

COMPARISON OF PLACE CODING AND THERMOMETER CODING  
IN NEURAL NETWORKS

By

BEUM-SEUK LEE

Bachelor of Science

Seoul, South Korea

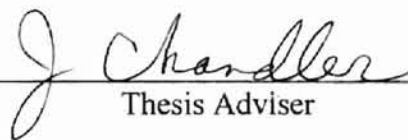
1996

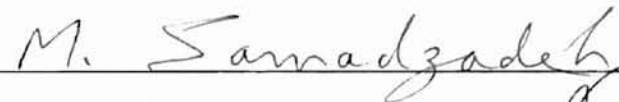
Submitted to the faculty of  
the Graduate College of  
the Oklahoma State university  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE

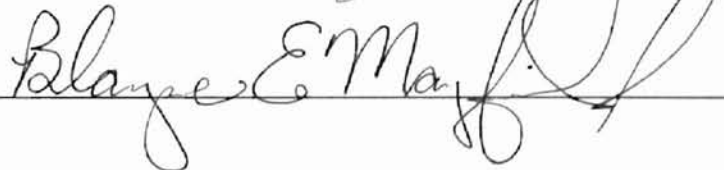
May 1999

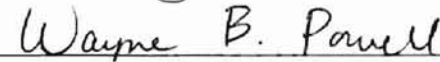
COMPARISON OF PLACE CODING AND THERMOMETER CODING  
IN NEURAL NETWORKS

Thesis Approved:

  
\_\_\_\_\_  
Thesis Adviser

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_  
Dean of the Graduate College

## PREFACE

In neural network research, there have been many studies about optimization, weight updating methods, and various structures of networks. However, there also has been some research on codings in neural networks used in solving the feature extracting problem. Meaningful input representation allows a network to work efficiently because of the specific way the network implements its function with given input samples. This can be explained by the Categorical Perception (CP) effect inside of a neural network.

This thesis shows the significance of meaningful input representation by comparing the behavior of a network trained with two different types of input codings (place coding and thermometer coding). Tests are conducted using different numbers of hidden nodes, different numbers of hidden layers, four types of transfer functions, and four sets of real world data.

## ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. John P. Chandler, for his intelligent guidance, constructive criticism, and inspiration. I also wish to thank my other committee members Drs. Blayne E. Mayfield and Mansur H. Samadzadeh.

I also would like to express my sincere gratitude to my original advisor, Dr. W. Nick Street for the direction of this research.

## TABLE OF CONTENTS

Chapter	Page
I INTRODUCTION -----	1
II LITERATURE REVIEW -----	2
2.1 Artificial Neural Networks (ANN) -----	2
2.1.1 What Is an ANN? -----	2
2.1.2 What Kind of Architecture Does an ANN Have? -----	2
2.1.3 How Does an ANN Learn? -----	4
2.1.4 How Can an ANN Work Better? -----	8
2.2 Categorical Perception(CP) Effect -----	9
2.2.1 What Is the CP Effect? -----	9
2.2.2 How Does the CP Effect Related to Coding? -----	9
2.2.3 What Makes Thermometer Coding Better? -----	10
III RESULTS -----	12
3.1 Learning with Default Values for the Neural Network -----	12
3.1.1 Learning without Noise -----	13
3.1.2 Learning with Noise	
(Gaussian Noise with Variance of 1/2 ) -----	15
3.2 Learning with Different Transfer Functions -----	17
3.3 Learning with Different Numbers of the Hidden Nodes -----	18
3.4 Learning with Different Numbers of Hidden Layers -----	19
3.5 Learning with Different Data -----	21
3.6 Learning with Different Optimizations -----	21
IV CONCLUSIONS AND FUTURE WORK -----	23

Chapter	Page
REFERENCES -----	25
APPENDICES -----	27
APPENDIX A - A GLOSSARY -----	28
APPENDIX B - DATA SPECIFICATION -----	29
APPENDIX C - PROGRAM LISTING -----	41

## LIST OF TABLES

Table	Page
I Result of Learning without Noise -----	15
II Result of Learning with Noise -----	16
III Result of Learning with RADBAS Transfer Function -----	17
IV Result of Learning with LOGSIG Transfer Function -----	17
V Result of Learning with TRIBAS Transfer Function -----	18
VI Result of Learning with 6 Hidden Nodes -----	18
VII Result of Learning with 10 Hidden Nodes -----	19
VIII Result of Learning with 50 Hidden Nodes -----	19
IX Result of Learning with 2 Hidden Layers -----	20
X Result of Learning with 3 Hidden Layers -----	20
XI Result of Learning with 4 Hidden Layers -----	20
XII Result of Learning with Real World Data -----	21
XIII Result of Learning with Different Optimizations -----	22

## LIST OF FIGURES

Figure	Page
1 Types of decision regions formed by different layers [22] -----	3
2 A neural network with one hidden layer [19] -----	4
3 An ANN representation of a neuron [13] -----	5
4 Place coding and Thermometer coding -----	10
5 MSE graph of place coding without noise -----	14
6 MSE graph of thermometer coding without noise -----	14
7 MSE graph of place coding with noise -----	15
8 MSE graph of thermometer coding with noise -----	16



## CHAPTER I

### INTRODUCTION

This thesis concerns the representation of data in neural nets. The performance of neural nets depends not only on the learning rule and the architecture but also on the method of encoding (i.e., the representation). It is intuitively obvious that when a net is trying to establish a mapping between a set of inputs and a set of outputs, the task will be easier if similar inputs are mapped to similar outputs [8].

This thesis suggests that a proportional form of coarse encoding may be appropriate for the representation of data for neural networks. We can observe some biological systems using a proportional form of coarse coding. Each cell responds to a range of input values, in-between but overlapping with those of its neighbors. Any given input is influenced by the relative activity of a number of neighboring cells [7].

This thesis starts with the Categorical Perception (CP) effect inside neural networks. I will conclude this paper by showing how the input representation of data in an ANN can influence the network's behavior by comparing place code and thermometer code from the viewpoint of the CP effects.

## CHAPTER II

### LITERATURE REVIEW

#### 2.1 Artificial Neural Networks (ANN)

Neurons are neural cells and neural networks are networks of these cells (e.g., brain). The three important parts of a computational system based on Artificial Neural Networks are the transfer function, the architecture, and the learning rule.

##### 2.1.1 What Is an ANN?

An ANN is a distributed computational system with some number of processing elements connected to each other and also may be defined as an adaptive, dynamic, and parallel system with self-learning capabilities that can carry out information processing tasks [13].

Artificial Neural Net models try to achieve good performance through interconnection of simple computational elements. In this respect, ANN structures are based on our present understanding of biological nervous systems.

##### 2.1.2 What Kind of Architecture Does an ANN Have?

The architecture of the network is the manner of connections in an ANN through which information in the network flows. There are useful architectural configurations like single-layer, multi-layer, feedforward, feedback and lateral connectivity [15]. The capabilities of multi-layer perceptrons come from the nonlinearities used within nodes.

