หลักการเพิ่มอัตราการลู่เข้าและการวางนัยทั่วไปสำหรับโครงข่ายป้อนไปข้างหน้าชนิดเซลล์ประสาทคล้ายซิกมอยด์

นายคำรณ สุนัติ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรดุษฎีบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์ ภาควิชาคณิตศาสตร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2546
ISBN 974-17-5120-6

PRINCIPLES OF CONVERGENT RATE AND GENERALIZATION ENHANCEMENT FOR FEEDFORWARD SIGMOID-LIKE NETWORK

Khamron Sunat

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Computer Science

Department of Mathematics
Faculty of Science
Chulalongkorn University

Academic year 2003
ISBN 974-17-5120-6

Thesis Title	PRINCIPLES OF CONVERGENT RATE AND GENERALIZATION ENHANCEMENT FOR FEEDFORWARD SIGMOID-LIKE NETWORK
By	Mr. Khamron Sunat
Field of Study	Computer Science
Thesis Advisor	Professor Chidchanok Lursinsap, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctor's Degree

.....Dean of the Faculty of Science
(Professor Piamsak Menasveta, Ph.D.)

THESIS COMMITTEE

.....Chairman
(Associate Professor Jack Asavanant, Ph.D.)

.....Thesis Advisor
(Professor Chidchanok Lursinsap, Ph.D.)

..... Member
(Siripun Sanguansintukul, Ph.D.)

..... Member
(Assistant Professor Krung Sinapiromsaran, Ph.D.)

..... Member
(Associate Professor Kosin Chamnongthai, Ph.D.)

..... Member
(Assistant Professor Thitipong Tanprasert, Ph.D.)

..... Member
(Chularat Tanprasert, Ph.D.)

นายคำรณ สุนัติ : หลักการเพิ่มอัตราการลู่เข้าและการวางนัยทั่วไปสำหรับโครงข่ายป้อนไปข้างหน้าชนิดเซลล์ประสาทคล้ายซิกมอยด์: (PRINCIPLES OF CONVERGENT RATE AND GENERALIZATION ENHANCEMENT FOR FEEDFORWARD SIGMOID-LIKE NETWORK) อ. ที่ปรึกษา : ศาสตราจารย์ ดร. ชิดชนก เหลือสินทรัพย์, 82 หน้า. ISBN 974-17-5120-6.

วิทยานิพนธ์ฉบับนี้แสดงการสร้างตัวก่อกำเนิดพหุนามเป็นช่วง ชนิด พี-รีเคอร์ซีฟ และอนุพันธ์ เวลาที่ใช้คำนวณการป้อนไปข้างหน้าของโครงข่ายชนิดป้อนไปข้างหน้าหลายชั้น สามารถทำให้ลดลงได้โดยการใช้ฟังก์ชันที่เสนอเป็นฟังก์ชันกระตุ้น มีการนำเสนอการปรับแต่งสามอย่างคือ (1) ฟังก์ชันผิดพลาดที่ปรับแต่งแล้วถูกใช้เพื่อขจัดตัวประกอบอนุพันธ์ซิกมอยด์ของการปรับค่าใหม่ที่หน่วยข้อมูลออก (2) รูปแบบข้อมูลเข้าถูกทำให้เป็นบรรทัดฐาน เพื่อทำให้พิสัยพลศาสตร์ของข้อมูลเข้ามีความสมดุล (3) ฟังก์ชันลงโทษใหม่ถูกเพิ่มให้แก่ชั้นซ่อน เพื่อให้ได้กฎ ปฏิ-เฮบเบียน ซึ่งจะให้ข้อมูลแก่ชั้นซ่อนในขณะที่ฟังก์ชันกระตุ้นมีตัวประกอบอนุพันธ์ซิกมอยด์เป็นศูนย์

การปรับแต่งทั้งสามอย่าง ถูกใช้ผสมกับขั้นตอนวิธี เอสเออาร์พรอพ กระบวนการที่ถูกเสนอให้ผลลัพธ์ที่ดีมาก โดยไม่ต้องระมัดระวังการเลือกพารามิเตอร์สำหรับการสอน ไม่ใช่เพียงขั้นตอนวิธีเท่านั้นแต่รูปร่างของฟังก์ชันกระตุ้นก็เช่นกันที่มีอิทธิพลอย่างมากต่อสมรรถนะการสอน

ภาควิชา **คณิตศาสตร์** ลายมือชื่อนิสิต.....

สาขาวิชา **วิทยาการคอมพิวเตอร์** ลายมือชื่ออาจารย์ที่ปรึกษา.....

ปีการศึกษา 2546

4273840123 : MAJOR COMPUTER SCIENCE

KEY WORD: FEEDFORWARD NETWORK /ANTI-HEBBIAN RULE / TRAINING ALGORITHM /ACTIVATIONFUNCTION / GENERALIZATION / SUCCESS RATE.

KHAMRON SUNAT: PRINCIPLES OF CONVERGENT RATE AND GENERALIZATION ENHANCEMENT FOR FEEDFORWARD SIGMOID-LIKE NETWORK. THESIS ADVISOR: PROF. CHIDCHANOK LURSINSAP, Ph.D., 82 pp. ISBN 974-17-5120-6.

This dissertation demonstrates how the p-recursive piecewise polynomial (p-RPP) generators and their derivatives are constructed. The feedforward computational time of a multilayer feedforward network can be reduced by using these functions as the activation functions. Three modifications of training algorithms are proposed. First, the modified error function is used so that the sigmoid prime factor for the updating rule of the output units is eliminated. Second, the input patterns are normalized in order to balance the dynamic range of the inputs. Third, a new penalty function is introduced to the hidden layer to get the anti-Hebbian rules providing information when the activation functions have zero sigmoid prime factor.

The three modifications are combined with two versions of Rprop (Resilient propagation) algorithm. The proposed procedures achieved the excellent results without the need for careful selection of the training parameters. Not only the algorithm but also the shape of the activation function has important influence on the training performance.

Department **Mathematics** Student's signature.....

Field of study **Computer Science** Advisor's signature.....

Academic year **2003**

Acknowledgments

A few lines of acknowledgments do not fully express my gratitude and appreciation for those guided and supported me through these last five years. I have been fortunate to be surrounded by teachers, family and friends with whom I can share all my joys and frustrations.

I am greatly indebted to Professor Chidchanok Lursinsap for his contributions to all aspects of my work on this thesis, as well as my life at AVIC. He was a great teacher, educator, and advisor, who provided a lot of the ideas and inspiration for the thesis, and taught me the beauty of neural computation with his unparalleled intuition and excitement. He was the most enthusiastic and generous research supervisor and co-author, and without his contributions and guidance this would not have been possible.

There is one more unofficial supervisor for this dissertation: Assoc. Prof. C.-H. H. Chu of the CACS, ULL. I thank him both for the close collaboration on this thesis, as well as for the pleasant atmosphere he created since I visited the CACS.

The financial support of the RGJ grant from the TRF is also gratefully acknowledged.

I am greatly thankful to the entire AVICists, to all people who make it what it is. Assoc. Prof. Suchada and the students all contributed to the environment and atmosphere of the AVIC, making it an exciting place to come to every day and night, be it for work or socializing, and often both.

Finally, I would like to thank my parents and family whose support and encouragement has been constant from day one.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

MFNN	Multilayer Feedforward Neural Network
tanh	Hyperbolic Tangent Function
tanh-like	Approximating Function of Hyperbolic Tangent Function
sigmoid-like	Approximating Function of Sigmoidal Function
KTLF	Kwan's Tanh-Like Function
ZSLF	Zhang's Sigmoid-Like Function
AP	Adaptive Polynomial
CRS	Adaptive Catmull-Rom Spline
p-RPP	p-Recursive Piecewise Polynomial
p-RPPNN	p-Recursive Piecewise Polynomial Neural Network
p-RPPRBF	p-Recursive Piecewise Polynomial Radial Basis Function
MLP	Multilayered Perceptron
BP	Error Backpropagation Algorithm
Rprop	Resilient Propagation Algorithm
iRprop$^+$	Resilient Propagation with Weight Backtracking Algorithm
SARPROP	Simulated Annealing and Weight Decay Resilient Propagation Algorithm

CEN-RPROP	Cross Entropy Normalized Input Rprop Algorithm
CEN-iRprop$^+$	Cross Entropy Normalized Input iRprop$^+$ Algorithm
CEN-SARPROP	Cross Entropy Normalized Input SARPROP
ECEN-iRprop$^+$	Enhanced CEN-iRprop$^+$
ECEN-SARPROP	Enhanced CEN-SARPROP
SSEN-iRprop$^+$	Sum-Squared Error Normalized Input iRprop$^+$ Algorithm
SSEN-iRprop$^+$APRS	Sum-Squared Error Normalized Input iRprop$^+$ Algorithm Combined with Adaptive Regularization Parameter Selection
SSEN-SARPROP	Sum-Squared Error Normalized Input SARPROP
ESSEN-iRprop$^+$APRS	Enhanced SSEN-iRprop$^+$ Combined with Adaptive Regularization Parameter Selection
ESSEN-SARPROP	Enhanced SSEN-SARPROP
SSE	Sum Squared Error
MSE	Mean Squared Error
CE2	Oh's Modified Error Function, $n = 2$
LUT	The Look Up Table
RAM	Random-Access Memory
ROM	Read Only Memory

List of Symbols

$N^{(l)}$	Dimension of Layer l
$\mathbf{x^{(l)}}$	State Vector of Layer l
$\mathbf{x^{(0)}}$	Input Vector of Neural Network
$\mathbf{x^{(2)}}$	Output Vector of Neural Network
$\hat{\mathbf{x}}_j^{(l)}$	Activation Value of Node j in Layer l
$\mathbf{w}$	Weight Vector
w_{ji}^l	Weight Connected to Node i of Layer $l-1$ and Node j of Layer l
$E(\mathbf{w})$	Classification Error Function
$E_B(\mathbf{w})$	Added Error Function
$\overline{E}(\mathbf{w})$	Total Error Function
$f(.)$	Activation Function
$\phi(.)$	Basis Function for Generating the p-RPP Function
G	Simplified Sigmoidal Activation Function
G'	Derivative of Simplified Sigmoidal Activation Function
δ	Generalized Delta Term
s	Index of Pattern
S	Number of Patterns in Data Set
$H_{j,s}$	The Distance Between the Activation Value of Neuron j in Layer l due to the Input Pattern $\mathbf{x}^s$
$\mathbf{t^s}$	Target Vector Corresponding to $\mathbf{x}^s$

CHAPTER I

Introduction

A multilayer feedforward neural network (MFNN) [1, 2] is a popular tool for pattern classification. It typically comprises several layers of interconnected neurons, each of which outputs a value by applying a nonlinearity to a weighted sum of the input. The hyperbolic tangent, tanh, and the sigmoid, $1/(1 + \exp(-\hat{x}))$, are the most commonly used nonlinear activation functions. A simplified tanh (or sigmoid) function, referred to as tanh-like (or, correspondingly, sigmoid-like), is often used in practice, especially in hardware implementations, to reduce the complexity of the forward phase of the network computation [3–7]. The tanh-like function is monotonically increasing on a finite interval (a, b); this interval is the unsaturation or active region of the function. If the activation value, $\hat{x}$, is less than or equal to a, the function is saturated and outputs -1 (or zero), while if the input is greater than or equal to b, the function is also saturated and outputs $+1$.

Some examples of the simplified activation functions are the lookup table (LUT) [6], an adaptive polynomial (AP) [7], a second-order polynomial [3, 4], and an adaptive Catmull-Rom spline (CRS) [8, 9]. Each of these functions comes with its own drawback. The LUT not only needs a large memory space for storing its outputs, it also quantizes the activation function. The quantization errors can reduce the network performance.

The CRS only needs to store its adjustable controlling points so that its memory requirement is less than that of the LUT. Nevertheless, the network capability depends on the size of the table which must be chosen carefully. An AP neural network could have problems due to local minima and numerical instability, especially in high-degree polynomials. Even though a second-order polynomial requires less amount of computational time and realization components than other methods, the training process either converges slowly or fails to converge.

1.1 Motivation

The motivation regarding to simplified activation function can be drawn from the following example. We want to compare networks with some of the existing simplified activation functions [3, 4] to a trained network with the actual tanh as the activation function. Fig. 1.1 shows the decision boundaries in the two-spiral problem of four different two-layered feedforward networks. Each network has two nodes in the input layer, 30 nodes in the hidden layer, and one node in the output layer. This is a very compact structure for performing the two-spiral classification task [10]. To demonstrate the effect of the activation functions, every network uses the same weight vector and uses tanh as the output activation function. The only difference is in the activation functions in the hidden layer. Fig. 1.1(a) shows the decision boundary of the network with tanh in the hidden layer. Figures 1.1 (b), (c), and (d) show the decision boundaries of the networks with polynomials of degrees two, five, and eleven, respectively, as the hidden layer activation function. We can see that only the network shown in Fig. 1.1(a) yields

the correct classification results. Furthermore, it is clear that the decision boundaries are totally different, even though the activation functions differ only slightly (see Fig. 3.2).

From the previous example, we see that we cannot train a network with a particular activation function and implement the network with a simplified function. The logical procedure is to train a network with neurons equipped with simplified activation functions. Nevertheless, from our observations, training a network with tanh converges faster than the network with the same structure but having tanh-like activation function in the hidden layer. The reason is that, during training, some of the activation values fall beyond the unsaturation regions. In the saturation regions, the derivative of the tanh-like function is zero so that the weights are not changed. In the original sigmoid and tanh activation functions, their derivatives in the saturation regions are small, but nonzero, so that even when learning is slow, it is not halted. Nevertheless, this phenomenon of "false local minima" [11] or "error saturation" [12], can be addressed by decreasing the slope of the activation function in order to extend the active input region [11], or by adding an error function in order to increase the gradient value [12, 13]. We propose another approach so that the activation value will be decayed if it is outside the active region.

1.2 Objectives

In this dissertation, our objectives are as follows:

1. To develop a class of low computational complexity sigmoid-like activation func-

tions to replace the logistic sigmoidal function and the hyperbolic tangent function of the feedforward multilayer perceptron;

2. To propose a training algorithm for enhancing the generalization capability as well as the rate of convergent training of a network based on the proposed activation function.

1.3 Scope of Work

An investigation was constrained on a three-layered feedforward network. The cause and effect of the deterioration of the existing training algorithm due to the use of sigmoid-like activation function have been solved. The network with the proposed activation functions have been applied to the classification benchmark problems. The following classification problems have been considered: the Sonar [14], the encoder60 and encoder120 [15], the higher order parity problem (8-bit [10] and 9-bit [16] parity) and two-spiral [14] problems. The data sets corresponding to these benchmarks, but the encoder60 and encoder120, are publicly available from the CMU Repository of Neural Network Benchmarks at http://www.boltz.cs.cmu.edu.

1.4 Contributions of the Dissertation

In this dissertation we instantiate the solutions of the above objectives in a novel method for a neuron construction that using the recursion process of a parametric piecewise polynomial.

The proposed functions have several interesting features:

- they are easily to compute;
- they retain the squashing property of the sigmoid; and
- they are easy to implement both in hardware and in software.

And we proposed a modified error function to sense the degree of saturation of the hidden layer's neuron. Furthermore, the proposed construction process can be extended to construct a class of low complexity RBF-like neuron efficiently.

1.5 Situating the Work Relative to Existing Related Works

The proposed functions are built upon a p-recursive piecewise polynomial. The key contribution of the proposed neuron is its low complicated tanh-like function. Typical use of the tanh-like function replaces the higher complicated tanh or sigmoidal function. The recursive polynomial-based neural networks (RPPNNs) are designed by using a neuron called the p-recursive piecewise polynomial (p-RPP) sigmoid-like function. The basic network scheme is, therefore, similar to the classical multilayer perceptrons (MLP) structures, but with simplified nonlinear sigmoidal activation functions. The weight adaptation algorithms are based on iRprop$^+$ [17] or SARPROP [18] algorithms with the inclusion of an additional annealing anti-Hebbian rule [19] showing up from the formulation of the training with the sum-squared post synaptic of the hidden layer.

