-29

[LMP88] J.H. Li, A.N. Michel and W. Porod, "Qualitative Analysis and Synthesis of a Class of Neural Network," *IEEE Trans. on Circuits and Systems*, Vol. 35, No. 8, Aug. 1988

[LS91a] B.W. Lee and B.J. Sheu, "Hardware Annealing in Electronic Neural Networks," *IEEE Trans. on Circuits and Systems*, Vol. 38, No. 1, Jan 1991, pp. 134-137

[LS91b] B.W. Lee and B.J. Sheu, "Modified Hopfield Neural Networks for Retrieving the Optimal Solution," *IEEE Trans. on Neural Networks*, Vol. 2, No. 1, January 1991, pp. 137-142

[LSCC92] J. Lee, B.J. Sheu, J. Choi, and R. Chellappa, "A Mixed-Signal VLSI Neuroprocrssor for Image Restoration," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 2, No. 3, Sept 1992, pp. 319-324

[LSD90] S.Lu, A. Szeto and J. Dawe, "Edge Detection and Reinforcement using Hierarchical Neural Networks," *Proceedings of 10th ICPR*, 1990, pp. 1031-1035

[Min67] M.L. Minsky, *Computation: Finite and Infinite Machine*, Englewood Cliffs:Prentice-Hall, 1967

[MH80] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. R. Soc. Lond. B*, 207, 1980, pp. 187-217

[MM91] C.A. Mead and M.A. Mahowald, "The Silicon Retina," *Scientific America*, May 1991, pp. 76-82

[MM88] C.A. Mead and M.A. Mahowald, "A Silicon Model of Early Visual Processing", *Neural Networks*, Vol. 1, 1988, pp. 91-97

[MP43] W.S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in neurons activity," *Bulletin of Mathematical Biophysics*, Vol. 5, 1943, pp. 115-133

[Nie90] R.H. Nielsen, *Neurocomputing*, Reading, Massachusetts:Addison-Wesley, 1990

[NC92] N.M. Nasrabadi, C.Y. Choo, "Hopfield Network for Stereo Vision Correspondence," *IEEE Trans. on Neural Networks*, Vol. 3, No. 1, Jan 1992, pp. 5-13

[Pet90] Carsten Peterson, "Parallel Distributed Approaches to Combinatorial Optimization: Benchmark Studies on Traveling Salesman Problem," *Neural Computation*, 2, 1990, pp. 261-269

[PK90] J.K. Paik and A.K. Katsaggelos, "Edge Detection using a Neural Network," *Proceedings of ICASSP'90*, pp. 2145-2148

[PK92] J.K. Paik and A.K. Katsaggelos, "Image Restoration Using a Modified Hopfield Network," *IEEE Trans. on Image Processing*, Vol. 1, No. 1, pp.49-63, January 1992

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization : Algorithm and Complexity*, New Jersey:Prentice-Hall, 1982

[PS89] C. Peterson and B. Soderberg, "A new method for mapping optimization problems onto neural networks," *Int. J. Neural System*, Vol. 1, No. 3, 1989

[PT86] T. Poggio and V. Torre, "On Edge Detection", *IEEE Trans.on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 2, pp.147-163, March 1986

[PTK85] T. Poggio, V. Torre and C. Koch, "Computational vision and regularization theory", *Nature*, Vol. 317, pp. 314-319, 26 Sept 1985

[PYHLGB89] D. Psaltis, A.A. Yamamura, K. Hsu, S. Lin, X. Gu and D. Brady, "Optoelectronic Implementations of Neural Networks," *IEEE Communications Magazine*, Vol. 27, No. 11, Nov 1989, pp. 37-40

[RC92] A. Rattarangsi and R.C. Chin, "Scale-Based Detection of Corners of Planar Curves," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 4, April 1992, pp. 430-449

[RM86] D. Rumelhart and J. McCleland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I & II*, MIT Press, Cambridge MA, 1986

[RTT90] G.V. Reklaitis, A.G. Tsirukis, M.F. Tenorio, "Generalized Hopfield Networks and Nonlinear Optimization," *Advances in Neural Information Processing Systems* 2, CA:Morgan Kaufmann Publishers, 1990. pp. 355-362

[RV91] F. Romeo and A. Sangiovanni-Vincentelli, "A Theoretical Framework for Simulated Annealing," *Algorithmica*, 6:302-345, 1991

[Sut92] D.Suter, "Mixed Finite Element Based Neural Networks in Visual Reconstruction," *Intl. J. of Pattern Recognition and Artificial Intelligence*, Vol. 6, No. 1, 1992, pp. 113-129

[ST88] M.A. Styblinski and T.S. Tang, "Stochastic approximation algorithm with function smoothing vs. simulated annealing," *Abstracts of the First Annual International Neural Network Society Meeting*, 1988, pp. 138

[Ter86] D. Terzopoulos, "Regularization of Inverse Visual Problems Involving Discontinuities," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 4, July 1986, pp. 413-423

[TG86] M. Takeda and J.W. Goodman, "Neural networks for computation: number representations and programming complexity," *Applied Optics*, Vol. 25, No. 18, Sept 1986, pp. 3033-3046

[TGD89] H.L. Tan, S.B. Gelfand and E.J. Delp, "A Comparative Cost Function Approach to Edge Detection," *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 10, N0. 6, Nov/Dec, pp. 1337-1349

[TGD91] H.L. Tan, S.B. Gelfand and E.J. Delp, "A Cost Minimization Approach to Edge Detection Using Simulated Annealing," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 1, Jan 1991, pp. 3-18