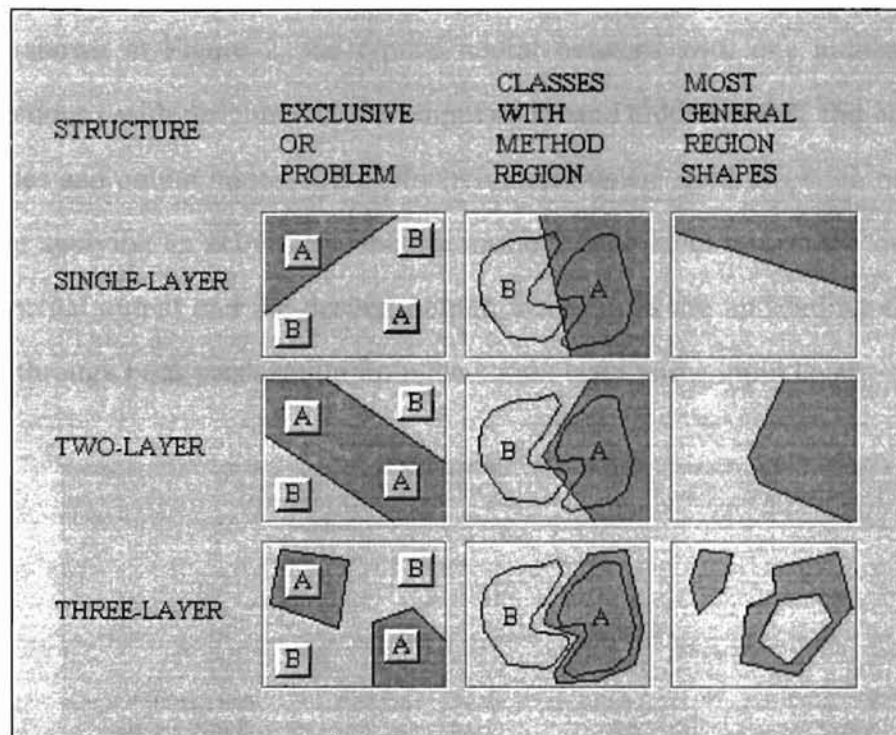


Figure 1. Types of decision regions formed by different layers [17]

As shown in Figure 1, a single layer perceptron used for a linearly separable classification problem forms hyperplane-decision regions. A two-layer perceptron can form any possibly unbounded region in the space generated by the inputs. Such regions include polygons. In the two-layer perceptron, each node in the first layer behaves like a single-layer perceptron and has a “high” output only for points on the side of the hyperplane formed by its weights and offset. This works as a logical AND operation in the output node and builds up a final decision region that is the common area of all the hyperplane regions. Intersections of such hyperplanes form regions. The number of these regions’ sides corresponds to the number of nodes in the first layer.

A three-layer perceptron can form arbitrary complex decision regions and can separate the complicated classes. The output of second layer nodes will be “high” only for inputs within each hypercube. We can observe similar behavior in multi-layer perceptrons, with multiple output nodes with sigmoidal nonlinearities, and the decision value is used to select the class corresponding to the output with the largest output.

As shown in Figure 2, the typical neural network with one hidden layer has interconnections (with weights) between input nodes and hidden nodes, and also between hidden nodes and output nodes. Those interconnections are used to get an output given an input by applying an activation function in the hidden layer and in the output layer. With this actual output and the desired output, the weights are updated by the learning rule, often through back propagation from the output layer to the input layer.

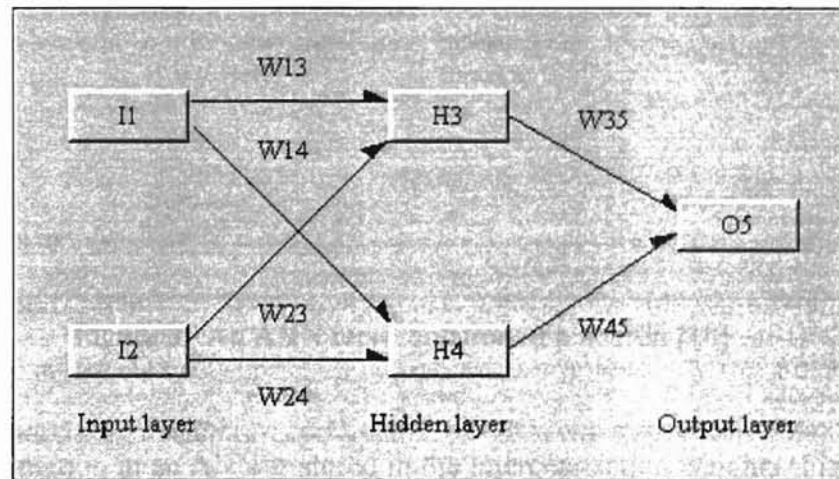


Figure 2. A neural network with one hidden layer [15]

### 2.1.3 How Does an ANN Learn?

Weights in an ANN function as memory in a conventional computer. The learning rule is the method used to adjust the weights in the process of training the network. That is, artificial neural systems are not programmed, rather they are taught. The learning can be supervised or unsupervised. The most widely used supervised learning rule is the back-propagation method, although this is not a very efficient method.

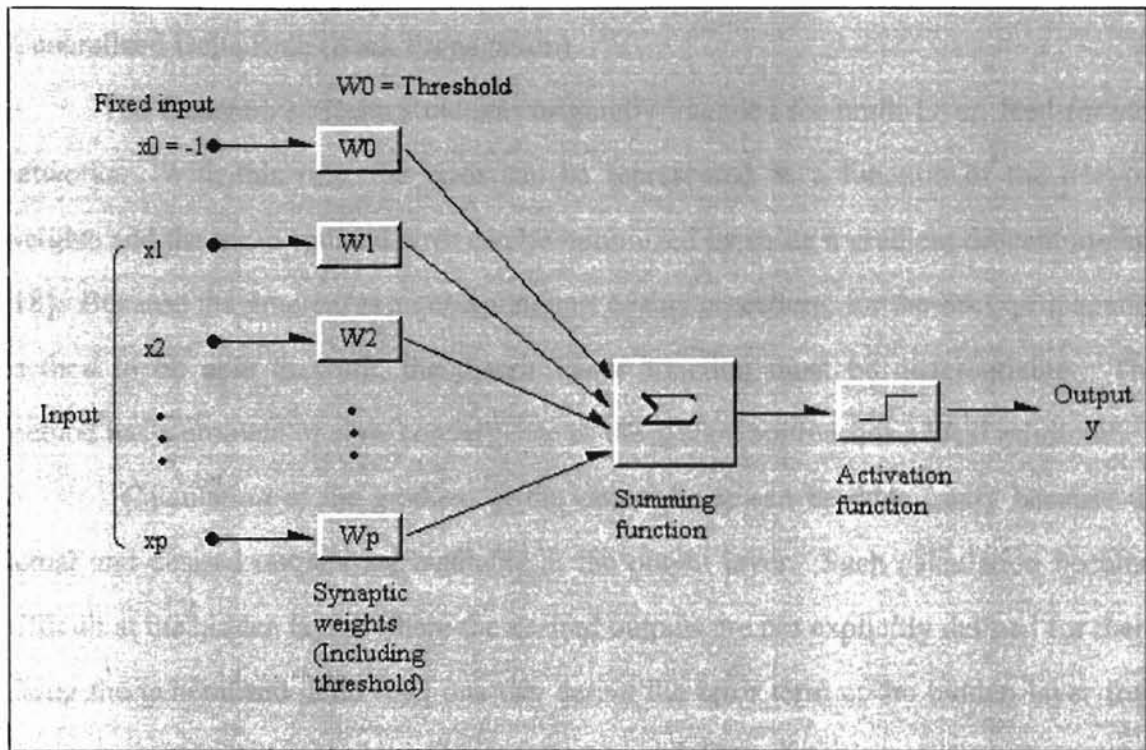


Figure 3. An ANN representation of a neuron [10]