Since the active region of the function can be extended by a specific parameter corresponding to the degree of the polynomial while the slope of function is slightly altered, a proper p-RPP neuron of a training network can be selected to fit with the task. Regardless to the activation function used, the novel modified error function shows

that the anti-Hebbian is a source of aid to help the weight adjustment procedure escapes from the error saturation situation. According to the use of the derived anti-Hebbian rule combining with the existing weight decay term as the penalty function, not only the convergent rate and the success training rate but the generalization of the trained network are also significantly increased.

1.6 Dissertation Organization

The rest of the dissertation is organized into five additional chapters. Chapter 2 introduces the sigmoid-like functions. Chapter 3 presents the p-recursive piecewise polynomial neurons. Chapter 4 introduces the p-recursive piecewise polynomial networks and training algorithms as well as the experimental results. Chapter 5 generalizes the modified error term in Chapter 4 and experimentally shows the improvement of the training algorithms. Chapter 6 discusses and concludes the dissertation.

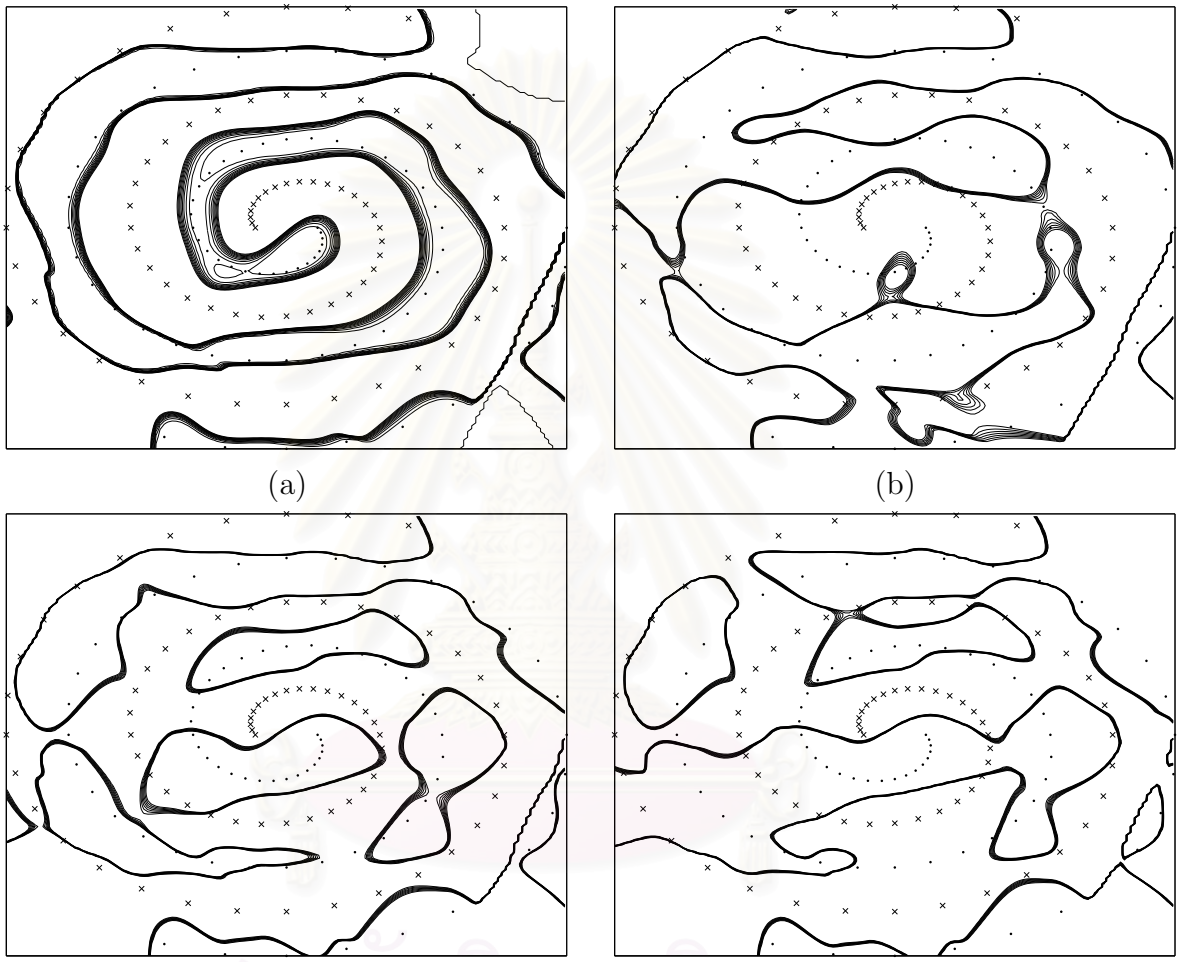

Figure 1.1: Panel (a) shows the decision regions of a network using hyperbolic tangent function as the activation function of the hidden and output layer. Panels (b), (c), and (d) show the decision regions of the network using the same weights as the network in (a) while their activation functions in the hidden layer are replaced by degrees 2, 5, and 11 polynomial $tanh - like$ function (The detail of each function is in Chapter 3.), respectively.

CHAPTER II

Neuron and Activation Function

In this chapter, the mathematical model of neuron will be presented. Some of sigmoid-like activation functions constructed from a look-up table and a polynomial degrees two will be discussed. The methods related to the convergent rate enhancement and generalization enhancement are shifted to the related chapter.

2.1 A Neuron

In neural networks, computational models or nodes or neurons are connected through weights that are adapted during its training to improve the performance. The simplest node provides a linear combination of N weights $w_1, \ldots, w_N$ and N inputs $x_1, \ldots, x_N$, and passes the result through a nonlinearity f, as shown in Figure 2.1.

In our work, the term *neuron* will refer to the McCulloch and Pitts neural model [20] which is an operator performing the mapping

$$\text{Neuron} : \Re^{N+1} \rightarrow \Re \tag{2.1}$$

as shown in Figure 2.1. The activation value $\hat{x}$ is is computed by

$$\hat{x} = \sum_{i=1}^{N} w_i x_i + w_0 \tag{2.2}$$

where the input vectors of the neuron is given by $x_1, \ldots, x_N$. Whereas $\mathbf{w} = w_0, w_1, \ldots, w_N$ is referred to as the weight vector of a neuron. The weights w_0 is the weight which corresponds to the bias input, which is typically set to unity. The output of neuron y is computed by

$$y = f(\hat{x}) \tag{2.3}$$

The activation function f is a mapping. The hard-limiter Heaviside (step) function was frequently used in the first implementations of neural networks, due to its simplicity. It is given by

$$H(x) = \begin{cases} 0, & x \leq \zeta, \\ 1, & x > \zeta, \end{cases} \tag{2.4}$$

where ζ is some threshold. However, this one is not differentiable means that the gradient-based training procedure can not be applied.

The S-shaped function f is a mapping from $\Re$ to $(0, 1)$ and is monotone and continuous. The hyperbolic tangent, tanh(.), and the logistic, 1/(1+exp(-x)) are the most commonly used. Their graphs and derivatives are depicted in Figure 2.2. However, tanh

can be computed from the logistic function by

$$\tanh(x) = \frac{2}{(1 + exp(-x))} - 1, \tag{2.5}$$

which is more simpler.

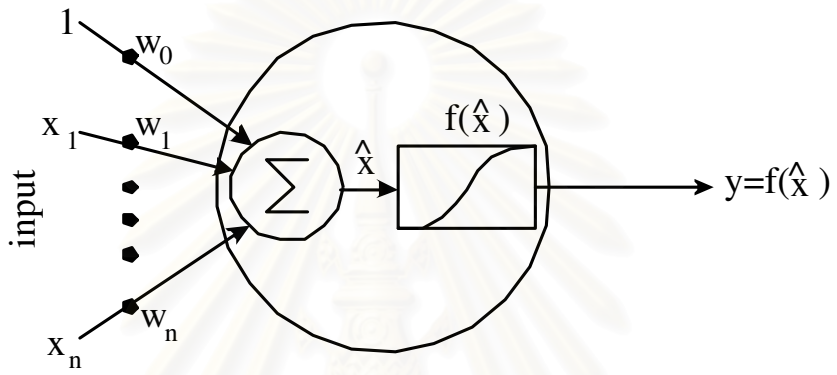

Figure 2.1: Connection within a node

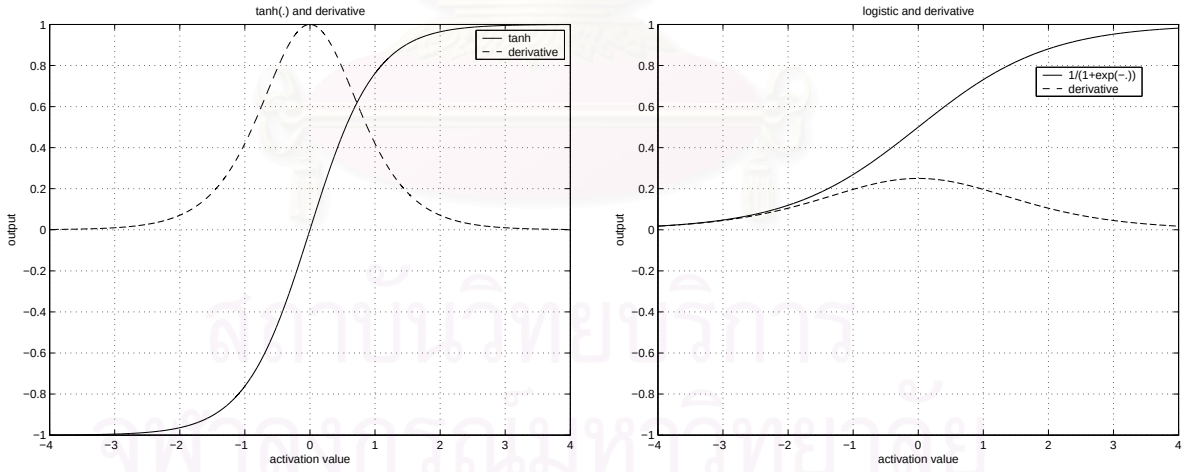

Figure 2.2: Sigmoidal functions. (a) Tanh. (b) Logistic.

2.1.1 Sigmoid-Like Activation Function

In our work, the term *sigmoid-like* will refer to an approximating of tanh or logistic function. A simple method of sigmoid-like implementation is a look-up table (LUT). For neurons based upon look-up table (LUT), samples of a chosen activation function are put into a ROM or RAM to store in the desired activation function. The size of table depends on the precision of sampling. A LUT based neuron is depicted in Figure 2.3.

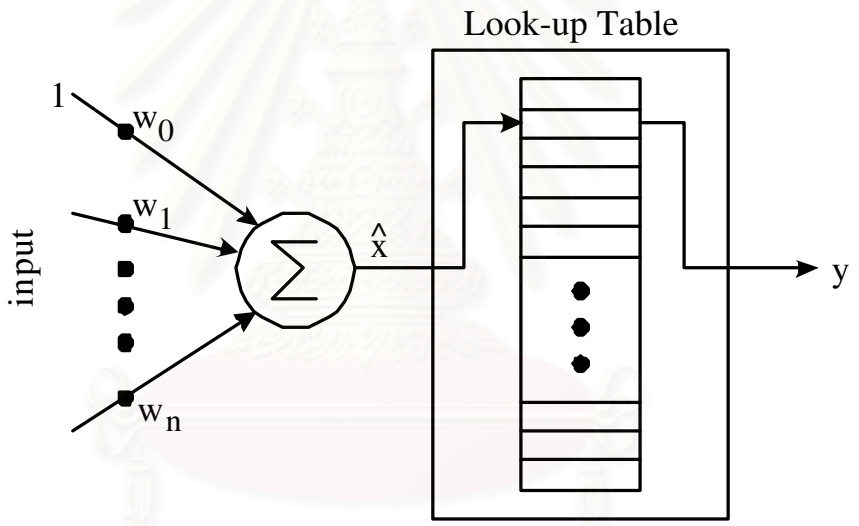

Figure 2.3: LUT.

Alternatively, simplified activation functions are used to approximate the chosen activation function which consume less processor time and memory. Thus, for instance, the sigmoidal curve can be presented by a polynomial function of degrees two, due to its simplicity of coefficient selections and small number of nonlinear curvature segments. The domain of function is divided into two parts. Each part is represented by a second-order function. The sigmoidal-curve is, then, completed by joining both parts which

includes a smooth sigmoidal-curve.

There is an intensive reviewing of neuron constructions in chapter 4 of [21]. An example of a second-order function is described in the Section 2.1.2.

2.1.2 Second-Order Activation Function

We review from a simple second-order nonlinear patch curve which has a tanh-like transition between the lower and the upper saturation regions given by [3]:

$$g(u) = \begin{cases} 1, & for \quad u > 1 \\ u(\beta - \theta u), & for \quad 0 \leq u \leq 1 \\ u(\beta + \theta u), & for \quad -1 \leq u < 0 \\ -1, & for \quad u < -1 \end{cases} \tag{2.6}$$

$$g'(u) = \begin{cases} \beta - 2\theta u, & for \quad 0 \leq u \leq 1 \\ \beta + 2\theta u, & for \quad -1 \leq u < 0 \\ 0, & otherwise \end{cases} \tag{2.7}$$

The shape of the sigmoid-like curve are determined by β and θ. The nonlinear region is in the interval of $u \in$[-1, 1]. Hence, a sigmoid-like symmetric function $F(\hat{x})$ and the derivative $F'(\hat{x})$ can be defined by

$$F(\hat{x}) = g(\frac{\hat{x}}{L}) \tag{2.8}$$

and

$$F'(\hat{x}) \quad = \quad \frac{1}{L} g'(\frac{\hat{x}}{L}) \tag{2.9}$$

where L represents the threshold on x-axis. The non-symmetric version of these functions are $f(\hat{x}) = \frac{1}{2}F(\hat{x}) + \frac{1}{2}$ and $f'(\hat{x}) = \frac{1}{2}F'(\hat{x})$, respectively.

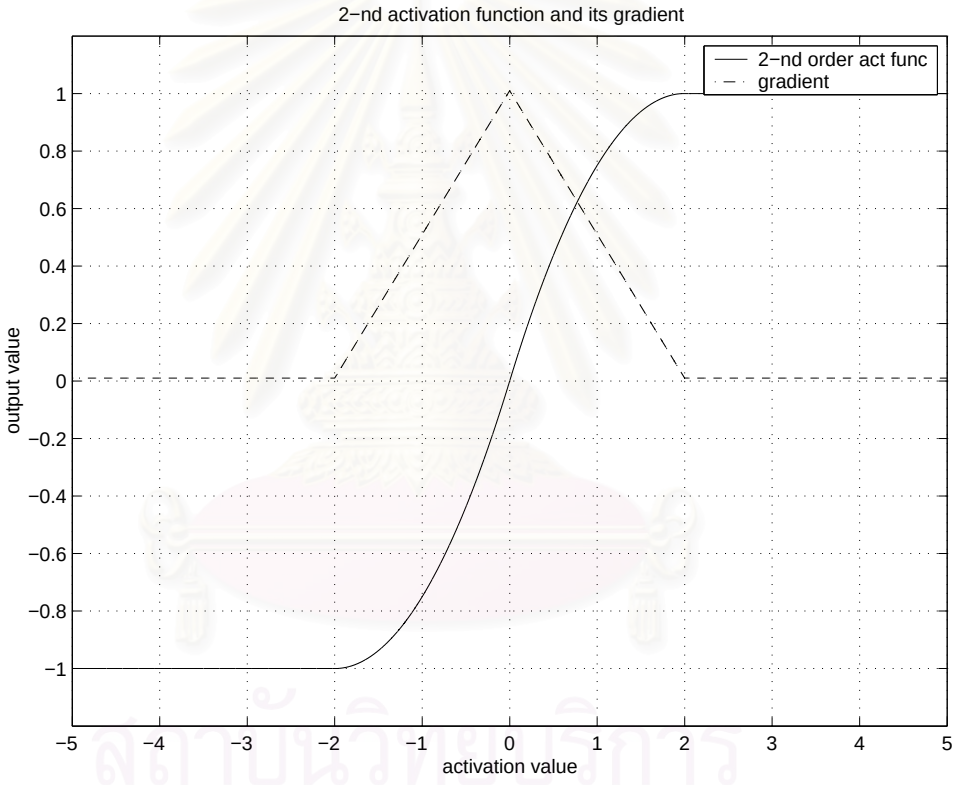

Figure 2.4: The second-order sigmoid-like activation function and the gradient with respect to the activation value. The parameters L, α, and β are 2, 1, and 2, respectively.