[TH86] D.W. Tank and J.J. Hopfield, "Simple "Neural" Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, Vol. 33, No. 5, May 1986, pp. 533-541

[TLA92] Y. Takefuji, K.C. Lee and H. Aiso, "An artificial maximum neural network: a winner-take-all neuron model forcing the state of the system in a solution domain," *Biological Cybernetics*, 67, 1992, pp. 243-251

[VZS75] A.V.Vilkov, N.P.Zhidkow and B.M.Schedrin, "A method of search for the global minimum of a function of one variable," *J. of Comput. Math. and Math. Phys.*, 15, pp. 1040-1042, 1975

[Wit83] A.P. Witkin, "Scale-space Filtering," *Proceedings of the 8th International Joint Conference on Artifical Intelligence*, Aug 1983, pp. 1019-1022

[WP88] G.V. Wilson and G.S. Pawley, "On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank," *Biological Cybernetics*, 58, 1988, pp. 63-70

[WS92] D.J. Williams and M. Shah, "A Fast Algorithm for Active Contours and Curvature Estimation," *CVGIP: Image Understanding*, Vol. 55, No. 1, Jan 1992, pp. 14-26

[WZX92] T. Wang, X. Zhuang and X. Xing, "Robust segmentation of noisy images using a neural network model," *Image and Vision Computing*, Vol. 10, No. 4, March 1992, pp. 233-240

[XT91] X. Xu and W.T. Tsui, "Effective Neural Algorithms for the Travelling Salesman Problem," *Neural Networks*, Vol. 4, 1991, pp. 193-205

[Yao88] Yong Yao, "Dynamic tunneling algorithm and simulated annealing circuit for global optimization," *Abstracts of the First Annual International Neural Network Society Meeting*, 1988, pp. 152

[Yao89] Yong Yao, "Dynamic Tunneling Algorithm for Global Optimization," *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 19, NO. 5, Sept/Oct 1989, pp. 1222-1230

[YP86] A.L. Yuille and T.A. Poggio, "Scaling Theorems for Zero Crossing," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 8, Jan 1986, pp. 15-25

[YT92] S.S. Yu and W.H. Tsai, "Relaxation by the Hopfield neural network," *Pattern Recognition*, Vol. 25, No. 2, 1992, pp. 197-209

[ZC92a] S. Zhang and A.G. Constantinides, "Lagrange Programming Neural Networks," *IEEE Trans. on Circuits and Systems*, Vol. 39, No. 7, July 1992, pp. 441-452

[ZC92b] Y.T. Zhou and R. Chellappa, *Artificial Neural Networks for Computer Vision*, Springer-Verlag:New York, 1992

[ZCVJ88] Y.T. Zhou, R. Chellappa, A. Vaid and B.K. Jenkins, "Image Restoration Using a Neural Network," *IEEE Trans. on Acoustics, Speech and Signal Processing*, Vol. 36, No. 7, July 1988, pp. 1141-1151

Appendix I:

The assignment of the boundary connection of 2-D recurrent neural network for gaussian filtering:

The corresponding energy of the 2-D recurrent neural network for gaussian filtering is restated here for reference.

$$E = \frac{\lambda}{\sigma_{ext}} \sum_x \sum_y |\hat{d}_{xy} - d_{xy}|^2 + \frac{1-\lambda}{\sigma_{int}} \sum_x \sum_y \left\{ |\hat{d}_{x+1,y} - 2\hat{d}_{x,y} + \hat{d}_{x-1,y}|^2 + |\hat{d}_{x,y+1} - 2\hat{d}_{x,y} + \hat{d}_{x,y-1}|^2 \right\} \quad ..(A1.1)$$

The consideration of the assignment can be divided into eight parts A-H as illustrated in Fig. A1.1.

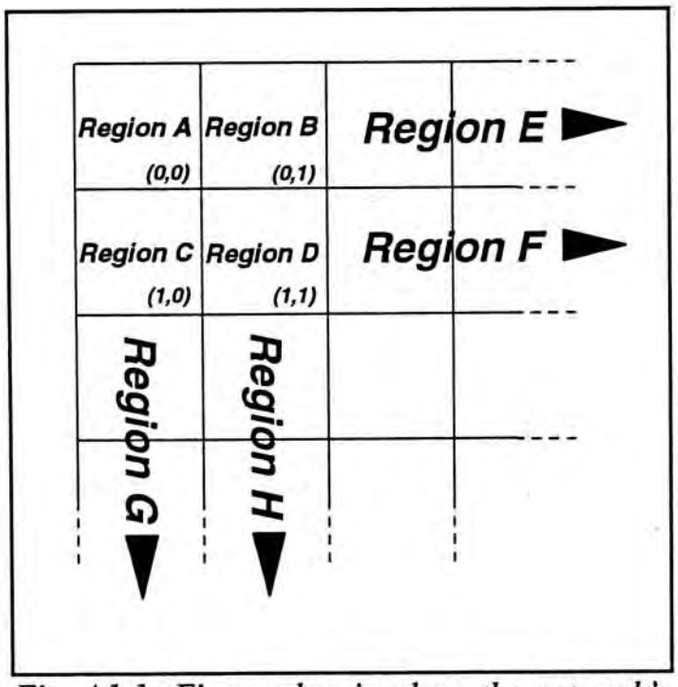

Fig. A1.1: Figure showing how the network's boundary is divided for connection weights assignment