All information in an ANN is stored in the interconnection weights (Figure 3) and, during the learning process, the weights are updated. A weight shows the strength of association - that is, the co-occurrence of connected features, characteristics, properties, or events during a training procedure. Tasks are typically understood as the minimization of an energy function in weight space [10].

### Back-propagation

The back-propagation (BP) learning method typically uses the Generalized Delta Rule. First, the network output is calculated with values from the input layer through hidden layer(s) [16]. An error at the output layer is computed by comparing this output and the desired output. Finally during a backward propagation of this error, to adjust future outputs, each neuron updates the weights of its input connections to decrease the error related to its output activation [1].

### Generalized Delta Rule (Back Propagation)

The Generalized Delta Rule was originally intended for multi-layer, feed-forward networks. With this rule, the error can be represented as a function of the network weights and the mean squared error can be minimized by using a gradient descent method [18]. Because the gradient is an essential part of this procedure, for the back-propagation method to be able to work, the discriminator function must be differentiable. This method has a problem of slow convergence as the system approaches a local minimum.

Calculation of the gradient at the output layer can be done easily because the actual and desired outputs are available at the output layer. Such calculation becomes difficult at the hidden layers where the desired outputs are not explicitly defined for them. Using the generalized delta rule, one can derive the error term at the hidden layer from the errors propagated back down through the network [7].

### Wiener-Hopf Equations

Let's say that sensors produce individual signals,  $x_1, x_2, \dots, x_p$ . These signals are then applied to corresponding set of weights,  $w_1, w_2, \dots, w_p$ . The weighted signals are then summed to produce the output signal "y".

$$\mathbf{y} = \sum_{k=1}^p \mathbf{w}_k \mathbf{x}_k$$

The object is to obtain the optimum setting of the weights,  $w_1, w_2, \dots, w_p$ , so as to minimize the difference between the system output "y" and some desired response "d" in a mean square sense [7].

$$\mathbf{J} = \frac{1}{2} \mathbf{E}[(\mathbf{d}-\mathbf{y})^2]$$

## Error Minimization Problem

The question in this error minimization problem is to determine the optimum set of weights for which the mean-squared error is minimum. The cost function versus the weights results in the error-performance surface, or simply “error surface”. Ideally, the error surface is bowl-shaped, with a well-defined bottom or global minimum where the mean-squared error has its minimum value. To get this optimum condition, we differentiate the cost function with respect to the weights and then set the result to zero for all nodes. This derivatives of the cost function is called the gradient of the error surface with respect to the weights [7].

## Method of Steepest Descent

The weights have a time-varying form, and their values are adjusted in an interactive way along the error surface, moving them progressively toward the optimum solution. The method of steepest descent continually seeks the bottom point of the error surface. Successive changes are applied to the weights in the direction of steepest descent of the error surface, that is, in a direction opposite to the gradient vector [7].

$$\mathbf{w}_k(\mathbf{n}+1) = \mathbf{w}_k(\mathbf{n}) - \eta \nabla_{\mathbf{w}_k} J(\mathbf{n})$$

Because back propagation is based on steepest descent, it is so slow that it is obsolete. There are some methods like conjugate gradient methods, Marquardt’s method, Newton’s method and Quasi-Newton methods, that are much faster than this method. But the back-propagation method is useful when we try to understand the basic behavior of an ANN because of its simple configuration. [7]

### 2.1.4 How Can an ANN Work Better?

#### Cross-Validation

A well-generalized network has a reasonably accurate mapping for future input-output patterns that were not used in creating or training the network. If a neural network has too many weights, it will learn the training well but it will be less able to generalize between untrained similar input-output patterns. The Cross-validation method is often adopted to solve this over-generalization problem [7].

In the Cross-validation method, the available data set is partitioned into a training set and a test set. The idea here is to measure the generalization performance of the network on a data set different from the one used for training the network. In this way we can select a network which learns enough about the past to generalize to the future. [7]

#### Momentum

By including a momentum term in the delta rule, we can increase the learning rate while still keeping the network stable. When the partial derivative of the error function with respect to the weight vector has the same sign on consecutive iterations, the exponentially weighted sum increases, and so the weight is updated by a large amount, accelerating change in a steady downhill direction in the error space. When the partial derivative error function with respect to the weight vector has opposite signs on consecutive iterations, the exponentially weighted sum decreases, and so the weight is changed by a small amount, stabilizing and avoiding long steps that oscillate in sign. [7]

#### Stopping Criteria

In general, for the back-propagation algorithm there are no well-defined criteria for stopping its operation. Rather, we can use some reasonable criteria to terminate the weight updates.

The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient value (or ||



step size  $\| < 10^{-5}$ ) and when the generalization performance passes through a local maximum and starts to decrease.

## 2.2 Categorical Perception (CP) Effect

### 2.2.1 What Is the CP Effect?

There is a method that learns the embedded regularities in samples of input by which patterns are sorted and named. Hanson defined “The Categorical Perception (CP) effect works as an interaction between discrimination (the capacity to tell pairs of stimuli apart, which is a relative judgment) and identification (the capacity to categorize or name individual stimuli, which is an absolute judgment)” [3].

When a network is trained to classify input patterns into categories, it compresses within-category distances and expands between-category distances. Such CP categories may be the basic form with which higher-order categories are made up.

The supervised learning in an ANN adjusts the pairwise distance between the inputs to sort them into the categories until it obtains sufficient within-category compression and between-category separation to accomplish reliable categorization.

There are four factors for the categorization during the learning process in an ANN:

(1) maximal separation between input patterns, (2) linear separability at the hidden layer level, (3) repulsive force to widen the gap between category boundary, and (4) the similarity of the input codings [6].

### 2.2.2 How Does the CP Effect Related to Coding?

All we expect an ANN to do for categorization, is to extract and encode relevant properties (symbol tokens) from the input patterns. Those tokens, which are interpreted only by their form (syntactic) rather than their meaning (semantic) must be reducible to non-symbolic, shape-preserving (iconic) representations. Iconicity representations are

used for relative discrimination because they preserve the common character of the input for same-different judgments and pattern-matching. The iconicity factor corresponds to  $12/2/98$  the number of common features between patterns. Therefore, in order for the network to show the CP effects, the patterns have to be represented in a “meaningful” way [4].