Since this function needs a very simple arithmetic operation, the computational complexity is very low. Its shape is adjustable by changing the value of L whose value is the half of the constructed nonlinear region. The graph of this second-order sigmoid-like function and its derivative are shown in Figure 2.4.

In the next chapter, we will propose a systematic construction of neuron. The following properties of function will be retained.

1. *Computational simplicity:* The design must be simple in a computational sense, given its cost/speed implications. In this sense, the use of an interpolation fundamentally based on a polynomial function provides a very appropriate framework.

2. *Programmability:* The generating scheme must allow for the modification of the parameters that define the shape of the sigmoid-like function.

CHAPTER III

The p-Recursive Piecewise Polynomial Neurons

In this chapter, we firstly describe a class of neurons that uses a parametric piecewise polynomial as activation functions. We describe algorithms for generating the new activation functions as well as their derivatives in Section 3.1. The reason for using a simplified activation function is to reduce the amount of computation for the nonlinearity in a neuron. Accordingly, the comparisons of their execution times with other implementations of the nonlinearities are in Section 3.3.

3.1 Neuron Construction

The p-recursive piecewise polynomial (p-RPP) neuron is a derivative of the piecewise second–order polynomials proposed by Kwan [3] and Zhang [4]. The second–order tanh-like approximation can be implemented by dividing the active input segment $(-2, 2)$ as $(-2, 0)$ and $[0, 2)$. When the input goes beyond these two segments, the function clamps the output to values of -1 or 1. The piecewise second-order polynomial

function is implemented as follows:

$$h(\hat{x}) = \begin{cases} -1, & \hat{x} \leq -2, \\ -1 + (1 + 2^{-1} \times \hat{x})^2, & -2 < \hat{x} < 0, \\ +1 - (1 - 2^{-1} \times \hat{x})^2, & 0 \leq \hat{x} < 2, \\ +1, & \hat{x} \geq 2. \end{cases} \tag{3.1}$$

Before we proceed to the discussion on the p-RPP neurons, we consider how well a polynomial can be made to approximate the tanh function. In the range $\hat{x}$ greater than or equal to zero, from Eq. (3.1), the second-order polynomial approximates $\tanh(\hat{x})$ by

$$h(\hat{x})|_{\hat{x}\geq 0} = \begin{cases} +1 - (1 - 2^{-1} \times \hat{x})^2, & 0 \leq \hat{x} < 2, \\ +1, & \hat{x} \geq 2. \end{cases} \tag{3.2}$$

Equation (3.2) can be generalized by the following general piecewise polynomial model approximating tanh.

$$h(\hat{x})|_{\hat{x}\geq 0} = \begin{cases} +1 - (1 - 2^{-(p+1)} \times \hat{x})^n, & 0 \leq \hat{x} < 2^{p+1}, \\ +1, & \hat{x} \geq 2^{p+1}, \end{cases} \tag{3.3}$$

where p and n are the parameters. In order to look for their relationship, we compare the parametric model with $p = 0,\ 1,\ 2,$ and 3, and with $n = 2,\ 3,\ \cdots,\ 32$. Fig. 3.1 shows the graphs of the piecewise polynomials when $n = 2$, 5, 11, and 23 for $p = 0$, 1, 2, and 3, respectively. We can see that these curves are very close to the curve of tanh(.).

Following from the previous observation, the relationships of p and n are represented

by Eq. (3.4) and the values of the threshold computed from different p are depicted in Table 3.1.

$$n = (3 \times 2^p - 1). \tag{3.4}$$

Table 3.1: Relationship of the parameters p, n, and L_p. L_p is the thresholds on x-axis corresponding to p. If input of function is greater than or equal to L_p, output of function is $+1$. From this data, $n = (3 \times 2^p - 1)$.

p	n	$L_p = 2^{p+1}$
0	2	2
1	5	4
2	11	8
3	23	16

Following from the prevoius discussion, we construct a class of tanh-like functions, G, with the active input segments $(-2^{p+1}, 2^{p+1})$, $p = 0, 1, 2, 3$, as follows:

$$G(\hat{x}, p) = \begin{cases} -1, & \hat{x} \leq -2^{p+1}, \\ -1 + (1 + 2^{-(p+1)} \times \hat{x})^{3 \times 2^p - 1}, & -2^{p+1} < \hat{x} < 0, \\ +1 - (1 - 2^{-(p+1)} \times \hat{x})^{3 \times 2^p - 1}, & 0 \leq \hat{x} < 2^{p+1}, \\ +1, & \hat{x} \geq 2^{p+1}, \end{cases} \tag{3.5}$$

Theorem 1. *Function G is a differentiable function.*

Proof. Let x be an input and $p \in \{0, 1, 2, 3\}$ be the parameter of G, respectively. The domain of G comprises of four segments: $(-\infty, -2^{p+1})$, $(-2^{p+1}, 0)$, $(0, 2^{p+1})$, and $(2^{p+1}, \infty)$, and three connecting points: -2^{p+1}, 0, and 2^{p+1}.

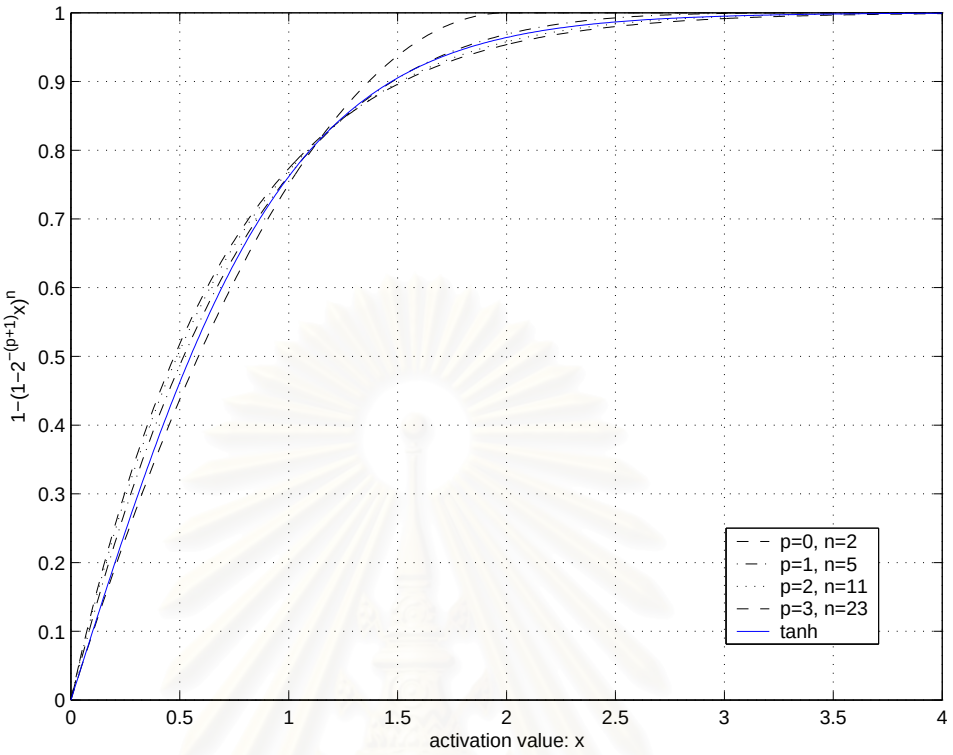

Figure 3.1: The curves of x vs. $1-(1-2^{-(p+1)}\times x)^n$ when $n = 2, 5, 11, 23$ for $p = 0, 1, 2, 3$, respectively. If input of function is greater than or equal to L_p, output of function is $+1$.

Case 1: x is in $(-\infty, -2^{p+1})$, G outputs -1. $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is zero.

Case 2: x is in $(2^{p+1}, \infty)$, G outputs $+1$. $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is zero.

Case 3: x is in $(-2^{p+1}, 0)$, G is a continuous increasing function. $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is $\frac{(3\times 2^p-1)}{2^{p+1}}(1+2^{-(p+1)}\times x)^{3\times 2^p-2}$.

Case 4: x is in $(0, 2^{p+1})$, G is a continuous increasing function. $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is $\frac{(3\times 2^p-1)}{2^{p+1}}(1-2^{-(p+1)}\times x)^{3\times 2^p-2}$.

Case 5: x is -2^{p+1}. If $\Delta t < 0$, $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is zero. If $\Delta t > 0$, $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is zero. So $G'(-2^{p+1}, p)$ is zero.

Case 6: x is 0. If $\Delta t < 0$, $x+\Delta t$ is in $(-2^{p+1}, 0)$. x is in $[0, 2^{p+1})$. $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is $\frac{(3\times 2^p-1)}{2^{p+1}}$. If $\Delta t > 0$, $x+\Delta t$ is in $(-2^{p+1}, 0)$. x is in $[0, 2^{p+1})$. $\lim_{\Delta t\to 0} \frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$

is $\frac{(3\times 2^p-1)}{2^{p+1}}$. So, $G'(0,p)$ is $\frac{(3\times 2^p-1)}{2^{p+1}}$.

Case 7: x is 2^{p+1}. If $\Delta t < 0$, the $\lim_{\Delta t\to 0}\frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is zero. If $\Delta t > 0$, $\lim_{\Delta t\to 0}\frac{G'(x+\Delta t,p)-G'(x,p)}{\Delta t}$ is zero. So $G'(2^{p+1},p)$ is zero.

Hence, from all cases, G is differentiable. □

Since G is differentiable, from Theorem 1, differentiating G with respect to the input variable $\hat{x}$ can be computed by

$$G'(\hat{x},p)=\begin{cases}0, & \hat{x}\le -2^{p+1},\\ \frac{(3\times 2^p-1)}{2^{p+1}}(1+2^{p+1}\times\hat{x})^{3\times 2^p-2}, & -2^{p+1}<\hat{x}<0,\\ \frac{(3\times 2^p-1)}{2^{p+1}}(1-2^{p+1}\times\hat{x})^{3\times 2^p-2}, & 0\le\hat{x}<2^{p+1},\\ 0, & \hat{x}\ge 2^{p+1},\end{cases} \tag{3.6}$$

3.2 Fast p-RPPs

With a fixed value of p, the number of multiplication operations in the realization of G depends on the degree of $(1+2^{-(p+1)}\times\hat{x})$, which is $3\times 2^p-1$ and the number of multiplication operations in the realization of G' also depends on the degree of $(1+2^{-(p+1)}\times\hat{x})$, which is $3\times 2^p-2$.

Let $\phi(\cdot,\cdot)$ be the basis function defined by

$$\phi(\hat{x},q)=\begin{cases}0, & \hat{x}\le -2^{q+1},\\ (1+2^{-(q+1)}\times\hat{x}), & -2^{q+1}<\hat{x}<0,\\ (1-2^{-(q+1)}\times\hat{x}), & 0\le\hat{x}<2^{q+1},\\ 0, & \hat{x}\ge 2^{q+1},\end{cases} \tag{3.7}$$

where q is an integer 0, 1, 2, or 3. We can see that both G and G' can be rewritten in terms of ϕ as follows.

$$G(\hat{x},p) = \begin{cases} -1+\phi(\hat{x},p)^{3\times 2^p-1}, & \hat{x}<0, \\ +1-\phi(\hat{x},p)^{3\times 2^p-1}, & \hat{x}\geq 0. \end{cases} \tag{3.8}$$

$$\begin{aligned} G'(\hat{x},p) &= \frac{(3\times 2^p-1)}{2^{p+1}}\phi(\hat{x},p)^{3\times 2^p-2}, \\ &= (\frac{3}{2}-\frac{1}{2^{p+1}})\phi(\hat{x},p)^{3\times 2^p-2}. \end{aligned} \tag{3.9}$$

A systematic procedure of raising ϕ to the power of $3\times 2^p-1$ can be done in two steps. First, transforming the term $\phi(\hat{x},p)^{3\times 2^p-1}$ to an equivalent recursion $g(\hat{x},p,n)$ is given by:

$$g(\hat{x},p,n) = \begin{cases} \phi(\hat{x},p)\times\phi(\hat{x},p), & n=0, \\ \phi(\hat{x},p)\times g(\hat{x},p,n-1)\times g(\hat{x},p,n-1), & n>0, \end{cases} \tag{3.10}$$

where n is an integer ranges from zero to p. Therefore, the new form of G is as follows:

$$G(\hat{x},p) = \begin{cases} -1+g(\hat{x},p,n), & \hat{x}<0, \\ +1-g(\hat{x},p,n), & \hat{x}\geq 0. \end{cases} \tag{3.11}$$

where $n=p$.

Second, transforming the recursion of $g(\hat{x}, p, n)$ to the loop invariance, which is simpler. The pseudocode of this computation of G is shown in Algorithm 1.

Algorithm 1 Compute $G(\hat{x}, p)$

Require: An activation value $\hat{x}$, an integer $p \in \{0, 1, 2, 3\}$
Ensure: $G(\hat{x}, p)$
if $\hat{x} < 0$ **then**
  $sgn := -1$
  $\phi := 1 + \hat{x}/2^{p+1}$
else
  $sgn := +1$
  $\phi := 1 - \hat{x}/2^{p+1}$
end if
if $\phi \leq 0$ **then**
  $G := sgn$
else
  $g := \phi \times \phi$
  for $n := 1$ to p **do** {loop invariant: $g := \phi(\hat{x}, p) \times g \times g = \phi(\hat{x}, p)^{3 \times 2^n - 1}$}
    $g := \phi \times g \times g$
  end for
  $G := sgn - sgn \times g$
end if

The same procedure can be applied directly to G' as follows:

$$\begin{aligned} G'(\hat{x}, p) &= \frac{(3 \times 2^p - 1)}{2^{p+1}} \phi(\hat{x}, p)^{3 \times 2^p - 2}, \\ &= (\frac{3}{2} - \frac{1}{2^{p+1}}) \phi(\hat{x}, p)^{3 \times 2^p - 2}, \\ &= (\frac{3}{2} - \frac{1}{2^{p+1}}) g'(\hat{x}, p, n), \quad n = p, \end{aligned} \tag{3.12}$$

where

$$g'(\hat{x}, p, n) = \begin{cases} \phi(\hat{x}, p), & n = 0, \\ (\phi(\hat{x}, p) \times g'(\hat{x}, p, n-1))^2, & n > 0. \end{cases} \tag{3.13}$$

The term $g'(\hat{x}, p, n)$ can be also transformed to a simpler loop.