Part A: The terms involving $\hat{d}_{00}$ in eqn.(A1.1), denoted by E_{00}, are given as:

$$E_{00} = \alpha \left(\hat{d}_{00} - d_{00}\right)^2 + \beta \left\{ \left(\hat{d}_{02} - 2\hat{d}_{01} + \hat{d}_{00}\right)^2 + \left(\hat{d}_{20} - 2\hat{d}_{10} + \hat{d}_{00}\right)^2 \right\}$$

$$\frac{\partial E_{00}}{\partial \hat{d}_{00}} = 2\alpha\left(\hat{d}_{00}-d_{00}\right) + 2\beta\left\{\left(\hat{d}_{02}-2\hat{d}_{01}+\hat{d}_{00}\right) + \left(\hat{d}_{20}-2\hat{d}_{10}+\hat{d}_{00}\right)\right\}$$

$$= \hat{d}_{00}(2\alpha+4\beta) - 4\beta\hat{d}_{01} - 4\beta\hat{d}_{10} + 2\beta\hat{d}_{02} + 2\beta\hat{d}_{00} - 2\alpha d_{00}$$

The corresponding connection weights are:

$$\begin{aligned} T_{00;00} &= -(2\alpha + 4\beta) \\ T_{01;00} &= T_{10;00} = 4\beta \\ T_{02;00} &= T_{20;00} = -2\beta \\ I_{00} &= 2\alpha d_{00} \end{aligned}$$

Part B: The terms involving $\hat{d}_{01}$ in eqn.(A1.1), denoted by E_{01}, are given as:

$$E_{01} = \alpha\left(\hat{d}_{01}-d_{01}\right)^2 + \beta\left\{\left(\hat{d}_{01}-2\hat{d}_{11}+\hat{d}_{21}\right)^2 + \left(\hat{d}_{00}-2\hat{d}_{01}+\hat{d}_{02}\right)^2 + \left\{\hat{d}_{01}-2\hat{d}_{02}+\hat{d}_{03}\right)^2\right\}$$

$$\frac{\partial E_{01}}{\partial \hat{d}_{01}} = 2\alpha\left(\hat{d}_{01}-d_{01}\right) + \beta\left\{2(\hat{d}_{01}-2\hat{d}_{11}+\hat{d}_{21})-4\left(\hat{d}_{00}-2\hat{d}_{01}+\hat{d}_{02}\right)+2\left(\hat{d}_{01}-2\hat{d}_{02}+\hat{d}_{03}\right\}\right.$$

$$= -4\beta\hat{d}_{00}+(12\beta+2\alpha)\hat{d}_{01}-8\beta\hat{d}_{02}+2\beta\hat{d}_{03}-4\beta\hat{d}_{11}+2\beta\hat{d}_{21}-2\alpha d_{01}$$

The corresponding connection weights are:

$$\begin{aligned} T_{00;01} &= 4\beta \\ T_{01;01} &= -(12\beta+2\alpha) \\ T_{02;01} &= 8\beta \\ T_{03;01} &= -2\beta \\ T_{11;01} &= 4\beta \\ T_{21;01} &= -2\beta \\ I_{01} &= 2\alpha d_{01} \end{aligned}$$

Part C: The terms involving $\hat{d}_{10}$ in eqn.(A1.1), denoted by E_{10}, are given as:

$$E_{10} = \alpha\left(\hat{d}_{10}-d_{10}\right)^2 + \beta\left\{\left(\hat{d}_{10}-2\hat{d}_{11}+\hat{d}_{12}\right)^2 + \left(\hat{d}_{00}-2\hat{d}_{10}+\hat{d}_{20}\right)^2 + \left\{\hat{d}_{10}-2\hat{d}_{20}+\hat{d}_{30}\right)^2\right\}$$

Comparing with E_{01}, the neural network formulation can be obtained by swapping the first and second index, third and forth index. The corresponding connection weights are:

$$\begin{aligned}
T_{00;10} &= 4\beta \\
T_{10;10} &= -(12\beta+2\alpha) \\
T_{20;10} &= 8\beta \\
T_{30;10} &= -2\beta \\
T_{11;10} &= 4\beta \\
T_{12;10} &= -2\beta \\
I_{10} &= 2\alpha d_{01}
\end{aligned}$$

Part D: The terms involving $\hat{d}_{11}$ in eqn.(A.1), denoted by E_{11}, are given as:

$$E_{11} = \alpha\left(\hat{d}_{11}-d_{11}\right)^2 + \beta\left\{\left(\hat{d}_{10}-2\hat{d}_{11}+\hat{d}_{12}\right)^2 + \left(\hat{d}_{11}-2\hat{d}_{12}+\hat{d}_{13}\right)^2 - 4\left(\hat{d}_{01}-2\hat{d}_{11}+\hat{d}_{21}\right) + 2\left(\hat{d}_{11}-2\hat{d}_{21}+\hat{d}_{31}\right)\right\}$$

$$\frac{\partial E_{11}}{\partial \hat{d}_{11}} = (20\beta+2\alpha)\hat{d}_{11}-4\beta\hat{d}_{10}-4\beta\hat{d}_{01}-8\beta\hat{d}_{12}-8\beta\hat{d}_{21}+2\beta\hat{d}_{31}+2\beta\hat{d}_{13}$$

The corresponding connection weights are:

$$\begin{aligned}
T_{11;11} &= -(20\beta+2\alpha) \\
T_{10;11} &= T_{01;11} = 4\beta \\
T_{12;11} &= T_{21;11} = 8\beta \\
T_{13;11} &= T_{31;11} = -2\beta \\
I_{11} &= 2\alpha d_{11}
\end{aligned}$$

* *According to the network geometric symmetry, the connection weights assignment of the other three corners can be copied from Part A to Part D.*

Part E: For this part of the network, let one of the neurons' value denoted by $\hat{d}_{x,y}$.