### 2.2.3 What Makes Thermometer Coding Better?

Place coding is the simplest form of ANN codings. In place coding, there is only one output of “1” where the value range ends, otherwise output is “0”. Thermometer coding is very similar to place coding, except that each unit remains as “1” if the value is equal to or less than the value range as shown in Figure 4 [7].

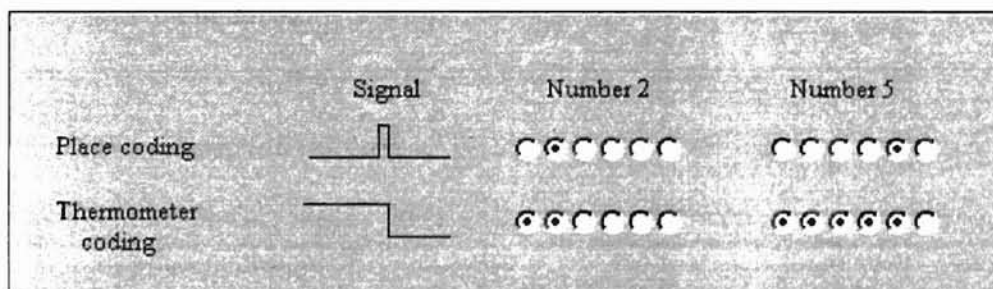


Figure 4. Place coding and Thermometer coding

The place code is more arbitrary and the thermometer code more analog because the thermometer code preserves some multi-unit constraints where the place code does not [4]. These extra multi-unit constraints mean relevant information spilled over to adjacent units, which is critical during categorization tasks. Usually an N+1-parameter account should be truer than an N-parameter account of the same data, in that more redundant information is used for reliability, robustness or speed [5].

Thermometer code has the advantage that the net only has to learn to turn a unit on when the stimulus increases without turning off the unit next to it, and that the number of active units is proportional to the coded value, whereas the place representation often

separates patterns into categories by expanding the value of hidden units to its maximum or minimum. Iconic factors build up internal representations in which similar patterns share overall physical similarities of shape. In the hidden-unit space, the more iconic nets (thermometer code) push similar patterns close to one another, whereas the more arbitrary input codings (place code) tend to push patterns to maximal bounds.

With place codings, only the endpoints of the category range are trained while the interior points within the category are left free to vary, whereas a thermometer representation interpolates to the untrained region because of the common parts from the structure-preserving form.

## CHAPTER III

### RESULTS

In this research work, a multi-layer perceptron was trained with the Generalized Delta Rule (back-propagation algorithm). The implementation languages were Java [14, 12, 9] and MATLAB [11].

To improve the learning, the initialization of the synaptic weights and the threshold levels of the network are uniformly distributed inside a small range, and pattern-by-pattern updating was done for on-line operation. The order in which the training examples are presented to the network was randomized (i.e., shuffled) from one epoch to the next. Finally a momentum constant was added in the equation for weight updating.

The performance of place coding and thermometer coding was checked by comparing 1) Least-mean-squared error and generalization (cross validation) error, 2) error for theoretical data and for real-world data, and 3) error with noise and without noise with different transfer functions, the numbers of hidden nodes and the numbers of hidden layers.

#### 3.1 Learning with Default Values for the Neural Network

- Learning rule for the backpropagation : Gradient descent with momentum and adaptive learning rate backpropagation
- Transfer function in the hidden layer : Hyperbolic tangent sigmoid transfer function
- The number of the nodes in the hidden layer : 3
- The number of the hidden layer : 1
- Maximum epochs to stop the training : 4000

- The number of input nodes : 100
- The number of output nodes : 1
- Minimum gradient value to stop the training : 5e-004
- Learning rate : 0.1
- Momentum constant : 0.9

Data : Theoretical data (Classification) : the input of the data is 100 bits encoded according to the type of coding. The output of data is the corresponding integer number to the input value. For example for place coding, if the  $i$ -th of input bit is "1" and all others are "0" then the output of this input is an integer " $i$ ". For the same output value of thermometer coding, all input bits until the  $i$ -th bit is "1" and after the  $i$ -th bit, all input bits are "0".

### 3.1.1 Learning Without Noise

Thermometer coding converged much slower but had much smaller MSE and generalization MSE than place coding . For place code the error values didn't change much even with more learning.