In the backward computation of the training procedure, both G and its derivative, G', are required. The pseudocode for computing both G and G' is shown in Algorithm 2. Algorithm 1 is a direct implementation of G. Algorithm 2 is not only a direct

Algorithm 2 Compute both $G'(\hat{x}, p)$ and $G(\hat{x}, p)$

Require: An activation value $\hat{x}$, an integer $p \in \{0, 1, 2, 3\}$
Ensure: $G'(\hat{x}, p)$ $G(\hat{x}, p)$
if $\hat{x} < 0$ **then**
  $sgn := -1$
  $\phi := 1 + \hat{x}/2^{p+1}$
else
  $sgn := +1$
  $\phi := 1 - \hat{x}/2^{p+1}$
end if
if $\phi \leq 0$ **then**
  $G' := 0$, $G := sgn$
else
  $g' := \phi$
  for $n := 1$ to p **do** {loop invariant: $g' := (\phi(\hat{x}, p) \times g')^2$}
    $g' := (\phi \times g')^2$
  end for
  $G' := (\frac{3}{2} - \frac{1}{2^{p+1}}) \times g'$, $G := sgn - sgn \times g' \times \phi$
end if

implementation of G' but is also an indirect implementation of G. The difference between the two algorithms is in how the power of ϕ is raised to $3 \times 2^p - 1$. The basis function is the second-order polynomial: ϕ^2; consequently, there is no loop to perform. If a polynomial of a higher-order is required and is controlled by p, then the polynomial of that order will be constructed by the loop invariance. We can see from both algorithms that p represents the number of loops, and that there are two multiplication operations in each loop. In Algorithm 1, the initial degree for g, which is a variable representing ϕ, is 2. Suppose that the degree of g before the loop begins is n, it will be $2n+1$ when each

iteration of the loop is performed. After the pth iteration has been done, the degree of g or, equivalently, the degree of ϕ will be $3 \times 2^p - 1$. Whereas, in Algorithm 2, the initial degree for g', which is also a variable representing ϕ, is 1. The degree of g' is added by one and then is doubled per iteration of the loop. Once the loop is finished, the degree of g' is $3 \times 2^p - 2$: one order fewer than that of g. Hence, in this case, G needs one more multiplication of ϕ than for G'.

The graphs of the p-RPPs are shown in Fig. 3.2. Their differences from tanh are shown in Table 3.2. We can see that when p is 1, the active region is $(-4, 4)$, and the differences between the 1-RPP and the tanh is minimum. When the input is 2.0 (or, correspondingly, -2.0), the 0-, 1-, 2-, and 3-RPPs output 1.0 (-1.0), 0.9688 (-0.9688), 0.9578 (-0.9578), 0.9536 (-0.9536), respectively. Each of which is an extreme or a nearly extreme value. Accordingly, the segment of $(-2, 2)$ can be considered as the common active input segment for all the p-RPP neurons.

The number of multiplications of raising ϕ to the power of $3 \times 2^p - 1$ are reduced from $3 \times 2^p - 2$ to $2p + 1$. Since the denominator, 2^{p+1}, is a power-of-two number, the division operation can be simplified to an equivalent bit-shifting operation, or can be simplified to a hashing table operation. The value from the hashing table is then used to multiply with the dividend to obtain the result. These evidences reveal the p-RPP neuron realizations to be simple and yet to be approximate to tanh function.

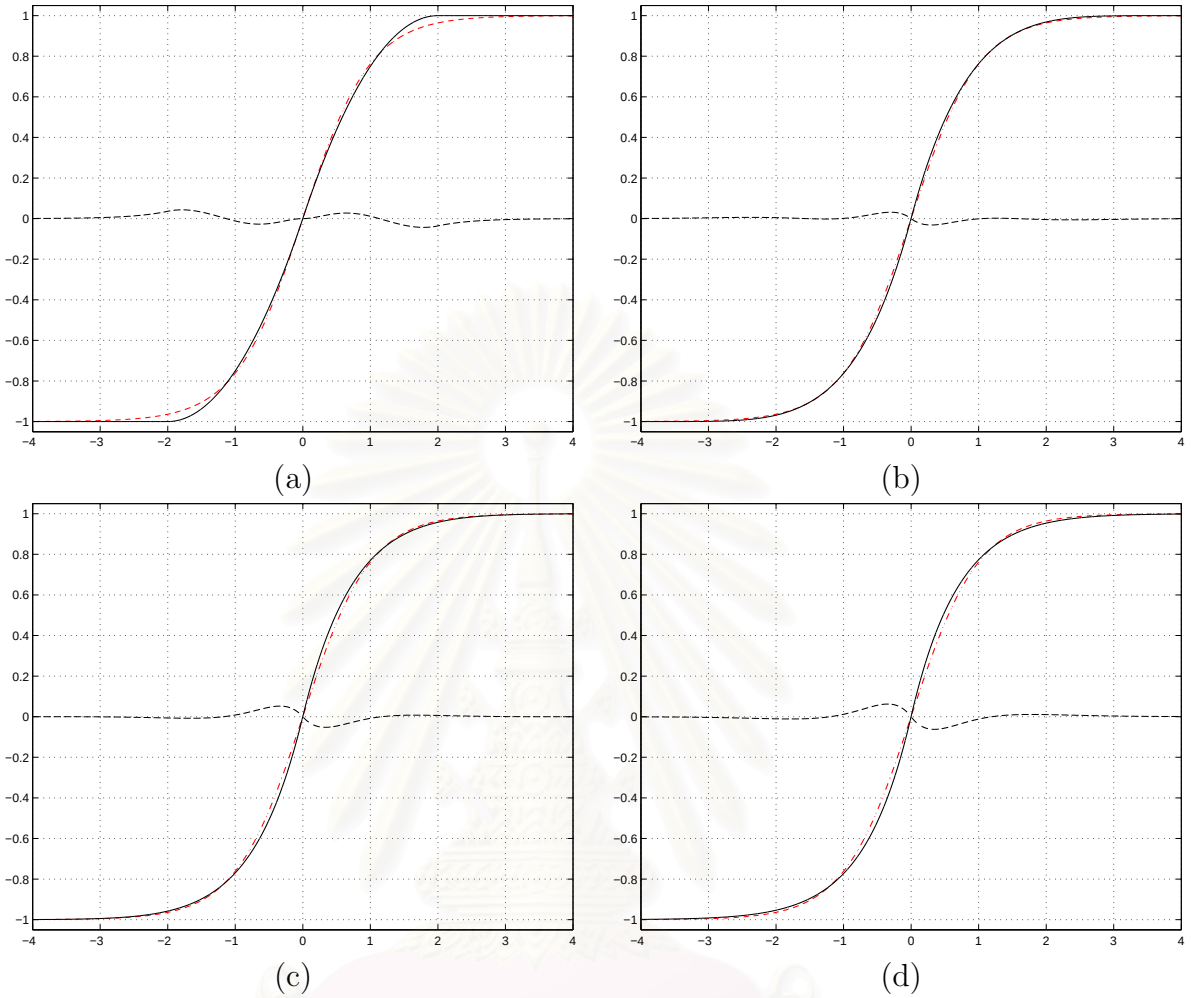

Figure 3.2: Graphs of p-RPPs, tanh and their differences. (a) 0-RPP. (b) 1-RPP. (c) 2-RPP. (d) 3-RPP. The thick lines represent p-RPP, the dash-dot lines represent tanh, and the dash-dash lines represent the differences between p-RPP and tanh.

3.3 Radial Basis Function-Like

From the above construction, we can also construct a class of the p-recursive piecewise polynomial radial basis function (p-RPPRBF) as radial basis function-like (RBF-like) as follows:

$$\mathrm{RPPRBF}(d^2, p) = \phi(d^2, p)^{3\times 2^p - 1}, \tag{3.14}$$

Table 3.2: Differences between p-RPPs and tanh. x_0, x_1 and x_2 are the points on the x-axis when p-RPPs equal to tanh. If input is greater than or equal to zero, 0-RPP has two points of that kind, whereas 1-, 2- and 3-RPPs each of which has three points. L_p is the threshold on the x-axis. If input of function is greater than or equal to L_p, output of function is +1. The differences are the integration of the difference between p-RPPs and tanh when input is in the interval of $[x_0,x_1]$, $[x_1,x_2]$, $[x_2,L_p]$, and $[L_p,16]$.

p	x_0	x_1	x_2	$L_p = 2^{p+1}$	Difference
0	0	1.3262577500	–	2	0.05875
1	0	1.0508487975	1.54909325	4	0.02771
2	0	1.1721823970	3.45095900	8	0.04298
3	0	1.1979472750	7.12802350	16	0.05809

where d is a difference of the center of RBF and input data. The pseudocode of this computation of RPPRBF is shown in Algorithm 3. The graphs of 0-, 1-, 2-, and 3-

Algorithm 3 Compute RPPRBF$(\hat{d}, p) := \phi(\hat{d}, p)^{3\times 2^p - 1}$

Require: A distance-squared $\hat{d} \geq 0$, an integer $p \in \{0, 1, 2, 3\}$
Ensure: RPPRBF$(\hat{d}, p) := \phi(\hat{d}, p)^{3\times 2^p - 1}$
1: $\phi := 1 - \hat{d}/2^{p+1}$
2: **if** $\phi \leq 0$ **then**
3: RPPRBF := 0
4: **else**
5: $g := \phi \times \phi$
6: **for** $n := 1$ to p **do** {loop invariant: $g := \phi(\hat{d}, p) \times g \times g = \phi(\hat{d}, p)^{3\times 2^n - 1}$}
7: $g := \phi \times g \times g$
8: **end for**
9: RPPRBF := g
10: **end if**

RPPRBFs, and $exp(-(d^2)/0.65)$ are shown in Fig 3.3. We can see that the curves of 1-, 2- and 3-RPPRBFs are closable to RBF.

3.4 Execution time of p-RPPs

In the previous section, we introduced a new class of neurons with p-RPP activation functions. The motivation for the new class of neurons is in improving computation

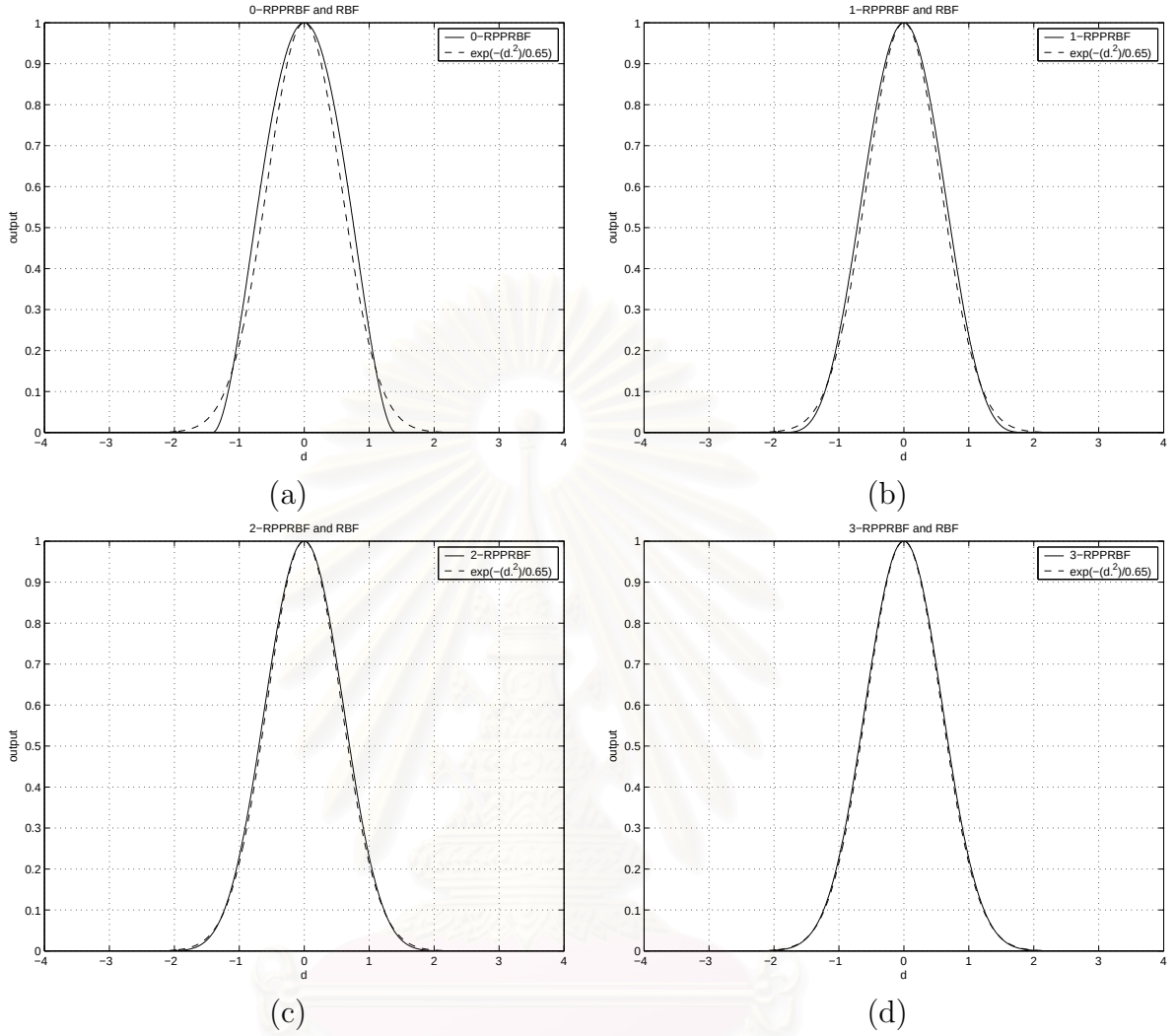

Figure 3.3: Graphs of RPPRBFs, and $exp(-(d^2)/0.65)$. (a) is for RPPRBF(d^2,0), (b) is for RPPRBF(d^2,1), (c) is for RPPRBF(d^2,2), and (d) is for RPPRBF(d^2,3), respectively. The thick lines represent p-RPPRBFs, the dash-dash lines represent $exp(-(d^2)/0.65)$.

efficiency. We therefore compare the execution times of the p-RPP neurons (Algorithm 1) with a neuron with the tanh-function as the activation function with the execution times of the other methods of implementing an activation function, viz. LUT, sigmoid: $2/(1+\exp(-\hat{x}))-1$, and tanh. The p-RPPs test is shown in Table 3.3. The measurements are the execution time of the code generated by three compilers: MSVC++ version 6.0 [29], MinGW32 version 3.2 [30], and Lcc-Win32 version 3.3 [31]. Each function

computes the outputs of 2×10^8 input values generated uniform randomly over the active input segment. The measured times in Table 3.3 depend on the compiler, with MSVC++ yielding the shortest time and Lcc-Win32 the longest time. Both p-RPP and LUT method are faster than either sigmoid or tanh. The p-RPPs of MinGW32 are faster than LUT for p set to zero to three. The p-RPP of MSVC++ are faster than LUT for p less than three. The p-RPP of Lcc-Win32 is faster than LUT for p equal to one.

Table 3.3: Time in seconds required to perform 2×10^8 activation operations. Single and double represent single precision and double precision floating points, respectively. $\text{Time}_{MSVC++6.0}$, $\text{Time}_{MinGW32}$, and $\text{Time}_{Lcc-Win32}$ are measured from the codes generated by three compilers: MSVC++6.0 [29], MinGW32 [30], Lcc-Win32 [31], respectively. The experiments are performed on the Windows XP PC with one Intel Pentium III 1.06 GHz CPU, 640 MB RAM.

p of RPP	$\text{Time}_{MSVC++6.0}$		$\text{Time}_{MinGW32}$		$\text{Time}_{Lcc-Win32}$	
	single	double	single	double	single	double
0	1.409	1.782	2.771	2.616	2.676	3.494
1	1.288	2.058	2.953	2.768	3.089	4.213
2	1.414	2.432	3.140	2.899	3.476	4.900
3	1.576	2.828	3.314	3.256	3.984	5.646
LUT	2.440	2.446	3.719	3.526	3.627	3.573
sigmoid	3.593	4.009	8.294	9.027	5.263	8.637
tanh	8.568	8.812	10.344	10.439	10.262	10.009

3.5 Summary

In this chapter, a systematic construction of adjustable active input segment sigmoid generators was shown. We not only generalized the p-RPP neuron model to a class of tanh-like functions having the interval of $(-2, 2)$ as their common active segment, but also proposed the p-RPPRBFs as a low complexity RBF-like. The shape and complexity of functions are controlled by a pameter p. The execution times of p-RPP neurons are

shorter than those of both sigmoid and tanh and yet comparable to the lookup table scheme.

CHAPTER IV

The p-Recursive Piecewise Polynomial Network's Training

A p-RPP network consists of the p-RPP neurons as described in the previous chapter, arranged as layers with full interconnection between successive layers. The weights are determined by a learning algorithm, such as backpropagation, which propagates and distributes the errors at a higher layer backwards. In this chapter, we describe our modifications of the error function as well as input normalization those facilitate a more efficient learning process and overcome deficiency due to using p-RPP neurons. The experimental results and conclusion are also included this chapter.