(In fact, y=0) The terms involving in eqn.(A1.1), denoted by E_e, are given as:

$$\begin{aligned} E_e = \beta \Big\{ & \left(\hat{d}_{x+1,y}-2\hat{d}_{x,y}+\hat{d}_{x-1,y}\right)^2 + \left(\hat{d}_{x+2,y}-2\hat{d}_{x+1,y}+\hat{d}_{x,y}\right)^2 \\ & + \left(\hat{d}_{x,y}-2\hat{d}_{x-1,y}+\hat{d}_{x-2,y}\right)^2 + \left(\hat{d}_{x,y+2}-2\hat{d}_{x,y+1}+\hat{d}_{x,y}\right)^2 \Big\} \\ & + \alpha \left(\hat{d}_{x,y}-d_{x,y}\right)^2 \end{aligned}$$

$$\begin{aligned} \frac{\partial E_e}{\partial \hat{d}_{x,y}} = \beta & \left\{ 14\hat{d}_{x,y}-8\hat{d}_{x-1,y}-8\hat{d}_{x+1,y}+2\hat{d}_{x-2,y}+2\hat{d}_{x+2,y}+2\hat{d}_{x,y+2}-4\hat{d}_{x,y+1} \right\} \\ & + 2\alpha \left(\hat{d}_{x,y}-d_{x,y}\right) \end{aligned}$$

The corresponding connection weights are:

$$\begin{aligned} T_{x,y;x,y} &= -14\beta-2\alpha \\ T_{x-1,y;x,y} &= T_{x+1,y;x,y} = 8\beta \\ T_{x,y+1;x,y} &= 4\beta \\ T_{x,y+2;x,y} &= -2\beta \\ T_{x-2,y;x,y} &= -2\beta \\ T_{x+2,y;x,y} &= -2\beta \\ I_{xy} &= 2\alpha d_{xy} \end{aligned}$$

Part F: For this part of the network, again let one of the neurons' value denoted by $\hat{d}_{x,y}$. (Here, y=1) The terms involving it in eqn.(A1.1), denoted by E_f, are given as:

$$\begin{aligned} E_f = \beta \Big\{ & \left(\hat{d}_{x+1,y}-2\hat{d}_{x,y}+\hat{d}_{x-1,y}\right)^2 + \left(\hat{d}_{x+2,y}-2\hat{d}_{x+1,y}+\hat{d}_{x,y}\right)^2 \\ & + \left(\hat{d}_{x,y}-2\hat{d}_{x-1,y}+\hat{d}_{x-2,y}\right)^2 + \left(\hat{d}_{x,y+2}-2\hat{d}_{x,y+1}+\hat{d}_{x,y}\right)^2 \\ & + \left(\hat{d}_{x,y-1}-2\hat{d}_{x,y}+\hat{d}_{x,y+1}\right)^2 \Big\} \\ & + \alpha \left(\hat{d}_{x,y}-d_{x,y}\right)^2 \end{aligned}$$

$$\frac{\partial E_f}{\partial \hat{d}_{x,y}} = \beta \left\{ 22\hat{d}_{x,y} - 8\hat{d}_{x-1,y} - 8\hat{d}_{x+1,y} + 2\hat{d}_{x-2,y} + 2\hat{d}_{x+2,y} + 2\hat{d}_{x,y+2} - 8\hat{d}_{x,y+1} - 4\hat{d}_{x,y-1} \right\}$$
$$+ 2\alpha \left(\hat{d}_{x,y} - d_{x,y} \right)$$

The corresponding connection weights are:

$$\begin{aligned}
T_{x,y;x,y} &= -22\beta - 2\alpha \\
T_{x-1,y;x,y} &= T_{x+1,y;x,y} = 8\beta \\
T_{x,y+1;x,y} &= 8\beta \\
T_{x,y+2;x,y} &= -2\beta \\
T_{x-2,y;x,y} &= -2\beta \\
T_{x+2,y;x,y} &= -2\beta \\
T_{x,y-1;x,y} &= 4\beta \\
I_{xy} &= 2\alpha d_{xy}
\end{aligned}$$

Part G: Connection weights can be obtained simply by swapping index of Part E.The connection weights are then:

$$\begin{aligned}
T_{x,y;x,y} &= -14\beta - 2\alpha \\
T_{x,y-1;x,y} &= T_{x,y+1;x,y} = 8\beta \\
T_{x+1,y;x,y} &= 4\beta \\
T_{x+2,y;x,y} &= -2\beta \\
T_{x,y-2;x,y} &= -2\beta \\
T_{x,y+2;x,y} &= -2\beta \\
I_{xy} &= 2\alpha d_{xy}
\end{aligned}$$

Part H: Connection weights can be obtained by swapping index of Part F. The connection weights are then:

$$\begin{aligned}
T_{x,y;x,y} &= -22\beta - 2\alpha \\
T_{x,y-1;x,y} &= T_{x,y+1;x,y} = 8\beta \\
T_{x+1,y;x,y} &= 8\beta \\
T_{x+2,y;x,y} &= -2\beta \\
T_{x,y-2;x,y} &= -2\beta \\
T_{x,y+2;x,y} &= -2\beta \\
T_{x-1,y;x,y} &= 4\beta \\
I_{xy} &= 2\alpha d_{xy}
\end{aligned}$$