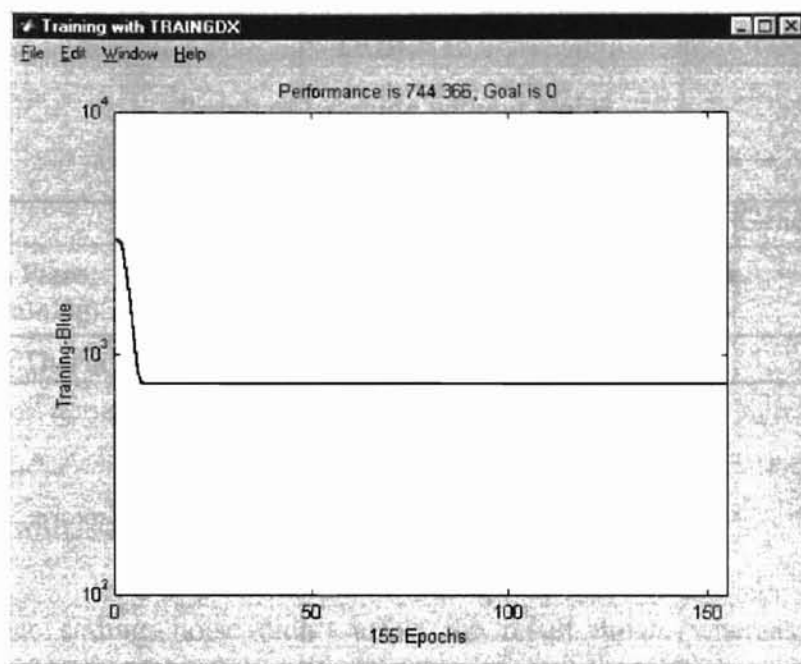


Figure 5. MSE graph of place coding without noise

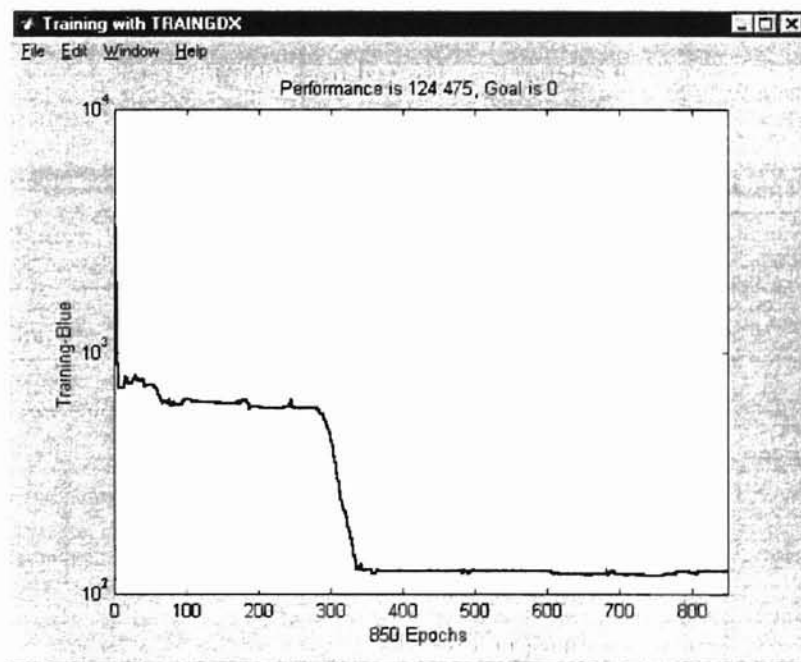


Figure 6. MSE graph of thermometer coding without noise

TABLE I

Result of Learning without Noise

	Epoch	MSE	General MSE
Classification Place (with more training)	155.7 (875)	829.86 (829.86)	897.65 (897.60)
Classification Therm	874.5	94.637	110.56

### 3.1.2 Learning with Noise (Gaussian Noise with Variance of $1/2$ )

For place coding, noise didn't affect the result much, whereas added noise increased MSE and generalization MSE by a great measure for thermometer coding. Again, increased training epochs didn't have much effect on the error of place coding.

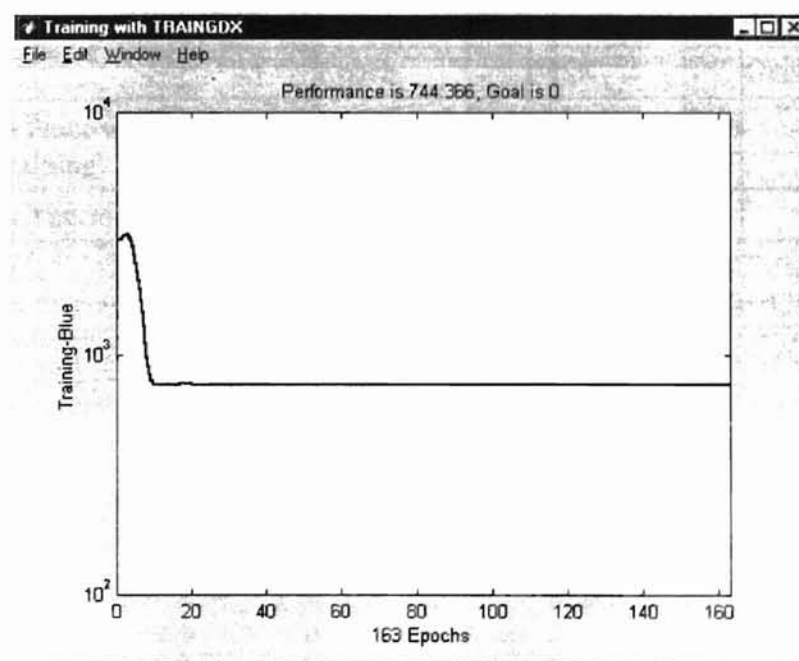


Figure 7. MSE graph of place coding with noise

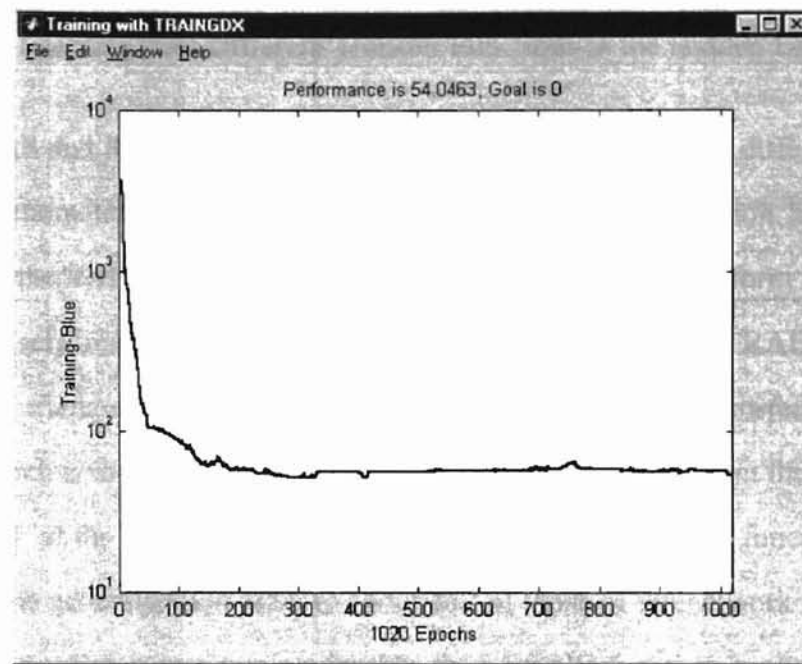


Figure 8. MSE graph of thermometer coding with noise

TABLE II

Result of Learning with Noise

	Epoch	MSE	General MSE
<b>Classification Place with Noise (with more training)</b>	168.1 (1140)	829.86 (829.86)	897.65 (910.58)
<b>Classification Therm with Noise</b>	1139.9	219.23	207.97



### 3.2 Learning with Different Transfer Functions in the Hidden Layer

For the TRIBAS and RADBAS transfer functions, there wasn't much difference between place and thermometer coding, whereas the LOGSIG transfer function had almost the same result as the TANSIG transfer function. This result shows the form of the transfer function is closely related to the CP effect since the TRIBAS and RADBAS transfer functions have triangular shape while the forms of the other two transfer functions are rectangular which is more similar to the input representation. Also from the result values, we can see that at the expense of more epochs, the LOGSIG transfer function decreased the MSE and the generalization MSE to about half of those of the network with TANSIG transfer function which is less rectangular than the LOGSIG function.

TABLE III

Result of Learning with RADBAS Transfer Function

	Epoch	MSE	General MSE
<b>Classification Place</b>	129	829.86	897.65
<b>Classification Place with Noise</b>	129	829.86	897.65
<b>Classification Therm</b>	174.8	829.72	900.57
<b>Classification Therm with Noise</b>	153.3	829.86	897.65

TABLE IV

Result of Learning with LOGSIG Transfer Function

	Epoch	MSE	General MSE
<b>Classification Place</b>	160.8	829.86	897.65
<b>Classification Place with Noise</b>	138.5	829.86	897.65
<b>Classification Therm</b>	3816.7	33.749	63.563
<b>Classification Therm with Noise</b>	3690.4	43.742	72.862

TABLE V  
Result of Learning with TRIBAS Transfer Function

	Epoch	MSE	General MSE
<b>Classification Place</b>	129	829.86	897.65
<b>Classification Place with Noise</b>	129	829.86	897.65
<b>Classification Therm</b>	129.1	829.86	899.29
<b>Classification Therm with Noise</b>	135.1	829.86	896.05

### 3.3 Learning with Different Numbers of the Hidden Nodes

The more hidden nodes the network has, the better we can see the CP effect of thermometer coding by checking the fact that for thermometer coding, the MSE and Generalization MSE keep decreasing in proportional to the number of hidden nodes while the networks with place coding didn't have much change with different numbers of hidden nodes.