4.1 Training Criteria

Without loss of generality, suppose the network has three layers of nodes in which the lth layer has $N^{(l)}$ nodes. According to the input vector $\mathbf{x}$, let the state vector of nodes in layer l be

$$\mathbf{x}^{(l)} = [x_1^{(l)}\, x_2^{(l)}\, \ldots\, x_{N^{(l)}}^{(l)}]^T,$$

where $\mathbf{x}^{(0)}$ is the input, and $\mathbf{x}^{(2)}$ is the output vector. Here, $x_j^{(l)} (l \neq 0)$ takes on a value between -1 and 1. Let the desired output vector, or target, corresponding to a training

pattern $\mathbf{x}$ be $\mathbf{t} = [t_1\, t_2 \ldots t_{N^{(2)}}]^T$.

When $\mathbf{x}$ is presented to the network with a current weight vector $\mathbf{w}$ and is propagated forward to determine the output vector, the state $x_j^{(l)}$ of neuron j in each lth layer is computed by

$$x_j^{(l)} = G(\hat{x}_j^{(l)}, p), \quad l = 1, 2, \tag{4.1}$$

and

$$\begin{aligned} \hat{x}_j^{(l)} &= w_{j0}^{(l)} + \sum_{i=1}^{N^{(l-1)}} w_{ji}^{(l)} x_i^{(l-1)} \\ &= \sum_{i=0}^{N^{(l-1)}} w_{ji}^{(l)} x_i^{(l-1)}, \quad \text{where} \quad x_0^{(l-1)} = 1, \end{aligned} \tag{4.2}$$

and where $w_{ji}^{(l)}$ denotes the weight connecting $x_i^{(l-1)}$ to neuron j, $w_{j0}^{(l)}$ denotes the bias weight incoming from a constant $x_0^{(l-1)}$, and $\hat{x}_j^{(l)}$ denotes the activation value of the neuron. Both $w_{ji}^{(l)}$ and $w_{j0}^{(l)}$ are elements of $\mathbf{w}$.

If the error function is $E(\mathbf{w})$, the gradient-descent direction which is used for iteratively optimizing $w_{ji}^{(l)}$ to minimize $E(\mathbf{w})$ at time t is calculated by the error backpropagation (BP) algorithm [22]:

$$\begin{aligned} -\frac{\partial E(\mathbf{w})}{\partial w_{ji}^{(l)}(t)} &= -\frac{\partial E(\mathbf{w})}{\partial \hat{x}_j^{(l)}} \frac{\partial \hat{x}_j^{(l)}}{\partial w_{ji}^{(l)}(t)}, \\ &= \delta_j^{(l)} x_i^{(l-1)}, \end{aligned} \tag{4.3}$$

where $\delta_j^{(l)}$ is the generalized delta term [22] defined by

$$\delta_j^{(l)} = \begin{cases} -G'(\hat{x}_j^{(l)}, p)\frac{\partial E(\mathbf{w})}{\partial x_j^{(l)}}, & l = 2, \\ G'(\hat{x}_j^{(l)}, p)\sum_{k=1}^{N^{(l+1)}} w_{kj}^{(l+1)}(t)\delta_k^{(l+1)}, & l = 1. \end{cases} \tag{4.4}$$

Since the generalized delta term, Eq.(4.4), at each node is multiplied by the sigmoid prime factor [23], $G'(\hat{x}_j^{(l)}, p)$, it is a nullity if $\hat{x}_j^{(l)}$ is outside the input active region. The network with tanh activation function encounters a very similar problem because this factor becomes small when $\hat{x}_j^{(l)}$ is large. Recently, Joost and Schiffmann [23] proposed the combination of two modifications called CEN Optimization (Cross Entropy combined with Pattern Normalization). Instead of the usual quadratic error, they used the cross entropy (CE) [24] as an error function to eliminate the sigmoid prime factor of the updating rule for the output units. Also, the input patterns were normalized in order to balance the dynamic range of the input. Oh [25] proposed a modified error function that can be reduced to the CE. The function is superior to CE in terms of effectively eliminating the uncorrected saturation and preventing overspecialization during training.

At the output layer, the sigmoid derivative can be eliminated by selecting the proper error function. The modified error function proposed by Oh is generally defined as

$$E(\mathbf{w}) = -\sum_{j=1}^{N^{(2)}} \int \frac{t_j^{n+1}(t_j - G(\hat{x}_j^{(2)}, p))^n}{2^{n-2}G'(\hat{x}_j^{(2)}, p)} dx_j^{(2)}, \tag{4.5}$$

where $t_j = \pm 1$ and $n = 1, 2, \cdots$. When $n = 1$ it is the CE function. In our work, we use $n = 2$ per the experimental results from Oh [25]. The generalized delta term used as

the factor for updating the weight w_{ji}^{l} in the iteration t can be computed in a backward manner by

$$\delta_j^{(l)} = \begin{cases} \frac{t_j^{n+1}(t_j - x_j^{(l)})^n}{2^{n-1}}, & l = 2, \\ G'(x_j^{(l)}, p) \sum_{k=1}^{N^{(l+1)}} w_{kj}^{(l+1)}(t) \delta_k^{(l+1)}, & l = 1. \end{cases} \tag{4.6}$$

The sigmoid derivative at the output layer is thus eliminated although it remains in the hidden layer.

Let $\mathbf{x}^s$, $1 \leq s \leq S$, be a member of a training set with S patterns. We calculate the $H_{j,s}$ to sense the distance between $\widehat{x^s}_j^{(l)}$, the activation value of neuron j in layer l due to the input pattern $\mathbf{x}^s$, and the common active input segment $(-2, 2)$. In order to inhibit the growth and to decay the magnitude of the activation value, especially when its value is outside this segment, $H_{j,s}$ can be defined as

$$\begin{aligned} H_{j,s}(\mathbf{w}) &= \left(\frac{w_{j0}^{(l)} + \sum_{i=1}^{N^{(l-1)}} w_{ji}^{(l)} x_i^{s(l-1)}}{2} \right)^2, \\ &= \left(\frac{\widehat{x^s}_j^{(l)}}{2} \right)^2. \end{aligned} \tag{4.7}$$

Differentiating $H_{j,s}(\mathbf{w})$ with respect to the weight $w_{ji}^{(l)}$ is computed by

$$\frac{\partial H_{j,s}(\mathbf{w})}{\partial w_{ji}^{(l)}} = \frac{x_i^{s(l-1)} \widehat{x^s}_j^{(l)}}{2}. \tag{4.8}$$

The effect of Eq.(4.8) is the more $\widehat{x^s}_j^{(l)}$ grows beyond $(-2, 2)$, the more penalty $H_{j,s}$ yields.

To remedy for the deficiency of the sigmoid prime factor in the hidden layer, we modify the error function. The total error function for all S training patterns, $\overline{E}(w)$, is the average of the summation of all $E(\mathbf{w})$ with the $H_{j,s}$ term:

$$\overline{E}(\mathbf{w}) = \frac{1}{S \times N^{(2)}} \left(\sum_{s=1}^{S} E(\mathbf{w}) + \frac{\alpha(t)}{N^{(l=1)}} \sum_{j=1}^{N^{(l=1)}} \sum_{s=1}^{S} H_{j,s}(\mathbf{w}) \right), \tag{4.9}$$

where $0 < \alpha < 1$ usually varies with time.

Finally, from equations (4.3), (4.6), (4.9), and (4.8) the gradient-descent direction $-\partial\overline{E}/\partial w_{ji}^{(l)}(t)$ is computed by

$$-\frac{\partial \overline{E}(\mathbf{w})}{\partial w_{ji}^{(l)}(t)} = \begin{cases} \frac{1}{S \times N^{(2)}} \sum_{s=1}^{S} \delta_j^{(l)} x_i^{s(l-1)}, & l = 2, \\ \frac{1}{S \times N^{(2)}} \left(\sum_{s=1}^{S} \delta_j^{(l)} x_i^{s(l-1)} - \frac{\alpha(t)}{N^{(l)}} \sum_{s=1}^{S} \frac{x_i^{s(l-1)} \widehat{x^s}_j^{(l)}}{2} \right), & l = 1. \end{cases} \tag{4.10}$$

Here, both $\delta_j^{(1)}$ and $\widehat{x^s}_j^{(1)}$ in Eq.(4.10) are multiplied by $x_i^{s(0)}$. If two inputs connect to the same neuron possess highly different dynamic range of their input values, the weights must be adapted in such a way that the influence of that neuron's input is equalized. This process of dynamic range adaptation can be very time consuming when the values of different inputs vary strongly.

The problem described above can be eliminated by the input normalization process [23]. The process is a simple linear transformation of the inputs and yields standardized inputs.

$$x_i^{s(0)} = \frac{x_i^s - \bar{x}_i}{\sigma_i}, \tag{4.11}$$

where x_i^s denotes the i-th component of the s-th input vector $\mathbf{x^s}$; $\bar{x}_i = \frac{1}{S}\sum_{s=1}^{S} x_i^s$ corresponds to the mean and $\sigma_i = \sqrt{\frac{1}{S-1}\sum_{s=1}^{S}(x_i^s - \bar{x}_i^s)^2}$ is the standard deviation of the original input data.

By means of Eq.(4.11), a multimodal distribution is centered around zero and dynamic range will be equalized because of the division by the standard deviations. The term $-\frac{\alpha(t)}{2} x_i^{s(0)} \widehat{x^s}_j^{(1)}$ is a form of the anti-Hebbian rule [19]; essentially, it acts as an unsupervised learning rule for the hidden neurons. When α is not zero, the gradient direction of the hidden layer comes from a combination of the supervised algorithm, viz. backpropagation [22], providing an excitatory force, and the unsupervised algorithm, viz. the anti-Hebbian rule [19], providing an inhibitory force [26]. When the excitatory force is less than the inhibitory force, the neuron reduces its magnitude of activation value. When α approaches zero, the gradient direction is determined by the supervised algorithm.

This allows the supervised algorithm and the unsupervised algorithm to cooperate in the training procedure. In the early phase of training, the unsupervised learning rule can provide supplement information to the supervised algorithm. During the training procedure, the values of the weights are initially small, typically around zero. As the weight values increase with training, the activation value and the derivative of the activation function grow in opposite directions. As the activation value of a neuron due to an input pattern increases, the derivative of the activation function due to this input decreases to zero. If there are some classified output bits having $t_j - x_j^{(2)}$ equal to zero, this bit does not provide any supervised information but it does supply unsupervised information for further weight updating process.

As training progresses, more and more output bits match the corresponding target bits so that the number of output bits providing supervised information are decreased. To reduce the effect of over-inhibiting, α should be annealed over time. In addition, the activation value should be scaled down so that the $H_{j,s}$ is lessened with more hidden units. Based on these observations, $\alpha(t)$ can be defined by

$$\alpha(t) = \beta \frac{C_{t_j \neq x_j^{(2)}}}{C_{total} \times N^{(1)}}, \tag{4.12}$$

where $C_{t_j \neq x_j^{(2)}}$ is the number of output bits such that $t_j \neq x_j^{(2)}$, $C_{total} = S \times N^{(2)}$, and β is a positive constant. In our experiments, setting β to be 0.05 works for every problem.

4.2 Experimental Results

In the previous chapter, we introduced a new class of neurons with p-RPP activation functions. In the previous section of this chapter a new error criteria for training networks with those neurons are proposed. The evaluations of the proposed enhancements (4.10) to the weights updating procedures are presented in this section.

We use two variations of Rprop (Resilient propagation) [27] algorithms: iRprop$^+$ (Improved Rprop with weight-backtracking) [17] and SARPROP (Simulated Annealing and weight decay Rprop) [18]. These algorithms are selected because they have performed well in terms of speed and generalization results. The temperature T of SARPROP is set to 0.01. The remaining parameters of the both algorithms are the default values.

Since the Oh's modified error function can be considered as an enhancement of CE [25], we use the same criterion as Joost and Schiffmann [23] to name the four

possible combinations of algorithms. The cooperations of iRprop$^+$ with and without anti-Hebbian, SARPROP with and without anti-Hebbian rule are named as follows: ECEN-iRprop$^+$ (Enhanced CEN iRprop$^+$), CEN-iRprop$^+$ (CEN iRprop$^+$), ECEN-SARPROP (Enhanced CEN SARPROP), and CEN-SARPROP (CEN SARPROP). ECEN-SARPROP and CEN-SARPROP differs as to what the additional penalty functions are. The former uses the anti-Hebbian while the latter uses the log squared weight decay terms.

Totally, there are sixteen combinations of networks with tanh, 0-, 1-, 2-RPP activation functions and the two original, the two proposed algorithms. They were tested on the training of standard three-layered networks architectures on the Sonar [14], the encoder60 and encoder120 [15], the higher order parity problem (8-bit [10] and 9-bit [16] parity) and a well-known classification benchmark, namely two-spiral [14] problem. The data sets corresponding to these benchmarks, but the encoder60 and encoder120, are publicly available from the CMU Repository of Neural Network Benchmarks at http://www.boltz.cs.cmu.edu. The details of the network architectures for each of these benchmarks and the performance of our algorithms are mentioned in the corresponding subsections.

All simulations were carried out on a Pentium III 1.06 GHz with 640 MB RAM PC. The MATLAB executable *mex* file of each algorithms was complied by MSVC++ 6.0. In all cases 100 training trials were performed (with uniformly random initialization of the weights in $(-0.5, 0.5)$. The maximum number of epochs was set to 20000, except two-spiral problem was set to 100000. Since the algorithms used in the experiments are in the same class, actually, they are the variations of Rprop algorithm. The computational

effort for one weight updating cycle is approximately the same which means that the number of epochs can be compared directly.

The criteria for evaluation and comparison will be:

- the convergence reliability: the success rate,
- number of epoch for a success training: mean, standard deviation (sd), minimum and maximum, and in some cases
- the quality of solution.

The training was considered successful whenever Fahlman's "40-20-40" criterion [28] was satisfied (i.e., values in the lowest 40% of the output range were treated as output -1, and values in the highest of 40% were treated as output $+1$, and values in the middle 20% were treated as indeterminate and therefore were considered as incorrect).

In the following, the experiments on the networks are presented, organized by the data sets used.

4.2.1 The Sonar benchmark

The Sonar benchmark is a very well-known classification problem. The task is to classify reflected sonar signals in two categories (metal cylinders (mines) and rocks). The related data set comprises 208 input vectors, each with 60 components. Recently, it has been pointed out that the problem indeed is linearly separable [32,33]. Despite this fact, Gorman and Sejnowski [14] reported a success rate of only 85% for a single-layered perceptrons, rising to 100% only when 12 hidden nodes are introduced in the feedforward

neural network architecture. A challenging task is to apply the proposed algorithms to the sonar problem using a network without hidden nodes. It has been argued [34] that the solution of this problem without hidden nodes is a difficult task because of the highly nonuniform distribution of data points in the input space. Therefore, conventional algorithms may take very long times to converge and this explains Gorman and Sejnowski's results. Hasenjäger and Ritter [35] showed the experimental results dealing with the generalization and convergence properties of various perceptron learning procedures using the Sonar benchmark. More advanced learning procedures for threshold perceptrons did not outperform the classical perceptron learning rule. It turned out that a continuous perceptron with sigmoidal activation function is not in general guaranteed to find a separating solution. Recently, Ampazis and Perantonis [10] reported that the BFGS [36], Inverse-BFGS [37] and CG/PR [38] methods failed to converge in all trials, and LM [39] is only 7% of trials.

In this dissertation, a network with 60 inputs, one output unit, and one hidden node was used. This network architecture is equivalent to a series of a perceptron in the hidden layer following by a *not* or a $buffer$ logic gate in the output layer. The number of adaptation cycles required by each of the sixteen combinations of algorithms and network architectures for converging to the total classification error 0 as well as the success rate of training, evaluated by averaging the results of 100 trials are shown in Table 4.1.