** Again, due to the network symmetry, the assignment of the connection weights in the bottom and right boundary can be directly copied from Part E to H.*

Appendix II:

Formula for connection weight assignment of 2-D recurrent neural network for gaussian filtering and the proof on symmetric property

In Section 5.2, a neural network for gaussian filtering is derived and it's shown that the network's connection weight matrix is symmetric with nonnegative diagonal elements and thus the convergence of the network can be guaranteed. However, it has been noticed that the connection weight assignment in the boundary region is very tedious (Refer to Appendix I). If the network's structure is modified, the connection weight assignment in the boundary has to be repeated again. To simplify the assignment task, we generalize the network structure, assuming each neuron connected to m neurons on the left and m neurons on the right. Then, we derive the formula for the connection weight assignment for the core part and the boundaries of the network and the symmetric property of the connection weight matrix is proved.

Consider the neural network for gaussian filtering with N neurons, the terms describing the neighboring relationship of the neurons is given by:

$$\sum_{j=2}^{N-1} |\hat{d}_{j+1} - 2\hat{d}_j + \hat{d}_{j-1}|^2 \qquad ..(A2.1)$$

where $\hat{d}_j$ is the value of the jth neuron.

For other applications, which may utilize other form of neighboring relationship of the neurons, we generalize the expression (A2.1) to the following form:

$$E = \sum_{j=m+1}^{N-m} \left(w_m \hat{d}_{j-m} + w_{m-1}\hat{d}_{j-m+1} + \ldots + w_0 \hat{d}_j + \ldots + w_m \hat{d}_{j+m}\right)^2 \qquad ..(A2.2)$$

where w_m is the coefficient of either $\hat{d}_{j+m}$ of $\hat{d}_{j-m}$.

N is the number of neurons in the network.

The corresponding network will have nodes connected to m neurons on the left and m neurons on the right. To obtain the dynamic equation of the network, firstly consider the

i^{th} neuron in the core part (ie. $\forall\ i$: $2m+1 \le i \le N-2m$)[2]. The terms in E involving it are,

$$
\begin{aligned}
E_i &= \sum_{j=i-m}^{i+m} \left(w_m \hat{d}_{j-m} + w_{m-1}\hat{d}_{j-m+1} + \ldots + w_0 \hat{d}_j + \ldots + w_m \hat{d}_{j+m}\right)^2 \\
&= \left(w_m \hat{d}_{i-2m} + w_{m-1}\hat{d}_{i-2m+1} + \ldots + w_{m-1}\hat{d}_{i-1} + w_m \hat{d}_i\right)^2 \\
&+ \left(w_m \hat{d}_{i-2m+1} + \ldots + w_{m-2}\hat{d}_{i-1} + w_{m-1}\hat{d}_i + w_m \hat{d}_{i+1}\right)^2 \\
&+ \ldots\ldots \\
&+ \left(w_m \hat{d}_i + w_{m-1}\hat{d}_{i+1} + \ldots + w_m \hat{d}_{i+2m}\right)^2
\end{aligned}
\qquad ..(A2.3)
$$

Then, differentiate E with respect to $\hat{d}_i$,

$$
\begin{aligned}
\frac{\partial E}{\partial \hat{d}_i} &= 2w_m\left(w_m \hat{d}_{i-2m} + w_{m-1}\hat{d}_{i-2m+1} + \ldots + w_{m-1}\hat{d}_{i-1} + w_m \hat{d}_i\right) \\
&+ 2w_{m-1}\left(w_m \hat{d}_{i-2m+1} + \ldots + w_{m-2}\hat{d}_{i-1} + w_{m-1}\hat{d}_i + w_m \hat{d}_{i+1}\right) \\
&+ \ldots\ldots \\
&+ 2w_m\left(w_m \hat{d}_i + w_{m-1}\hat{d}_{i+1} + \ldots + w_m \hat{d}_{i+2m}\right)
\end{aligned}
\qquad ..(A2.4)
$$

According to the dynamic equation (A2.4), the connection weight from the nth neighbor to the current neuron, $T_{i+n,i}$ and $T_{i-n,i}$ equals the negative value of the coefficient of $\hat{d}_{i+n}$ and $\hat{d}_{i-n}$ and can be formulated as:

$\forall i$: $2m+1 \le i \le N-2m \ \wedge\ \forall n$: $0 \le n \le 2m$

$$
T_{i-n,i} = T_{i+n,i} = 2\sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.5)
$$

It's noted that the connection weights are independent of i and thus are geometric invariant within the core part of the network. The proof of the connection weight's symmetric property is pretty straight forward and is given at the back of this Appendix.

Then, consider the ith neuron in the boundaries, ie. $\forall i$: $1 \le i \le 2m$ & $N-2m+1 \le i \le N$. As

---

2 The indexing scheme is such that the leftest neuron has an index 1 and the rightest neuron has the index N.

there are two boundaries, one being on the left side and the other on the right side, here we just consider the left side and the situation on the right side is exactly the same. For the neurons in the left boundary, the number of terms corresponding to the neighboring relationship with the other neurons, comparing with eqn.(A2.3) is reduced due to the boundary existence. And the number of reduced terms depends on i and thus the connection weights in the boundary are geometric specific.