TABLE VI  
Result of Learning with 6 Hidden Nodes

	Epoch	MSE	General MSE
<b>Classification Place</b>	156.1	829.49	904.76
<b>Classification Place with Noise</b>	170.8	829.49	904.76
<b>Classification Therm</b>	2688	158.37	216.29
<b>Classification Therm with Noise</b>	1291	157.41	284.84

TABLE VII  
Result of Learning with 10 Hidden Nodes

	Epoch	MSE	General MSE
<b>Classification Place</b>	248	829.86	897.65
<b>Classification Place with Noise</b>	259.8	829.86	897.65
<b>Classification Therm</b>	2381	87.013	125.54
<b>Classification Therm with Noise</b>	1125.8	116.58	181.82

TABLE VIII  
Result of Learning with 50 Hidden Nodes

	Epoch	MSE	General MSE
<b>Classification Place</b>	695.4	829.86	897.65
<b>Classification Place with Noise</b>	939.4	761.25	980.85
<b>Classification Therm</b>	4000	16.288	76.551
<b>Classification Therm with Noise</b>	4000	27.282	87.323

### 3.4 Learning with Different Numbers of Hidden Layers

More hidden layers in a network hindered the CP effect for thermometer coding while it didn't affect place coding.

TABLE IX  
Result of Learning with 2 Hidden Layers

	Epoch	MSE	General MSE
<b>Classification Place</b>	152.2	829.86	897.65
<b>Classification Place with Noise</b>	536.6	829.86	897.69
<b>Classification Therm</b>	2812.6	191.88	224.73
<b>Classification Therm with Noise</b>	3555.1	203.91	252.4

TABLE X  
Result of Learning with 3 Hidden Layers

	Epoch	MSE	General MSE
<b>Classification Place</b>	160	829.86	897.65
<b>Classification Place with Noise</b>	558.5	829.86	897.65
<b>Classification Therm</b>	2717.3	241.63	371.5
<b>Classification Therm with Noise</b>	1819.7	344.43	459.47

TABLE XI  
Result of Learning with 4 Hidden Layers

	Epoch	MSE	General MSE
<b>Classification Place</b>	199.7	829.86	897.65
<b>Classification Place with Noise</b>	206.6	829.86	897.65
<b>Classification Therm</b>	1498.9	519.9	562.9
<b>Classification Therm with Noise</b>	1158.2	496.08	560.44

### 3.5 Learning with Different Data

Normalized real-world data (Hayes-Roth, Balance-Scale, Glass and Breast-Cancer data) is used. Each column of data was encoded as one block with five bits and those bits are marked according to the data value and coding type. All the data is trained by networks with all the default values specified in this chapter. The appendix of this paper contains detailed specification for each data.

Thermometer coding gave the networks better MSE and generalization MSE with real-world data.

TABLE XII  
Result of Learning with Real World Data

	<b>Epoch</b>	<b>MSE</b>	<b>General MSE</b>
<b>Hayes-Roth Place</b>	3228	0.17	0.169
<b>Hayes-Roth Thermometer</b>	4000	0.059	0.0985
<b>Balance-Scale Place</b>	4000	0.0109	0.0143
<b>Balance-Scale Thermometer</b>	1898.4	0.0006	0.0011
<b>Glass Place</b>	3216.1	1.1126	2.8815
<b>Glass Thermometer</b>	4000	0.2747	2.0299
<b>Breast-Cancer Place</b>	2830.3	1.2353	6.2711
<b>Breast-Cancer Thermometer</b>	3726.6	0.5064	4.4451

### 3.6 Learning with Different Optimization

Using three different optimizations which are BFGS quasi-Newton (Q-Newton), Scaled Conjugate Gradient (SCG) and Gradient Descent (GD) Optimizations, input samples are trained with different range of desired output values. We can see that Q-Newton optimization converged quite faster than others with good results.

TABLE XIII  
Result of Learning with Different Optimizations

Output Range	Data	Optimization	Epoch	MSE	General MSE
-1 to 1	Classification Place	GDX	3727.4	0.0988	1.6889
		Q-Newton	135.1	0.0988	1.2378
		SCG	398.2	0	1.4307
	Classification Therm	GDX	40795	0	0.3565
		Q-Newton	52	0	0.4072
		SCG	871.9	0	0.3036
0 to 2	Classification Place	GDX	4117.4	0.0618	0.899
		Q-Newton	46.7	0.1981	1.5721
		SCG	390.3	0	0.9887
	Classification Therm	GDX	92787	0	0.0857
		Q-Newton	89.9	0	0.2833
		SCG	1493.1	0	0.1128

## CHAPTER IV

### CONCLUSIONS AND FUTURE WORK

I have analyzed from these results, how thermometer coding shows much more CP effect than place coding by comparing them with different data, learning rules, transfer functions and network structures. Thermometer coding converged much slower but had much smaller MSE and generalization MSE than place coding. For place code the error values didn't change much even with more learning. This also can mean that thermometer coding has an ability to overcome small local minima while place coding doesn't. For place coding, noise didn't affect the result much, whereas added noise increased the MSE and generalization MSE by a great measure for thermometer coding. This added noise may have changed the shape of the input samples, which slowed down learning and increased errors for thermometer coding.

As for comparison of the different transfer functions, this result showed the form of the transfer function is closely related to the CP effect since transfer functions more similar to the input representation gave better results. The input data had rectangular form so transfer functions (e.i., the LOGSIC and TANSIG transfer functions) with rectangular form results in smaller error than transfer functions (e.i., the RADBAS and TRIBAS transfer functions) with triangular shape.

The more hidden nodes the network has, the better we can see the CP effect of thermometer coding by checking the fact that for thermometer coding, the MSE and Generalization MSE keep decreasing in proportion to the number of hidden nodes, while the networks with place coding didn't have much change with different numbers of hidden nodes. This can be explained by the role of weights in a network as storage places for features of inputs. When a network tries to extract features from the given inputs and

to store them as weights, it is easy to see that more storage places can store more detailed information about input patterns. However if the input pattern itself doesn't have enough features to be stored, extra storage places don't mean much for this representation.

More hidden layers in a network hindered the CP effect for thermometer coding while it didn't affect place coding. The first hidden layer from the input layer does all the job of extracting features from the input pattern. Adding more hidden layers than one between the input layer and the output layer will minimize the effort of the first hidden layer.

Thermometer coding also gave the networks better MSE and better generalization MSE with real-world data. This is no surprise since we can think real-world data as theoretical data with noise, which already has been discussed.