Both CEN-iRprop$^+$ and CEN-SARPROP algorithms fail in all trials, while ECEN-iRprop$^+$ and ECEN-SARPROP algorithms converge to solution with very high success rates. The network with 0-RPP yields the lowest success rates: 66% from the training by

ECEN-SARPROP, and 98% from the training by ECEN-iRprop$^+$. It is noteworthy that the network with the wider active input segment activation functions: tanh, 1-RPP, and 2-RPP converge in all trials within less than 1000 epochs on average. Furthermore, the network with p-RPP converges faster than the network with tanh activation function.

Table 4.1: Success rate, required mean epoch and standard deviation as well as minimum and maximum epochs for the network applying to Sonar problem. NCs in the table means there is no convergent training. The structure of network is 60–1–1.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
CEN - iRprop$^+$	success(%)	0	0	0	0
	mean	NC	NC	NC	NC
	sd	NC	NC	NC	NC
	min	NC	NC	NC	NC
	max	NC	NC	NC	NC
ECEN - iRprop$^+$	success(%)	100	98	100	100
	mean	830	723	794	818
	sd	318	227	240	251
	min	339	299	412	400
	max	1683	1569	1324	1397
CEN - SARPROP	success(%)	0	0	0	0
	mean	NC	NC	NC	NC
	sd	NC	NC	NC	NC
	min	NC	NC	NC	NC
	max	NC	NC	NC	NC
ECEN - SARPROP	success(%)	100	66	100	100
	mean	971	715	902	962
	sd	485	146	318	341
	min	595	518	523	603
	max	3097	1120	2671	2294

4.2.2 The n–to–n encoder

In the next two set of experiments, a multilayered neural network was trained to function as an n–to–n encoder [15]. The n–to–n encoder is implemented by a neural network with

n inputs and n output units trained to map n input pattern into the output. Each of the n patterns contains only one active element, represented in this experiment by $+1$, while all the other elements are inactive, represented by -1. The number of hidden units of an n–to–n encoder, denoted by n_h, is typically smaller than n and at least equal to $\log_2(n)$. If $n_h < \log_2(n)$, it is a more difficult task called "tightly compression". In this dissertation, we intend to evaluate the proposed algorithms by using the tough one, a network with $N^{(0)} = 60$ inputs, $N^{(1)} = 3$ hidden units, and $N^{(2)} = 60$ units its was trained to function as an encoder60: 60–to–60 encoder. The encoder120: 120–to–120 encoder, was also realized by a network with $N^{(0)} = 120$ inputs, $N^{(1)} = 4$ hidden units, and $N^{(2)} = 120$ units. The number of adaptation cycles required by each of the sixteen combinations of algorithms and network architectures for converging to the total classification error 0 as well as the success rate of training, evaluated by averaging the results of 100 trials are shown in two tables. Table 4.2 shows the experimental results of encoder60, and Table 4.3 the experimental results of encoder120. These comparisons demonstrate the efficacy of the proposed additional term as well as the effect of active input segment size, especially in complex training tasks.

From Tables 4.2 and 4.3, both ECEN-iRprop$^+$ and ECEN-SARPROP are very successful algorithms when compared to CEN-iRprop$^+$ and CEN-SARPROP. ECEN-iRprop$^+$ converges nearly 100% in all acivation function used while CEN-iRprop$^+$ fails nearly 100% in all trials. CEN-SARPROP is more successful than CEN-iRprop$^+$. However, ECEN-SARPROP gets the highest success rate of 100% if 1-, 2-RPP and tanh activation function were used. The network with 0-RPP fails more frequently than other functions. Even the case it converges, the time is still very long. Our algorithms

not only have a higher successful training rate but also are faster than the original algorithms.

When $N^{(1)} < N^{(0)}$, the hidden units form a data compression layer. The function of the 60–3–60 architecture is about $60 : \log_{3.9}(60)$ compression because it comprises only 3 hidden units, as well as the 120–4–120 architecture, its function, is about $120 : \log_{3.3}(120)$ compression because it has only 4 hidden units. The former is more compact than the latter. And these compression ratio are more compact than $n : log_2(n)$ in Karayiannis's method [15]. Furthermore, that method not only required longer epochs it also used a more effective weights initialization procedure than ours. The more nonlinear activation function is, the faster training is. This can be seen from Tables 4.2 and 4.3. Every algorithm yields the higher success rates and fewer mean epochs if the networks use the wider active input segment activation function, and each of which requires a polynomial of higher degrees. When the activation function is concerned, the network with 2-RPP is not only faster but is also more stable than the network with tanh.

4.2.3 Higher order parity

Parity problems are difficult tasks for feedforward networks, especially as the order of the problem increases. Tesauro and Janssens [40] used backpropagation approach to study N-parity functions. They employed and N-$2N$-1 fixed architecture to reduce the problems of getting stuck in a local minima. They required an average of 1953 epochs for the 8-parity problem. However, 9-18-1 architecture was not applicable for the 9-parity problem. Yang and Kao [41] reported the proposed evolutionary based

Table 4.2: Success rate, required mean epoch and standard deviation as well as minimum and maximum epochs for the network applying to Encoder60 problem. NCs in the table means there is no convergent training. The structure of network is 60-3-60.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
	success(%)	57	0	19	59
CEN	mean	7318	NC	7662	6222
-	sd	2628	NC	3116	3029
iRprop$^+$	min	3080	NC	4036	2552
	max	17399	NC	16794	15035
	success(%)	100	99	100	100
ECEN	mean	4690	7085	4251	3694
-	sd	1403	2319	1861	1378
iRprop$^+$	min	2718	3564	2310	2257
	max	9733	16041	15189	9175
	success(%)	97	83	97	98
CEN	mean	4449	5523	4036	3155
-	sd	1673	2209	2315	1592
SARPROP	min	2261	2742	1711	1637
	max	13941	15958	17168	11639
	success(%)	100	45	100	100
ECEN	mean	1974	8187	2425	1905
-	sd	177	4142	422	183
SARPROP	min	1616	2699	1753	1554
	max	2563	19054	3749	2492

algorithm (FCEA) required an average of 3650 and 6704 epochs for the 8- and the 9-parity problems. And FCEA obtained 90% convergent rate for N-N-1 on the 8-parity problem. However, they did not report convergent rate on the 9-parity problem. Recently, Ampazis and Perantonis [10] showed the results of training an 8-8-1 (eight inputs, one hidden layer with eight nodes, and one output node) network on 8-bit parity problem according to the second-order training algorithms. Note that all conventional training algorithms (BFGS [36], Inverse-BFGS [37], CG/PR [38], LM [39]) failed to converge in all trials. The LMAM [10] algorithm was able to solve the problem at least in 14% of the trials. The OLMAM [10] algorithm showed a very high success rate

Table 4.3: Success rate, required mean epoch and standard deviation as well as minimum and maximum epochs for the network applying to Encoder120 problem. NCs in the table means there is no convergent training. The structure of network is 120-4-120.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
	success(%)	67	0	0	2
CEN	mean	5328	NC	NC	4006
-	sd	1482	NC	NC	1804
iRprop$^+$	min	2922	NC	NC	2730
	max	10966	NC	NC	5281
	success(%)	100	100	100	100
ECEN	mean	3318	4999	2944	2643
-	sd	576	929	474	375
iRprop$^+$	min	2217	3282	1983	1842
	max	5274	7952	4203	3725
	success(%)	67	13	67	77
CEN	mean	4780	9461	4648	4088
-	sd	1646	5289	2566	3137
SARPROP	min	2646	3999	2507	2198
	max	12411	18017	14145	19747
	success(%)	100	53	100	100
ECEN	mean	2364	6153	3039	2181
-	sd	197	3594	575	180
SARPROP	min	2003	2647	2130	1800
	max	2927	18282	4904	2640

(94%) along with a smaller mean value of epochs than LMAM. However, they are all the second-order algorithms.

In our experiments, we used two network architectures. An 8-8-1 (eight inputs, one hidden layer with eight nodes, and one output node) network is for the 8-bit parity problem, and a 9-9-1 (nine inputs, one hidden layer with nine nodes, and one output node) network is for the 9-bit parity problem. The number of adaptation cycles required by each of the sixteen combinations of algorithms and network architectures for converging to the number of total classification error 0 as well as the standard deviation, minimum and maximum cycles required, the number of successful training over 100 trials, which

all these measurements are evaluated by averaging the results of 100 trials are shown in two tables. Table 4.4 shows the experimental results of 8-bit parity, and Table 4.5 the experimental results of 9-bit parity problems, respectively.

From Table 4.4, the algorithms with the anti-Hebbian rule are more stable than the algorithms without anti-Hebbian rule. Accordingly, the success rates of ECEN-iRprop$^+$ comparing to CEN-iRprop$^+$ enormously increase from 12 to 69, 32 to 58, 15 to 91, and 15 to 94. And likewise, the success rates of ECEN-SARPROP comparing to the original CEN-SARPROP increase significantly from 25 to 95, 19 to 20, 16 to 89, and 22 to 98, when the activation functions are tanh, 0-, 1-, and 2-RPPs, respectively.

Although the number of required mean epochs for a convergent training of the algorithms with anti-Hebbian are more than those of the algorithms without the anti-Hebbian rule, the characteristic of the n-parity problem itself is an explanation. There is a very long plateau before a solution is discovered. We can interpret them from the results of training by CEN-iRprop$^+$ and CEN-SARPROP. The mean number of adaptation cycles is very short but the number of successful trainings is very low. That means most of the time the training process has to fight against the plateau. When the training process encounters a plateau, that training may not get a solution in time. According to unsupervised information from the anti-Hebbian rule, the training process can get back at the plateau latter.

The network with 0-RPP activation functions is not only the most unstable but also is the slowest network compared to the network with other activation functions. That means the length of the active input segment before outputting either -1 or $+1$ plays a significantly important role during training.

Table 4.5 shows the results from the 9-bit parity problem. The interpretations and conclusions are the same as from Table 4.4. The algorithms with anti-Hebbian rule do more successfully solve the 9-parity problem than the algorithms without anti-Hebbian rule.

From the experimental results regarding the algorithms with anti-Hebbian rule, the performance of the algorithm with anti-Hebbian rule is less sensitive to initial weights of RPP network than the algorithm without anti-Hebbian rule. The wider active input segment function has, the more stable network is.

Table 4.4: Success rate, required mean epoch and standard deviation as well as minimum and maximum epochs for the network applying to 8-bit parity problem. The structure of network is 8-8-1.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
	success(%)	12	32	15	15
CEN	mean	536	4049	465	1356
-	sd	413	4440	173	2377
iRprop$^+$	min	189	485	234	284
	max	1777	15210	780	9150
	success(%)	69	58	91	94
ECEN	mean	4964	10452	4258	3370
-	sd	3524	4940	4544	4175
iRprop$^+$	min	576	2193	296	496
	max	16705	19302	19671	19511
	success(%)	25	19	16	22
CEN	mean	1040	4631	2245	938
-	sd	851	3850	4766	650
SARPROP	min	357	584	427	436
	max	4269	15758	18068	2797
	success(%)	95	20	89	98
ECEN	mean	4000	6678	3307	1732
-	sd	4805	6058	4078	1714
SARPROP	min	295	288	304	340
	max	19726	18224	19585	11226

Table 4.5: Success rate, required mean epoch and standard deviation as well as minimum and maximum epochs for the network applying to 9-bit parity problem. The structure of network is 9-9-1.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
	success(%)	13	39	18	14
CEN	mean	631	3997	516	1160
-	sd	616	4510	279	1458
iRprop$^+$	min	199	276	193	201
	max	2462	17297	1095	5151
	success(%)	58	65	96	97
ECEN	mean	4178	9444	2940	2218
-	sd	3573	5320	3139	2784
iRprop$^+$	min	381	1189	277	276
	max	16187	19946	17477	15318
	success(%)	15	33	14	24
CEN	mean	1097	6250	1888	1586
-	sd	660	4793	2727	1835
SARPROP	min	389	518	254	208
	max	2800	16089	9959	6446
	success(%)	97	36	96	96
ECEN	mean	2324	5089	3379	2476
-	sd	1902	5205	3619	2940
SARPROP	min	253	248	309	167
	max	10168	19987	18734	15569

4.2.4 The two-spiral benchmark

The two-spiral problem is a typical benchmark in the field of neural networks. This problem was originally proposed by A. Wieland as a challenging benchmark for feedforward networks, where two spiral parts totaling 194 associations should be classified by training. The task is to construct a classifier capable of distinguishing between the two classes. Some researchers reported that gradient-descent-based algorithms failed to converge when tested on this training set. Furthermore, even in cases where these algorithms were successful, the number of required weights adaptation epochs for convergence was

very high [42]. Karayiannis [15] also reported a convergent gradient-descent-based training. The weights were initialized by a very efficient method: generalized anti-Hebbian rule. The training set used in these experiments consisted of 150 points belonging to two interspersed spiral-shaped classes, with 75 samples for each class. However, the number of epochs was still very high.

In our work, we use a conventional feedforward network comprising of two inputs, 30 hidden nodes, one output unit, without any shortcut connections between nonadjacent layers. This is the same structure of Karayiannis's experiments [15]. The training set consists of two spiral parts of 150 associations. Specifically, the training set example sequences $(\mathbf{a}^{(51)}, \mathbf{b}^{(51)}), (\mathbf{a}^{(52)}, \mathbf{b}^{(52)}) \cdots$, with $\mathbf{a}^{(t)} = (a_1^{(t)}, a_2^{(t)}) \in \mathcal{R}^2$, $\mathbf{b}^{(t)} = b^{(t)} \in \mathcal{R}^1$, are given by the following equations. For $n = 26, 27, \cdots, 97$,

$$\begin{aligned} a_1^{(2n-1)} &= 1 - a_1^{(2n)} = r_n \sin \alpha_n + 0.5, \\ a_2^{(2n-1)} &= 1 - a_2^{(2n)} = r_n \cos \alpha_n + 0.5, \end{aligned} \tag{4.13}$$

where $r_n = 0.4((105 - n)/104)$, $\alpha_n = \pi(n-1)/16$, and $b^{2n-1} = 1$, $b^{2n} = -1$. This network architecture with the p-RPPs and tanh activation functions varying training algorithms are used to complete the task. The number of required adaptation cycles by each of the sixteen combinations of algorithms and network architectures for converging to the number of total classification error 0 as well as the standard deviation, minimum and maximum cycles required, the number of successful training over 100 trials, which all these measurements are evaluated by averaging the results of 100 trials, are shown in Table 4.6.

From Table 4.6, the network trained by the algorithm with anti-Hebbian rule is not the fastest network. In addition, the ECEN-iRprop$^+$ is slower than the CEN-iRprop$^+$. In case of CEN- and ECEN-SARPROPs, their mean epochs are comparable. This is because the characteristic of problem itself. There is a report that the number of hidden units plays a crucial role for the classification and generalization quality of the network [16]. While a 100% correct classification of a fixed number of patterns can be achieved relatively easy, it is rather difficult to achieve a perfect generalization. As anti-Hebbian rule is added to the error term, both the error surface and the solution are modified.

From the experiments, the networks with 1-RPP and 2-RPP are both faster and more stable than the network with 0-RPP. The generalization ability of the networks trained by various algorithms was evaluated by visualizing the outputs of the input vectors not included in the training set. The testing input set consists of data 171248 patterns: 154x139, which are different from 150 training inputs. The results of each of the sixteen combinations of network architectures and the training algorithms are displayed in Fig. 4.1 as follows. Rows 1, 2, 3, and 4 represent the separations and contours of the two-dimensional (2-D) input vector spaces produced by CEN-iRprop$^+$, ECEN-iRprop$^+$, CEN-SARPROP, and ECEN-SARPROP , respectively. The outputs of networks with tanh, 0-RPP, 1-RPP, and 2-RPP activation functions are in the first, the second, the third and the last columns, respectively. The results shown in the figure are the simple averaging [43] of 100 trial outputs. The contoured outputs is only class -1.