To derive the formula for the connection weights in the left boundary, let's consider the terms in E involving node i in the boundary. Take the derivative and note that coefficients of $\hat{d}_{i-n}$ and $\hat{d}_{i+n}$, the connection weights from the nth node to the left and right of the current neuron are respectively:

$\forall i$: $1 \le i \le 2m \ \wedge \ \forall n$: $0 \le n \le i$

$$T_{i-n,i} = 2\sum_{j=2m-i}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.6)$$

$\forall i$: $1 \le i \le 2m \ \wedge \ \forall n$: $0 \le n \le 2m$

It can be proved that the connection weight symmetric property is also hold in the boundary and the mathematical proof is given at the back of this Appendix. Moreover, as it's obvious that all the above derivation remains the same if we reverse the indexing scheme, ie. the rightest neuron has index no. 1 and the leftest one N, the connection weight at the right boundary can be easily deduced to be:

$\forall i$: $N-2m+1 \le i \le N \ \wedge \ \forall n$: $0 \le n \le N-i$

$$T_{i+n,i} = 2\sum_{j=2m-N+i}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.8)$$

$\forall i$: $N-2m+1 \le i \le N \ \wedge \ \forall n$: $0 \le n \le 2m$

$$T_{i-n,i} = 2\sum_{j=0}^{\min(2m-n,N-i)} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.9)$$

The symmetric property remains to be hold. Thus, it can be concluded that the way

we adopted to assign the connection weights of the recurrent neural network including the boundary region can be formulated as eqn.(A2.5-9) and can guarantee the connection weight symmetric property throughout the network. Thus, the convergence of the network is proved.

*** Note to 2-D case:**

All the work is done on 1-D gaussian filtering. However, extension to 2-D case is pretty straight forward and the symmetric property will be inherited then.

Firstly, the energy terms involving neuron with index (i,k), which describes the horizontal neighboring relationship with m neurons on both left and right side and vertical neighboring relationship with p neurons on both upper and lower side, are:

$\forall i$: $2m+1 \le i \le N-2m \quad \wedge \quad \forall k$: $2p+1 \le k \le P-2p$

$$E_{ik} = \sum_{j=i-m}^{i+m} \left(w_m \hat{d}_{j-m,k} + w_{m-1} \hat{d}_{j-m+1,k} + \ldots + w_o \hat{d}_{j,k} + \ldots + w_m \hat{d}_{j+m,k} \right)^2 + \sum_{l=k-p}^{k+p} \left(r_p \hat{d}_{i,l+p} + r_{p-1} \hat{d}_{i,l+p-1} + \ldots + r_o \hat{d}_{i,l} + \ldots + r_p \hat{d}_{i,l-p} \right)^2$$

where N is horizontal dimension of the network,

P is vertical dimension of the network.

As the energy terms corresponding to the vertical and horizontal neighboring relationship are independent, the connection weight can be easily deduced from the case of 1D and referring to eqn.(A2.10-11), their formula are:

$$T_{i,k;i,k} = 2\sum_{j=0}^{2m} w_{|m-j|}^{2} + 2\sum_{l=0}^{2p} r_{|p-l|}^{2}$$

$\forall n$: $1 \le n \le 2m$

$$T_{i-n,k;i,k} = T_{i+n,k;i,k} = 2\sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.10)$$

$\forall n$: $1 \le n \le 2p$

$$T_{i,k-n;i,k} = T_{i,k+n;i,k} = 2\sum_{l=0}^{2p-n} r_{|p-l|} r_{|-p+n+l|} \qquad ..(A2.11)$$

At the boundaries, first consider the upper corner. The energy terms involving neuron in that region are:

$\forall i$: $1 \le i \le 2m$ $\wedge$ $\forall k$: $1 \le k \le 2p$

$$E_{ik} = \sum_{j=m+1}^{N-m} \left(w_m \hat{d}_{j-m,k} + w_{m-1} \hat{d}_{j-m+1,k} + \ldots + w_o \hat{d}_{j,k} + \ldots + w_m \hat{d}_{j+m,k} \right)^2 + \sum_{l=p+1}^{P-p} \left(r_p \hat{d}_{i,l+p} + r_{p-1} \hat{d}_{i,l+p-1} + \ldots + r_o \hat{d}_{i,l} + \ldots + r_p \hat{d}_{i,l-p} \right)^2$$

Again due to the independence between the vertical and horizontal relationship. The corresponding connection assignment formula are,

$$T_{i,k;i,k} = 2\sum_{j=0}^{i} w_{|m-j|}^{2} + 2\sum_{l=0}^{k} r_{|p-l|}^{2}$$

$\forall n$: $1 \le n \le i$

$$T_{i-n,k;i,k} = 2\sum_{j=2m-i}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.12)$$

$\forall n$: $1 \le n \le 2m$

$$T_{i+n,k;i,k} = 2\sum_{j=0}^{\min(2m-n,i)} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.13)$$

$\forall n$: $1 \le n \le k$

$$T_{i,k-n;i,k} = 2\sum_{l=2p-k}^{2p-n} r_{|p-l|} r_{|-p+n+l|} \qquad ..(A2.14)$$

$\forall n$: $1 \le n \le p$

$$T_{i,k+n;i,k} = 2\sum_{l=0}^{\min(2p-n,k)} r_{|p-l|} r_{|-p+n+l|}$$

The connection weights in the other three corner can be obtained in a similar manner.

<u>Proof on the symmetric property of the connection weight matrix derived for 1-D gaussian filtering:</u>

The core part includes the neurons with index within the range $[2m+1, N-2m]$ while the boundaries part includes neurons with index within the range $[1, 2m]$ and $[N-2m+1, N]$.

i) <u>Prove the connection weight symmetric property in the core part</u>:

The formula for the connection weight in the core part is restated here.