Because of the particular way the weight updating method works in the hidden layers, some representations of input data work better than others. When similar inputs have more common factors in their coding representation, an ANN is expected to show better mapping result with its outputs than when there are not many common units between similar inputs. For example, with thermometer coding a line of length "4" (11110000) will share those four bits with a line of length "5" (11111000), whereas a place coding (00010000 for "4" and 00001000 for "5") would not preserve any similarity between these two lines. The place code is more arbitrary and the thermometer code more iconic, in that with the thermometer codings, some of the analog structure is preserved through multi-unit constraints.

Further work could usefully investigate the construction of a neural network to encode its input by itself with the result of the training [2] and comparison of thermometer coding and place coding with different optimization algorithms like genetic algorithms and simulated annealing. Comparison of different types of codings other than thermometer coding and place coding like interpolation, continuous thermometer, proportional coarse coding also could enrich these results.



## REFERENCES

- [1] Cowan, J. D. and Sharp, D. H. (1988) *Artificial Intelligence*, Journal of the American Academy of Arts and Sciences Vol.117, No.1. pp. 11-57
- [2] Hancock, P. J. (1989) *Data Representation in Neural Nets*. Connectionist Models Summer School, San Mateo, CA. M. Kaufmann, pp. 11-20.
- [3] Hanson, H. S. (1991) *Categorical Perception and the Evolution of Supervised Learning in Neural Nets*. In D.W. Powers & L. Reeker (Eds.), "Working Papers of the AAAI Spring Symposium on Machine Learning of Natural Language and Ontology", pp. 65-74.
- [4] Harnad, S. (1987) *Categorical Perception: The Groundwork of Cognition*. New York: Cambridge University Press.
- [5] Harnad, S. (1991) *The Symbol Grounding Problem and Categorical Perception*. AAAI Symposium on Symbol Grounding: Problem and practice, pp. 65-73.
- [6] Harnad, S. (1994) *Symbol Processors and Connectionist Network Models in Artificial Intelligence and Cognitive Modelling: Steps Toward Principled Integration*. Learned Categorical Perception in Neural Nets: Implications for Symbol Grounding. In: V. Honavar & L. Uhr (Eds.), Academic Press.
- [7] Haykin, S. (1944) *Neural Networks: A Comprehensive Foundation*. IEEE Computer Society Press, pp. 1-40, 121-128, 138-184.
- [8] Hush, D. R. and Horne B. G. (1993) *Progress in Supervised Neural Networks*. IEEE Signal Processing Magazine, pp. 8-39.
- [9] Karanjit, S. (1997) *Inside Java*. Indianapolis IN. New Riders.
- [10] Lippmann, R. P. (1987) *An Introduction to Computing with Neural Nets*. IEEE ASSP Magazine, Vol.4. No.2, pp. 4-22.

- [11] Littlefield, B. and Hanselman, D. (1997) *The Student Edition of MATLAB*. the math works Inc, pp. 42-149.
- [12] Martin, L. (1997) *Java Programming with Visual J++*. M&T Books, New York, N.Y.
- [13] McClelland, L. and Rumelhart, D. E. (1988) *Explorations in Parallel Distributed Processing*. MIT Press, Cambridge, Mass. pp. 121-141.
- [14] Michael, M. (1996) *Java Unleashed*. Sams.net, Indianapolis, IN.
- [15] Russell, S. and Norvig, P. (1995) *Artificial Intelligence a Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey.
- [16] Street, W. N. (1998) *Proceedings of the Fifteenth International Conference on Machine Learning*. A Neural Network Model for Prognostic Prediction. In J. Shavlik, editor, Morgan Kaufmann, San Francisco, CA. pp. 540-546.
- [17] Vemuri, V. R.(1992) *Artificial Neural Networks: Concepts and Control Applications*. IEEE Computer Society Press, pp. 1-9.
- [18] Widrow, B. and Stearns, S. D. (1985) *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, N.J. pp. 1-54.

APPENDICES

## Appendix A

### A GLOSSARY

ANN: Artificial Neural Network

BP: Back Propagation

CP: Categorical Perception

Cross-validation method: a method to measure the generalization performance of the network on a data set different from the one used for training the network

GDX : Gradient Descent Optimization

LOGSIG : Log sigmoid transfer function

Momentum: a constant value in the delta rule to increase the learning rate while still keeping the network stable

MSE: Mean Squared Error

Place coding : a coding in which there is only one output of "1" where the value range ends, otherwise output is "0".

Thermometer coding: a coding in which each unit remains as "1" if the value is equal to or less than the value range

RADBAS : Radial basis transfer function

SCG : Scaled Conjugate Gradient Optimization

TANSIG : Hyperbolic tangent sigmoid transfer function

TRIBAS : Triangular basis transfer function

Q-Newton : BFGS quasi-Newton Optimization

## Appendix B

### DATA SPECIFICATION

The following real world data is presented in this appendix.

B.1. Glass Identification Database

B.2. Balance Scale Weight & Distance Database

B.3. Hayes-Roth & Hayes-Roth (1977) Database

B.4. Wisconsin Prognostic Breast Cancer (WPBC)

## B.1. Glass Identification Database

### 1. Title: Glass Identification Database

### 2. Sources:

- (a) Creator: B. German
  - Central Research Establishment
  - Home Office Forensic Science Service
  - Aldermaston, Reading, Berkshire RG7 4PN
- (b) Donor: Vina Spiehler, Ph.D., DABFT
  - Diagnostic Products Corporation
  - (213) 776-0180 (ext 3014)
- (c) Date: September, 1987

### 3. Past Usage:

- Rule Induction in Forensic Science
  - Ian W. Evett and Ernest J. Spiehler
- Central Research Establishment
  - Home Office Forensic Science Service
  - Aldermaston, Reading, Berkshire RG7 4PN
- Unknown technical note number (sorry, not listed here)
- General Results: nearest neighbor held its own with respect to the rule-based system

### 4. Relevant Information:n

Vina conducted a comparison test of her rule-based system, BEAGLE, the nearest-neighbor algorithm, and discriminant analysis. BEAGLE is a product available through VRS Consulting, Inc.; 4676 Admiralty Way, Suite 206; Marina Del Ray, CA 90292 (213) 827-7890 and FAX: -3189. In determining whether the glass was a type of "float" glass or not, the following results were obtained (# incorrect answers):

Type of Sample	Beagle	NN	DA
Windows that were float processed (87)	10	12	21
Windows that were not: (76)	19	16	22

The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

### 5. Number of Instances: 214

- ### 6. Number of Attributes: 10 (including an Id#) plus the class attribute
- all attributes are continuously valued

## 7. Attribute Information:

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass: (class attribute)
  - 1 building\_windows\_float\_processed
  - 2 building\_windows\_non\_float\_processed
  - 3 vehicle\_windows\_float\_processed
  - 4 vehicle\_windows\_non\_float\_processed (none in this database)
  - 5 containers
  - 6 tableware
  - 7 headlamps