Fig. 4.1 shows that the decision boundaries produced from the network trained by the algorithms with and without anti-Hebbian rule are very similar. However, when we

consider the contour lines, the algorithms with anti-Hebbian rule yields a better contour than the algorithms without anti-Hebbian rule. Comparing Fig. 4.1 (C) and (D) to (c) and (d), the contours produced from the network with 1-RPP and 2-RPP trained by ECEN-iRprop$^+$ are smoother than that from the network with the same activation function trained by CEN-iRprop$^+$. Comparing row 3 to row 4, the contours produced from the network trained by ECEN-SARPROP are smoother than that from the network with the same activation function trained by CEN-SARPROP.

Hence, not only the number of hidden units but also the shape of the hidden unit and the regularization term play important roles in the training, classification and generalization quality of the network.

4.3 Summary

In this chapter, three modifications were made for new algorithm. First, modified error function was selected to eliminate the sigmoid prime factor for the output units. Second, the input patterns were normalized in order to balance the dynamic range of the inputs. And third, anti-Hebbian rule was added to cooperate with the error back-propagated from the output layer. It was experimentally verified that the enhanced algorithms proposed in this dissertation considerably improves the convergence rate and the successful training rate of the two sophisticated first-order gradient-descent-based algorithms: ECEN-iRprop$^+$ and ECEN-SARPROP, used to perform nontrivial training tasks.

Table 4.6: Success rate, required mean epoch and standard deviation as well as minimum and maximum epochs for the two-spiral problem. NCs in the table means there is no convergent training. The structure of network is 2-30-1.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
	success(%)	88	0	99	99
CEN	mean	31648	NC	17629	15089
-	sd	20963	NC	13207	7833
iRprop$^+$	min	6219	NC	6640	6491
	max	87671	NC	93870	51857
	success(%)	9	0	94	99
ECEN	mean	53912	NC	24687	22059
-	sd	19368	NC	15880	14965
iRprop$^+$	min	36704	NC	7594	7465
	max	98322	NC	91249	88309
	success(%)	100	99	100	100
CEN	mean	2762	21058	3656	3436
-	sd	613	19781	2807	1601
SARPROP	min	1383	2891	1725	1351
	max	4542	99246	29260	11712
	success(%)	100	99	100	100
ECEN	mean	2564	8627	3633	3931
-	sd	1140	8028	2016	2312
SARPROP	min	1089	1097	1504	1468
	max	9398	44743	15904	17354

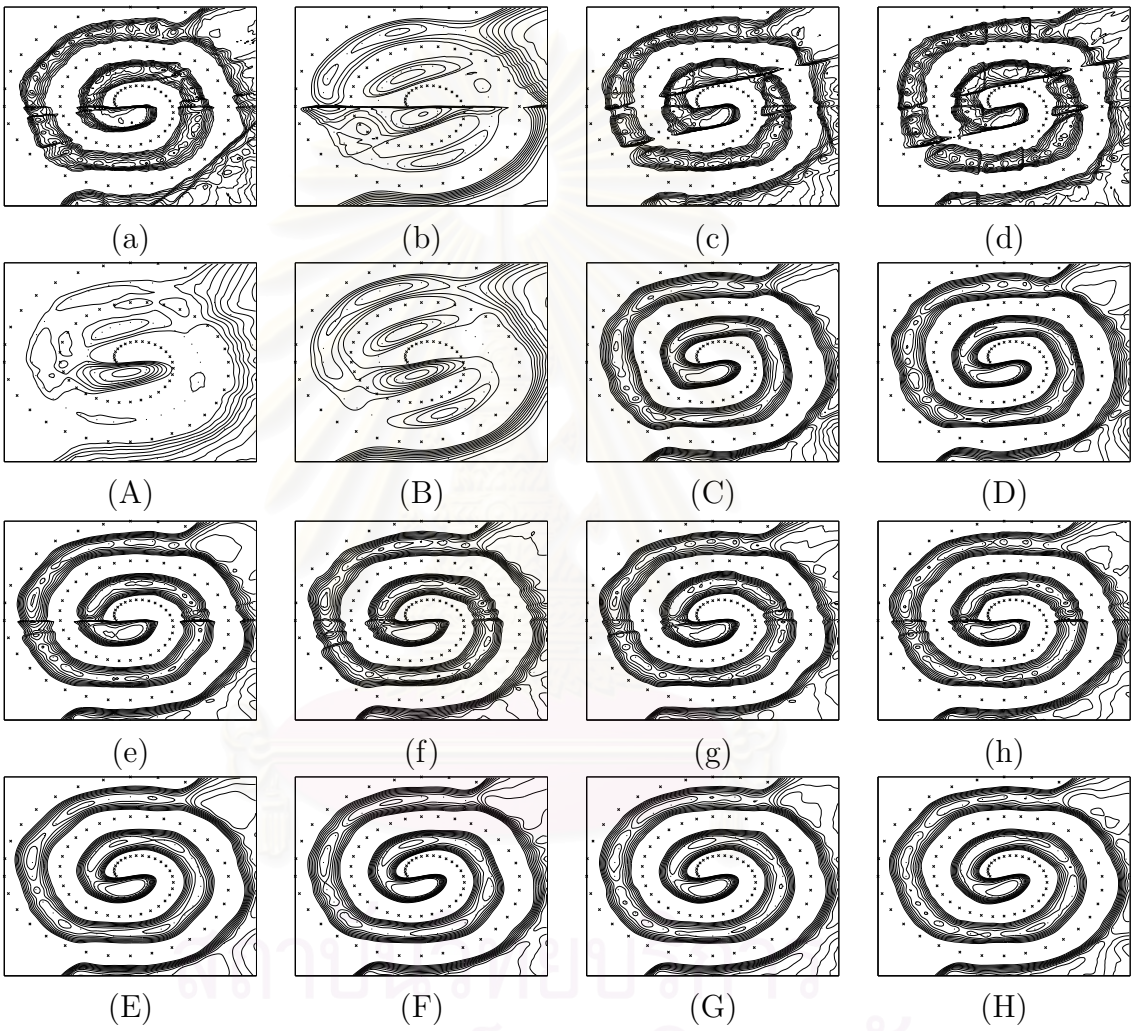

Figure 4.1: The decision regions of the networks. The training algorithms are CEN-iRprop$^+$ and ECEN-iRprop$^+$ for the two upper panels, and CEN-SARPROP and ECEN-SARPROP for the two lower panels. From left to right, the activation functions are tanh, 0-RPP, 1-RPP, and 2-RPP, respectively.

CHAPTER V

New Modified Error Function

In the previous Chapter, the local minima problem was overcome by the combination of Oh's error function excluding the derivative of the output node and the additional error function revealing the additional anti-Hebbian term for updating the weights connecting the input and the hidden layers. In this chapter, we generalized that additional error function by automatically accounting the correct output.

5.1 New Modified Additional Error Function

A new modified error function is given by

$$\overline{E}(\mathbf{w}) = \frac{1}{S \times N^{(2)}} \left(\sum_{s=1}^{S} E(\mathbf{w}) + \alpha \sum_{s=1}^{S} E(\mathbf{w}) \left(\sum_{j=1}^{N^{(l=1)}} H_{j,s}(\mathbf{w}) \right) \right), \tag{5.1}$$

An added term can be defined by

$$E_B = \sum_{s=1}^{S} E(\mathbf{w}) \left(\sum_{j=1}^{N^{(l=1)}} H_{j,s}(\mathbf{w}) \right). \tag{5.2}$$

This added term is used to keep the magnitude of activation value in the hidden layer small while $E(\mathbf{w})$ is large. While the output layer approximates the desired targets, the

effect of term E_B will be diminished and will eventually become zero. Using the above error function as the objective function, the gradient-descent information for updating rule of weight $w_{ji}^{(l)}$ can be rewritten as

$$-\frac{\partial \overline{E}(\mathbf{w})}{\partial w_{ji}^{(l)}} = -\frac{\partial E(\mathbf{w})}{\partial w_{ji}^{(l)}} - \alpha^{(l)} \frac{\partial E_B(\mathbf{w})}{\partial w_{ji}^{(l)}}, \quad l = 1, 2, \tag{5.3}$$

where $\alpha^{(1)}$ and $\alpha^{(2)}$ are the learning rate for $E_B(\mathbf{w})$ in hidden layer and output layer, respectively. We usually set $\alpha^{(1)} = 0.01$ and $\alpha^{(2)} = 0.004$.

If $E(\mathbf{w})$ is the *sum-squared-error* (SSE) defined by

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{N^{(l=2)}} (t_k^s - x_k^{s(2)})^2, \tag{5.4}$$

then, for pattern s, the derivative $\frac{\partial E(\mathbf{w})}{\partial w_{ji}}$ can be computed by BP. $\frac{\partial E_B(\mathbf{w})}{\partial w_{ji}}$ is easily obtained as follows. For the weight connected to the output layer,

$$\frac{\partial E_B(\mathbf{w})}{\partial w_{ji}^{(l=2)}} = \frac{\partial E(\mathbf{w})}{\partial w_{ji}^{(l=2)}} \sum_{j=1}^{N^{(l=1)}} H_{j,s}(\mathbf{w}). \tag{5.5}$$

For weights connected to the hidden layer,

$$\begin{aligned} \frac{\partial E_B(\mathbf{w})}{\partial w_{ji}^{(l=1)}} &= \frac{\partial E(\mathbf{w})}{\partial w_{ji}^{(l=1)}} \sum_{j=1}^{N^{(l=1)}} H_{j,s}(\mathbf{w}) + x_i^{s(0)} \hat{x}_j^{s(1)} E^s(\mathbf{w}), & (5.6) \\ &= \frac{\partial E(\mathbf{w})}{\partial w_{ji}^{(l=1)}} \sum_{j=1}^{N^{(l=1)}} H_{j,s}(\mathbf{w}) + x_i^{s(0)} \hat{x}_j^{s(1)} \sum_{k=1}^{N^{(l=2)}} (t_k^s - x_k^{s(2)})^2. & (5.7) \end{aligned}$$

If $E(\mathbf{w})$ is Oh's modified function, $\frac{\partial E_B(\mathbf{w})}{\partial w_{ji}^{(l)}}$ can be obtained by the same way of above

formulas as follows. For the weight connected to the output layer, $\frac{\partial E_B(\mathbf{w})}{\partial w_{ji}^{(l=2)}}$ is Eq. (5.5). For weights connected to the hidden layer, $\frac{\partial E_B(\mathbf{w})}{\partial w_{ji}^{(l=1)}}$ is Eq. (5.6). However, for simplicity of error computation, we can use Eq. (5.7) rather than Eq. (5.6).

5.2 Experimental Results

The evaluations of the proposed enhancements Eq.(5.1) to the weights updating procedures are presented in this section. We compare the effect of using two different $E(\mathbf{w})s$: the classical SSE and the Oh's modified function of the previous chapter. SARPROP and iRprop$^+$ algorithms are used as the weight updating procedures. However, we also combine the adaptive regularization parameter selection (ARPS) method [47] for assigning the appropriate amount of the weight decay term to iRprop$^+$ algorithm.

We use the same criterion as in Chapter 4 to name the ten combinations of algorithms. For the network with $E(\mathbf{w})s$ is SSE, SSEN-iRprop$^+$ means the SSE Normalized input iRprop$^+$, SSEN-iRprop$^+$ARPS means the SSE Normalized input iRprop$^+$ with ARPS method, ESSEN-iRprop$^+$ means the Enhanced SSE Normalized input iRprop$^+$) with ARPS method (the anti-Hebbian rule is included), SSEN-SARPROP means Enhanced SSE Normalized input SARPROP, ESSEN-SARPROP means the Enhanced SSE Normalized input SARPROP with ARPS method (the anti-Hebbian rule is included).

For the network with $E(\mathbf{w})s$ is the the Oh's modified function, CEN-iRprop$^+$ means the CEN iRprop$^+$, CEN-iRprop$^+$ARPS means the CEN iRprop$^+$ with ARPS method, ECEN-iRprop$^+$ means the Enhanced CEN iRprop$^+$) with ARPS method (the anti-Hebbian rule is included), CEN-SARPROP means CEN SARPROP, ECEN-SARPROP

means the Enhanced CEN SARPROP with ARPS method (the anti-Hebbian rule is included).

There are four activation function types: tanh, 0-, 1-, 2-, and 3-RPPs. Hence, there are totally 40 combinations of experimental sets. The data set used in these experiments is the IRIS problem. The data set is publicly available from the CMU Repository of Neural Network Benchmarks at http://www.boltz
.cs.cmu.edu. The IRIS has four input variables and has three output classes . Each class is comprising of 50 instances, and, totally, 150 instances are in the whole data set. For each run, the 40 out of 50 instances in each class are randomly selected as the training set and the remaining instances are in test set. Hence, there are 120 instances in the training set, and 30 instances in the test set.

The network architecture for each of these experiments is 4-3-3, and each of which uses only one activation function type.

All simulations were carried out on a Pentium III 1.06 GHz with 640 MB RAM PC. The MATLAB executable *mex* file of each algorithms was complied by MSVC++ 6.0.

In all cases 50 training trials were performed (with uniformly random initialization of the weights in $(-0.5, 0.5)$. The maximum number of epochs was set to 20000. The temperature T of SARPROP is set to 0.01. The remaining parameters of the both algorithms are the default values.

The criteria for evaluation and comparison will be:

- the percentage of the successful training,
- the quality of solution: the percentage of correct classification when the trained

network classifies the unseen input. Regarding to all the successful training, the mean, standard deviation (sd), minimum and maximum percentages of the correct classification are measured.

The training was considered successful whenever Fahlman's "40-20-40" criterion [28] was satisfied (i.e., values in the lowest 40% of the output range were treated as output -1, and values in the highest of 40% were treated as output $+1$, and values in the middle 20% were treated as indeterminate and therefore were considered as incorrect).

In the following, the experiments on the networks are concluded, organized by the error functions used.

5.2.1 Sum-Squared-Error (SSE)

From Table 5.1, we can see that the algorithms cooperated with anti-Hebbian rule and weight decay term yield a higher successful training rate than the algorithms without anti-Hebbian rule and weight decay term. In all case, the success rates are significantly increased. Regarding to the generalization, the network with tanh, 2-, and 3-RPPS yield better percentage of classification corrections if the SARPROP cooperates with anti-Hebbian rule.

5.2.2 The Oh's Modified Error, $n = 2$ (CE2)

Comparing the success rates in Table 5.1 and Table 5.2, the network using CE2 yields a significantly better successful training rate than the network using SSE. The lowest success is more than 56% from the algorithm without the aid of anti-Hebbian rule. The

highest success of the network with SSE from the algorithm with and without the aid of anti-Hebbian rule are 56.0% and 38.67%, respectively. The iRprop$^+$ algorithm with the weight decay term or the anti-Hebbian rule has higher success rate than that without them. However, the generalization is slightly decreased. The SARPROP algorithm with the anti-Hebbian rule not only has a significantly higher success rate but also yields a better generalization than that algorithm without the anti-Hebbian rule. This one is the best among our experiments.

5.3 Summary

The new modified additional error function was presented in this chapter. The experimental results revealed that SARPROP cooperated with the anti-Hebbian rule showed the best results. It is the most reliable algorithm in term of success rate and generalization of the trained network.