$\forall i$: $2m+1 \le i \le N-2m \;\wedge\; \forall n$: $0 \le n \le 2m$

$$T_{i-n,i} = T_{i+n,i} = 2\sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|}$$

To prove the symmetric property, we have to show that:

$$a)\; T_{i-n,i} = T_{i,i-n}\,,$$
$$b)\; T_{i+n,i} = T_{i,i+n}$$

a) For $2m+1 \le i \le 4m$, $T_{i,i-n}$ is defined by eqn.(A2.7)

$$\begin{aligned} T_{i,i-n} &= 2 \sum_{j=0}^{\min(2m-n,i)} w_{|m-j|} w_{|-m+n+j|} \\ &= 2 \sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad as\ i > 2m-n \\ &= T_{i-n,i} \end{aligned}$$

For $4m+1 \le i \le N-2m$, $T_{i,i-n}$ is then defined by eqn.(A2.5)

$$\begin{aligned} T_{i,i-n} &= T_{i'+n,i'} \qquad \forall\ 2m+1 \le i' \le N-4m \\ &= 2 \sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \\ &= T_{i-n,i} \end{aligned}$$

Therefore, $\forall\ 2m+1 \le i \le N-2m$, it's proved that $T_{i,i-n} = T_{i-n,i}$.

b) For $N-4m+1 \le i \le N-2m$, $T_{i,i+n}$ is defined by eqn.(A2.9)

$$\begin{aligned} T_{i,i+n} &= 2 \sum_{j=0}^{\min(2m-n,N-i)} w_{|m-j|} w_{|-m+n+j|} \\ &= 2 \sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad as\ N-i > 2m-n \\ &= T_{i+n,i} \end{aligned}$$

For $2m+1 \le i \le N-4m$, $T_{i,i+n}$ is then defined by eqn.(A2.5)

$$\begin{aligned} T_{i,i+n} &= T_{i'-n,i'} \qquad \forall\ 4m+1 \le i' \le N-2m \\ &= 2 \sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \\ &= T_{i+n,i} \end{aligned}$$

Therefore, $\forall\ 2m+1 \le i \le N-2m$, it's proved that $T_{i,i+n} = T_{i+n,i}$.

which completes the proof. ■

ii) <u>Prove the connection weight symmetric property in the boundaries part</u>:

First, consider the left boundary, the formula for the connection weights is:

$\forall i$: $1 \le i \le 2m$ $\wedge$ $\forall n$: $0 \le n \le i$

$$T_{i-n,i} = 2 \sum_{j=2m-i}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.16)$$

$\forall i$: $1 \le i \le 2m$ $\wedge$ $\forall n$: $0 \le n \le 2m$

$$T_{i+n,i} = 2 \sum_{j=0}^{\min(2m-n,i)} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.17)$$

A) When $min(2m\text{-}n,i) = 2m\text{-}n$, $2m\text{-}n < i$ or $i+n > 2m$.

To prove $T_{i+n,i} = T_{i,i+n}$:

$$T_{i+n,i} = 2 \sum_{j=0}^{2m-n} w_{|m-j|} w_{|-m+n+j|} \qquad ..(A2.18)$$

describes the connection from the neurons with index no. greater than $2m$. The reverse connection weight value, $T_{i,i+n}$ thus is determined by eqn.(A2.5) which agrees with eqn.(A2.18).

To prove $T_{i-n,i} = T_{i,i-n}$:
From eqn.(A2.16),

$$T_{i+n,i} = 2 \sum_{j=0}^{\min(2m-n,i)} w_{|m-j|} w_{|-m+n+j|}$$

Put $i = i - n$,

$$\begin{aligned}
T_{i,i-n} &= 2 \sum_{j=0}^{\min} (2m-n,i-n) w_{|m-j|} w_{|-m+n+j|} \\
&= 2 \sum_{j=0}^{i-n} w_{|m-j|} w_{|-m+n+j|} \\
&= 2 \sum_{j=2m-n}^{2m-i} w_{|-m+n+j|} w_{|m-j|} \\
&= 2 \sum_{j=2m-i}^{2m-n} w_{|-m+n+j|} w_{|m-j|} \\
&= 2 \sum_{j=2m-i}^{2m-n} w_{|-m+n+j|} w_{|m-j|} \\
&= T_{i-n,i}
\end{aligned}$$

■

B) When $min(2m\text{-}n,i) = i$,

To prove $T_{i+n,i} = T_{i+n,i}$:

Starting with

$$T_{i-n,i} = 2\sum_{j=2m-i}^{2m-n} w_{|m-j|}w_{|-m+n+j|}$$

Put $i = i + n$,

$$\begin{aligned} T_{i,i+n} &= 2\sum_{j=2m-i-n}^{2m-n} w_{|m-j|}w_{|-m+n+j|} \\ &= 2\sum_{j=-i}^{0} w_{|m-j-2m+n|}w_{|-m+n+j+2m-n|} \\ &= 2\sum_{j=i}^{0} w_{|-m+n+j|}w_{|m-j|} \\ &= 2\sum_{j=0}^{i} w_{|m-j|}w_{|-m+n+j|} \\ &= T_{i+n,i} \end{aligned}$$

■

To prove $T_{i,i-n} = T_{i-n,i}$:

Starting with

$$T_{i+n,i} = 2\sum_{j=0}^{i} w_{|m-j|}w_{|-m+n+j|}$$

Put $i = i - n$,

$$\begin{aligned} T_{i,i-n} &= 2\sum_{j=0}^{i-n} w_{|m-j|}w_{|-m+n+j|} \\ &= 2\sum_{j=-2m+n}^{i-2m} w_{|m-j-2m+n|}w_{|-m+n+j+2m-n|} \\ &= 2\sum_{j=2m-i}^{2m-n} w_{|m-j|}w_{|-m+n+j|} \\ &= T_{i-n,i} \end{aligned}$$

■

This completes the proof. ■

Appendix III

On index manipulation for reshaping higher-order Hopfield energy function

In Chapter 3, using linear approximation to reshape the Hopfield energy function, the energy function of eqn.(3.9) can be written as eqn.(3.12) by grouping different terms together. The grouping and the index manipulation are depicted in the following.

Consider the expression

$$\frac{1}{k}\sum_{i_1}\sum_{i_2}\cdots\sum_{i_k} T_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \ldots x_{i_k} \tag{A3.1}$$

and assume that the value of the weighting, $T_{i_1 i_2 \ldots i_k}$ is independent of the index ordering. The assumption can be ensured by using the averaging method described by eqn.(3.11).

Definition A3.1: The **distinct index set** of a weighting is defined as a set with elements being the distinct indices of the weighting. For example, the distinct index set of T_{111233} and T_{644236} are {1,2,3} and {2,3,4,6} respectively.

Recall from Chapter 3, the basic idea of the reshaping strategy is to replace x_i^k by x_i. Refer to the expression (A3.1), let m be the number of different indices, k be the total number of indices, $\{i_1, i_2, \ldots i_m\}$ be the distinct index set and denote the number of occurrence of the index i_j as λ_j, where $\sum_{i=1}^{m} \lambda_i = k$. Using the strategy, $x_{i_1}^{\lambda_1} x_{i_2}^{\lambda_2} \ldots x_{i_m}^{\lambda_m}$ has to be substituted by $x_{i_1} x_{i_2} \ldots x_{i_m}$ for any possible set of λ. After the substitution, the weightings with same distinct index set will be grouped together.

i) For a fixed m and a particular set of λ, the number of weightings with λ_1 index i_1, λ_2 index i_2, ..., λ_m index i_m within the expression (A3.1) equals

$$\frac{k!}{\lambda_1!\lambda_2!\ldots\lambda_m!}$$

Due to the assumption that the value of the weighting being independent of the index ordering,

$$T_{i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}} = T_{<i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}>} = constant$$

where $T_{i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}}$ is the weighting with λ_1 index i_1, followed by λ_2 index i_2, ... followed by λ_m index i_m. $< i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m} >$ stands for any possible combination of λ_1 index i_1, λ_2 index i_2, ... λ_m index i_m.

Therefore, the sum of the weightings of $x_{i_1} x_{i_2} \ldots x_{i_m}$ for a fixed m and a particular set of λ

$$= \frac{k!}{\lambda_1! \lambda_2! \ldots \lambda_m!} T_{i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}}$$

ii) By grouping together the weightings with same distinct index set, i.e. grouping of T_{11123}, T_{12333}, and so on, the sum will then be

$$\sum_{\lambda_1=1}^{k-m+1} \sum_{\lambda_2=1}^{k-\lambda_1-m+2} \cdots \sum_{\lambda_{m-1}=1}^{k-1-\sum_{i=1}^{m-2}\lambda_i} \frac{k!}{\lambda_1! \lambda_2! \ldots \lambda_m!} T_{i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}}$$

iii) Afterwards, this sum should be averaged and assigned to every element of the new weighting set $T_{<i_1 i_2 \ldots i_m>}$. This is to ensure the derived network convergence by the similar argument of eqn.(3.11). The sum is corresponding to the term $\sum_{s \in <i_1, \ldots, i_p>} T_s^{(old)}$ of eqn.(3.11) and size($<i_1, \ldots i_m>$) equals $m!$. Thus, each element of the new weighting set $T_{<i_1 i_2 \ldots i_m>}$ will be

$$\sum_{\lambda_1=1}^{k-m+1} \sum_{\lambda_2=1}^{k-\lambda_1-m+2} \cdots \sum_{\lambda_{m-1}=1}^{k-1-\sum_{i=1}^{m-2}\lambda_i} \frac{k!}{m!} \frac{1}{\lambda_1! \lambda_2! \ldots \lambda_m!} T_{i_1^{\lambda_1} i_2^{\lambda_2} \ldots i_m^{\lambda_m}}$$

iv) Also, the number of different indices, m can be 1 to k-1. Therefore, the expression (A3.1) can be written as

$$\frac{1}{k}\left(\sum\sum_{i_1\neq i_2\ldots\neq i_k}\ldots\sum T_{i_1 i_2\ldots i_k}x_{i_1}x_{i_2}\ldots x_{i_k}\right.$$
$$\left.+\sum_{m=1}^{k-1}\sum_{i_1\neq i_2,\ldots\neq i_m}\ldots\sum\sum_{\lambda_1=1}^{k-m+1}\sum_{\lambda_2=1}^{k-\lambda_1-m+2}\ldots\sum_{\lambda_{m-1}=1}^{k-1-\sum_{t=1}^{m-2}\lambda_t}\frac{k!}{m!}\cdot\frac{1}{\lambda_1!\lambda_2!\ldots\lambda_m!}T_{i_1^{\lambda_1}i_2^{\lambda_2}\ldots i_m^{\lambda_m}}x_{i_1}x_{i_2}\ldots x_{i_m}\right)$$

By repeating the above steps, eqn.(3.12) can be obtained from eqn.(3.9) in a similar manner.