Table 5.1: Success rate, The mean and standard deviation as well as minimum and maximum percent of testing correction for the IRIS problem. The $E(\mathbf{w})$ is SSE. The structure of network is 4-3-3.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
SSEN - iRprop$^+$	success(%)	26.0	28.0	24.0	14.0
	correction(%)				
	mean	94.87	93.33	94.72	93.33
	sd	4.64	3.92	4.60	1.92
	min	83.33	83.33	83.33	90.00
	max	100.00	100.00	100.00	96.67
SSEN - iRprop$^+$ARPS	success(%)	46.0	34.0	30.0	34.0
	correction(%)				
	mean	93.84	93.18	92.47	93.33
	sd	4.26	3.63	4.58	4.75
	min	86.67	86.67	80.00	80.00
	max	100.00	100.00	100.00	100.00
ESSEN - iRprop$^+$ARPS	success(%)	56.0	52.00	32.00	40.00
	correction(%)				
	mean	94.04	94.49	93.96	95.00
	sd	4.06	3.52	3.04	3.33
	min	86.67	90.00	90.00	90.00
	max	100.00	100.00	100.00	100.00
SSEN - SARPROP	success(%)	38.67	32.67	26.0	28.0
	correction(%)				
	mean	93.62	94.63	92.39	92.86
	sd	3.38	4.18	3.33	4.00
	min	83.33	86.87	86.87	83.33
	max	100.00	100.00	100.00	100.00
ESSEN - SARPROP	success(%)	44.67	36.00	33.00	38.67
	correction(%)				
	mean	94.18	93.33	93.47	93.33
	sd	4.07	3.89	3.56	3.45
	min	83.33	83.33	86.67	86.67
	max	100.00	100.00	100.00	100.00

Table 5.2: Success rate, The mean and standard deviation as well as minimum and maximum percent of testing correction for the IRIS problem. The $E(\mathbf{w})$ is CE2. The structure of network is 4-3-3.

		Activation Function			
Algorithm	Performance	tanh	0-RPP	1-RPP	2-RPP
	success(%)	56.00	68.00	62.00	60.00
	correction(%)				
CEN	mean	94.64	95.00	95.48	95.11
-	sd	3.67	4.44	3.90	3.69
iRprop$^+$	min	83.33	83.33	86.67	86.67
	max	100.00	100.00	100.00	100.00
	success(%)	90.00	68.00	88.00	96.00
	correction(%)				
CEN	mean	93.56	93.73	94.55	94.24
-	sd	4.52	4.77	4.39	4.70
iRprop$^+$ARPS	min	80.00	83.33	83.33	80.00
	max	100.00	100.00	100.00	100.00
	success(%)	100.00	96.00	100.00	98.00
	correction(%)				
ECEN	mean	94.33	94.31	94.80	94.56
-	sd	5.10	4.81	4.53	4.55
iRprop$^+$	min	83.33	83.33	83.33	83.33
	max	100.00	100.00	100.00	100.00
	success(%)	64.00	44.00	66.00	70.00
	correction(%)				
CEN	mean	94.90	94.85	95.76	95.81
-	sd	4.23	3.52	3.93	4.30
SARPROP	min	86.67	90.00	86.67	86.67
	max	100.00	100.00	100.00	100.00
	success(%)	100.00	100.00	100.00	100.00
	correction(%)				
ECEN	mean	95.73	95.40	95.87	95.80
-	sd	3.87	3.56	3.60	3.55
SARPROP	min	86.67	86.67	83.33	90.00
	max	100.00	100.00	100.00	100.00

CHAPTER VI

Conclusions

This dissertation showed an example that the network with tanh and the network with tanh-like activation functions quite differed in response while the shape of those activation functions are very similar. A class of p-RPP tanh-like neurons was systematically developed. The shape and computational complexity of neurons can be changed by the parameter p. Their execution times are shorter than those of both sigmoid and tanh and yet comparable to the lookup table scheme. Regarding to the generalization and convergent enhancement, three modifications were made for the new algorithm. First, input normalization process of [23] was performed in order to balance the dynamic range of the inputs. Second, modified error function of Oh was selected to eliminate the sigmoid prime factor for the output units. And third, anti-Hebbian rule was generalized and added to cooperate with the error back-propagated from the output layer. The proposed scheme was combined with the weight decay term in order to improve the generalization of the trained network as well as the convergence of training process.

Our ongoing work includes the following. In Chapter 4 and 5, we focus on the error saturation elimination of classification problem which highly depends on the error function selected. With the cooperation of anti-Hebbian rule, the hidden layer accomplishes the data compression task. Applications range from image compression [44] to function

approximation [45, 46]. Since the radial basis function-like is also directly constructed by the proposed method, the application of RBF-like function to the existing devices using RBF should be explored.

We proposed some heuristics to set the factor $\alpha(t)$ of the anti-Hebbian term in the weight updating rule. We are developing a general criterion to select the proper value of $\alpha(t)$. This may contribute to improve not only the learning convergence but also the network generalization. In our formulation of the p-RPP neurons, the size of the active input segment is fixed for a given p. It would be useful if the segment can be extend during training. This may contribute to improve the learning convergence.

A major advantage of the proposed algorithms is that they achieved their training performance without the need for careful selections of training parameters. The generalization of the trained network is also enhanced. This makes them attractive choices among the first-order training algorithms.

References

[1] Haykin S. (1999). *Neural networks: A comprehensive foundation.* 2nd ed. Upper Saddle River, New Jersey: Prentice–Hall.

[2] Pao YH. (1989) *Adaptive pattern recognition and neural networks.* Reading, MA: Addison–Wesley.

[3] Kwan HK. (1992). Simple sigmoid-like activation function suitable for digital hardware implementation. *IEE Electronics Letters* 28(15): 379–1380.

[4] Zhang M, Vassiliadis S, Delgado-Frias JG. (1996). Sigmoid generators for neural computing using piecewise approximation. *IEEE Trans Computers* 45(2): 1045–1049.

[5] Basterretxea K, Tarela JM, del Campo I. (2002). Digital design of sigmoid approximation for artificial neural networks. *IEE Electronics Letters* 38(1): 35–37.

[6] Piazza A. (1993). Neural networks with digital LUT activation function. *Proceedings of 1993 International Joint Conference on Neural Networks (IJCNN'93)* 2: 1401-1404.

[7] Piazza A, Uncini A, Zenobi M. (1992). Artificial neural networks with adaptive polynomial activation function. *Proceedings International Joint Conference on Neural Networks (IJCNN)*, Beijing, China, II:343-349.

[8] Guarnieri S, Piazza F, Uncini A. (1999). Multilayer feedforward networks with adaptive spline activation function. *IEEE Trans Neural Networks* 10(3): 672–683.

[9] Vecci L, Piazza F, Uncini A. (1998). Learning and approximation capabilities of adaptive spline neural networks. *Neural Networks* 11(2): 259–270.

[10] Ampazis N, Perantonis SJ. (2002). Two highly efficient second-order algorithms for training feedforward networks. *IEEE Trans Neural Networks* 13(5): 1064–1074.

[11] Fukuoka Y, Matsuki H, Minamitani H, Ishida A. (1998). A modified back-propagation method to avoid false local minima. *Neural Networks* 11(6): 1059–1072.

[12] Lee H-M, Chen C-M, Huang T-C. (2001). Learning efficiency improvement of back-propagation algorithm by error saturation prevention method. *Neurocomputing* 41(1-4): 125–143.

[13] Ng S-C, Cheung C-C, Leung S-H. (2003). Integration of magnified gradient function and weight evolution with deterministic perturbation into back-propagation. *IEE Electronics Letters* 39(5): 447–448.

[14] Gorman RP, Sejnowski TJ. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks* 1: 75–89.

[15] Karayiannis NB. (1996). Accelerating the training of feedforward neural networks using generalized hebbian rules for initializing the internal representations. *IEEE Trans Neural Networks* 7(2): 419–426.

[16] Mandischer M. (2002). A comparison of evolution strategies and backpropagation for neural network training. *Neurocomputing* 42(1-4): 87–117.

[17] Igel C, Huesken M. (2003). Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing* 50: 105–123.

[18] Treadgold NK, Gedeon TD. (1998). Simulated annealing and weight decay in adaptive learning: The SARPROP algorithm. *IEEE Trans Neural Networks* 9(4): 662–668.

[19] Hertz J, Krogh A, Palmer RG. (1991). *Introduction to the theory of neural computation.* Redwood City: Addison-Wesley.

[20] McCulloch WS, Pitts W. (1943). A logical calculus of the ideas imminent in nervous activity . *Bull Math Biophys* 5: 115-133.

[21] Mandic DP, Chambers JA. (2001). *Recurrent neural networks for prediction, learning algorithms, architectures and stability.* West Sussex: John Wiley & Sons.

[22] Rumelhart DE, Hinton GE, Williams RJ. (1986). Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL, editors. *Parallel Distributed Processing (PDP): Exploration in the microstructures of cognition*, pp.318–362. Cambridge, MA: MIT Press.

[23] Joost M, Schiffmann W. (1998). Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization. *International Journal of Uncertainty, Fuzzyness and Knowledge-Based Systems* 6(2): 117–126.

[24] van Ooyen A, Nienhuis B. (1992). Improving the convergence of the backpropagation algorithm. *Neural Networks* 5: 465–471.

[25] Oh S-H. (1997). Improving the error backpropagation algorithm with a modified error function. *IEEE Trans Neural Networks* 8(3): 799–803.

[26] Palmieri F, Catello C, D'Orio G. (1999). Inhibitory synapses in neural networks with sigmoidal nonlinearities. *IEEE Trans Neural Networks* 10(3): 635–644.

[27] Riedmiller M, Brun H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA USA, 586-591.

[28] Fahlman SE. (1988). *An empirical study of learning speed in back-propagation networks.* Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA.

[29] http://www.microsoft.com

[30] http://www.mingw.org

[31] http://www.cs.virginia.edu/∼lcc-win32

[32] Rujan P. (1997). Playing billiards in version space. *Neural Computation* 9: 99–122.

[33] Torres Moreno JM, Gordon MB. (1998). Characterization of the sonar signals benchmarks. *Neural Processing Letters* 7(1): 1–4.

[34] Perantonis SJ, Virvilis V. (1999). Input feature extraction for multilayered perceptrons using supervised principal components analysis. *Neural Processing Letters* 10(3): 243–252.

[35] Hasenjäger M, Ritter H. (1999). Perceptron learning revisited: The sonar targets problem. *Neural Processing Letters* 10(1): 1–8.

[36] Battiti R. (1992). First and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation* 4(2): 141–166.

[37] Battiti R. (1989). Accelerated backpropagation learning: Two optimization methods. *Complex syst* 3: 331–342.

[38] Johansson EM, Dowla FU, Goodman DM. (1991). Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems* 2(4): 291–301.

[39] Hagan MT, Menhaj M. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Trans Neural Networks* 5(6): 989–993.

[40] Tesauro G, Janssens B. (1988). Scaling relationships in backpropagation learning. *Complex Systems*, 2: 39–84.

[41] Yang J-M, Kao C-Y. (2001). A robust evolutionary algorithm for training neural networks. *Neural Comput & Applic* 10(3): 214–230.

[42] Whitley D, Karunanithi N. (1991). Generalization in feedforward neural networks. *IJCNN-91-Seattle International Joint Conference on Neural Networks (IJCNN'91)*, 2: 77-82.

[43] Perrone MP, Cooper LN. (1993). When networks disagree: Ensemble Methods for Hybrid Neural Networks. In: Mamone RJ (editor). *Neural Networks for Speech and Image Processing*, pp.126–142. Chapmal-Hall.

[44] Jiang J. (1999). Image compression with neural networks-A survey. *Signal Processing: Image communication* 14(9): 737–760.

[45] Cybenko G. (1989). Approximation by superposition of a sigmoid function. *Mathematics of Control, Signals and Systems* 2(4): 303–314.

[46] Chen DS, Jain RC. (1994). A robust back propagation learning algorithm for function approximation. *IEEE Trans Neural Networks* 5(3): 467–479.

[47] Leung C-T, Chow TWS. (1999). Adaptive regularization parameter selection method for enhancing generalization capability of neural network. Artificial intelligence, 107: 347-356.

Appendix A

iRprop$^+$ Algorithm

Require: The input/output data set and the initial weights and biases of network.
Ensure: The weights and biases of the trained network.

{The iRprop$^+$ error gradient is computed by

$$\frac{\partial E}{\partial w_{ji}(t)}^{iRprop^+} = \frac{\partial \overline{E}}{\partial w_{ji}(t)}$$

}

$\forall\, i,j:\ \Delta_{ji}(0) = \Delta_0$
$\forall\, i,j:\ \frac{\partial E}{\partial w_{ji}(0)} = 0$
$t = 1$
repeat
  Compute iRprop$^+$ Gradient $\frac{\partial E}{\partial w_{ji}(t)}$
  for all weights and biases **do**
    if $\frac{\partial E}{\partial w_{ji}(t)} \times \frac{\partial E}{\partial w_{ji}(t-1)} > 0$ **then**
      $\Delta_{ji}(t) = \text{minimum}(\Delta w_{ji}(t-1) \times \eta^+, \Delta_{max})$
      $\Delta w_{ji}(t) = -\text{sign}(\frac{\partial E}{\partial w_{ji}(t)}) \times \Delta_{ji}(t)$
      $w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t)$
    else if $\frac{\partial E}{\partial w_{ji}(t)} \times \frac{\partial E}{\partial w_{ji}(t-1)} < 0$ **then**
      $\Delta_{ji}(t) = \text{maximum}(\Delta w_{ji}(t-1) \times \eta^-, \Delta_{min})$
      if $E(t) > E(t-1)$ **then**
        $w_{ji}(t+1) = w_{ji}(t) - \Delta w_{ji}(t-1)$
      end if
      $\frac{\partial E}{\partial w_{ji}(t)} = 0$
    else
      $\Delta w_{ji}(t) = -\text{sign}(\frac{\partial E}{\partial w_{ji}(t)}) \times \Delta_{ji}(t)$
      $w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t)$
    end if
  end for
  $t = t + 1$
until Converged or $t >$ maximum epoch

Appendix B

SARPROP Algorithm

Require: Temperature: T. The input/output data and the initial weights and biases of network.

Ensure: The weights and biases of the trained network.

{The SARPROP error gradient is computed by

$$\frac{\partial E}{\partial w_{ji}(t)}^{SARPROP} = \frac{\partial \overline{E}}{\partial w_{ji}(t)} + 0.1 \times w_{ji}/(1 + w_{ji}^2) \times SA,$$

}

$\forall\, i, j:\ \Delta_{ji}(0) = \Delta_0$
$\forall\, i, j:\ \frac{\partial E}{\partial w_{ji}(0)} = 0$
$t = 1$
repeat
  $SA = 2^{-t \times T}$
  Compute SARPROP Gradient $\partial E / \partial w_{ji}(t)$
  for all weights and biases **do**
    if $\frac{\partial E}{\partial w_{ji}(t)} \times \frac{\partial E}{\partial w_{ji}(t-1)} > 0$ **then**
      $\Delta_{ji}(t) = \text{minimum}(\Delta w_{ji}(t-1) \times \eta^+, \Delta_{max})$
      $\Delta w_{ji}(t) = -\text{sign}(\frac{\partial E}{\partial w_{ji}(t)}) \times \Delta_{ji}(t)$
      $w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t)$
    else if $\frac{\partial E}{\partial w_{ji}(t)} \times \frac{\partial E}{\partial w_{ji}(t-1)} < 0$ **then**
      if $\Delta_{ji}(t-1) < 0.4 * SA^2$ **then**
        $r = \frac{rand()}{RAND\_MAX} \quad \{r \in [0, 1)\}$
        $\Delta_{ji}(t) = \Delta_{ji}(t-1) \times \eta^- + 0.8 \times r \times SA^2$
      else
        $\Delta_{ji}(t) = \Delta_{ji}(t-1) \times \eta^-$
      end if
      $\Delta_{ji}(t) = \text{maximum}(\Delta_{ji}(t-1), \Delta_{min})$
      $\frac{\partial E}{\partial w_{ji}(t)} = 0$
    else
      $\Delta w_{ji}(t) = -\text{sign}(\frac{\partial E}{\partial w_{ji}(t)}) \times \Delta_{ji}(t)$
      $w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t)$
    end if
  end for
  $t = t + 1$
until Converged or $t >$ maximum epoch

Vita

Mr. Khamron sunat was born in November 2, 1965. He received bachelor degree in Chemical Engineering from department of Chemical Technology faculty of Science Chulalongkorn University in 1990. He received master degree in Computational Science from department of Mathematics faculty of Science Chulalongkorn University in 1999. He received RGJ award from the Thailand Research Fund (TRF) for pursing the doctorate. Now, he is a lecturer at department of Computer Engineering faculty of Engineering at Mahanakorn University of Technology, Bangkok.