## 8. Missing Attribute Values: None

## Summary Statistics:

Attribute:	Min	Max	Mean	SD	Correlation with class
2. RI:	1.5112	1.5339	1.5184	0.0030	-0.1642
3. Na:	10.73	17.38	13.4079	0.8166	0.5030
4. Mg:	0	4.49	2.6845	1.4424	-0.7447
5. Al:	0.29	3.5	1.4449	0.4993	0.5988
6. Si:	69.81	75.41	72.6509	0.7745	0.1515
7. K:	0	6.21	0.4971	0.6522	-0.0100
8. Ca:	5.43	16.19	8.9570	1.4232	0.0007
9. Ba:	0	3.15	0.1750	0.4972	0.5751
10. Fe:	0	0.51	0.0570	0.0974	-0.1879

## 9. Class Distribution: (out of 214 total instances)

- 163 Window glass (building windows and vehicle windows)
  - 87 float processed
    - 70 building windows
    - 17 vehicle windows
  - 76 non-float processed
    - 76 building windows
    - 0 vehicle windows
- 51 Non-window glass
  - 13 containers
  - 9 tableware
  - 29 headlamps

## B.2. Balance Scale Weight & Distance Database

1. Title: Balance Scale Weight & Distance Database

2. Source Information:

(a) Source: Generated to model psychological experiments reported by Siegler, R. S. (1976). *Three Aspects of Cognitive Development*. *Cognitive Psychology*, 8, 481-520.

(b) Donor: Tim Hume (hume@ics.uci.edu)

(c) Date: 22 April 1994

3. Past Usage: (possibly different formats of this data)

- Publications

1. Klahr, D., & Siegler, R.S. (1978). The Representation of Children's Knowledge. In H. W. Reese & L. P. Lipsitt (Eds.), *Advances in Child Development and Behavior*, pp. 61-116. New York: Academic Press
2. Langley, P. (1987). A General Theory of Discrimination Learning. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production System Models of Learning and Development*, pp. 99-161. Cambridge, MA: MIT Press
3. Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press
4. McClelland, J.L. (1988). *Parallel Distributed Processing: Implications for Cognition and Development*. Technical Report AIP-47, Department of Psychology, Carnegie-Mellon University
5. Shultz, T., Mareschal, D., & Schmidt, W. (1994). Modeling Cognitive Development on Balance Scale Phenomena. *Machine Learning*, Vol. 16, pp. 59-88.

4. Relevant Information:

This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (left-distance \* left-weight) and (right-distance \* right-weight). If they are equal, it is balanced.

5. Number of Instances: 625 (49 balanced, 288 left, 288 right)

6. Number of Attributes: 4 (numeric) + class name = 5

7. Attribute Information:



1. Class Name: 3 (L, B, R)
2. Left-Weight: 5 (1, 2, 3, 4, 5)
3. Left-Distance: 5 (1, 2, 3, 4, 5)
4. Right-Weight: 5 (1, 2, 3, 4, 5)
5. Right-Distance: 5 (1, 2, 3, 4, 5)

8. Missing Attribute Values:

none

9. Class Distribution:

1. 46.08 percent are L
2. 07.84 percent are B
3. 46.08 percent are R

### B.3. Hayes-Roth & Hayes-Roth (1977) Database

#### 1. Title: Hayes-Roth & Hayes-Roth (1977) Database

#### 2. Source Information:

- (a) Creators: Barbara and Frederick Hayes-Roth
- (b) Donor: David W. Aha (aha@ics.uci.edu) (714) 856-8779
- (c) Date: March, 1989

#### 3. Past Usage:

1. Hayes-Roth, B., & Hayes-Roth, F. (1977). Concept learning and the recognition and classification of exemplars. *Journal of Verbal Learning and Verbal Behavior*, 16, 321-338.

-- Results:

-- Human subjects classification and recognition performance:

1. decreases with distance from the prototype,
2. is better on unseen prototypes than old instances, and
3. improves with presentation frequency during learning.

2. Anderson, J.R., & Kline, P.J. (1979). A learning system and its psychological implications. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 16-21). Tokyo, Japan: Morgan Kaufmann.

-- Partitioned the results into 4 classes:

1. prototypes
2. near-prototypes with high presentation frequency during learning
3. near-prototypes with low presentation frequency during learning
4. instances that are far from prototypes

-- Described evidence that ACT's classification confidence and recognition behaviors closely simulated human subjects' behaviors.

3. Aha, D.W. (1989). Incremental learning of independent, overlapping, and graded concept descriptions with an instance-based process framework. Manuscript submitted for publication.

-- Used same partition as Anderson & Kline

-- Described evidence that Bloom's classification confidence behavior is similar to the human subjects' behavior. Bloom fitted the data more closely than did ACT.

#### 4. Relevant Information:

This database contains 5 numeric-valued attributes. Only a subset of 3 are used during testing (the latter 3). Furthermore, only 2 of the 3 concepts are "used" during testing (i.e., those with the prototypes 000 and 111). I've mapped all values to their zero-indexing equivalents.

Some instances could be placed in either category 0 or 1. I've followed the authors' suggestion, placing them in each category with equal probability.

I've replaced the actual values of the attributes (i.e., hobby has values chess, sports and stamps) with numeric values. I think this is how the authors' did this when testing the categorization models described in the paper. I find this unfair. While the subjects were able to bring background knowledge to bear on the attribute values and their relationships, the algorithms were provided with no such knowledge. I'm uncertain whether the 2 distractor attributes (name and hobby) are presented to the authors' algorithms during testing. However, it is clear that only the age, educational status, and marital status attributes are given during the human subjects' transfer tests.

5. Number of Instances: 132 training instances, 28 test instances
6. Number of Attributes: 5 plus the class membership attribute. 3 concepts.
7. Attribute Information:
  - 1. name: distinct for each instance and represented numerically
  - 2. hobby: nominal values ranging between 1 and 3
  - 3. age: nominal values ranging between 1 and 4
  - 4. educational level: nominal values ranging between 1 and 4
  - 5. marital status: nominal values ranging between 1 and 4
  - 6. class: nominal value between 1 and 3
9. Missing Attribute Values: none
10. Class Distribution: see below
11. Detailed description of the experiment:
  1. 3 categories (1, 2, and neither -- which I call 3)
    - some of the instances could be classified in either class 1 or 2, and they have been evenly distributed between the two classes
  2. 5 Attributes
    - A. name (a randomly-generated number between 1 and 132)
    - B. hobby (a randomly-generated number between 1 and 3)
    - C. age (a number between 1 and 4)
    - D. education level (a number between 1 and 4)
    - E. marital status (a number between 1 and 4)
  3. Classification:
    - only attributes C-E are diagnostic; values for A and B are ignored
    - Class Neither: if a 4 occurs for any attribute C-E
    - Class 1: Otherwise, if (# of 1's) > (# of 2's) for attributes C-E
    - Class 2: Otherwise, if (# of 2's) > (# of 1's) for attributes C-E
    - Either 1 or 2: Otherwise, if (# of 2's) = (# of 1's) for attributes C-E
  4. Prototypes:
    - Class 1: 111

- Class 2: 222
- Class Either: 333
- Class Neither: 444
- 5. Number of training instances: 132
  - Each instance presented 0, 1, or 10 times
  - None of the prototypes seen during training
  - 3 instances from each of categories 1, 2, and either are repeated 10 times each
  - 3 additional instances from the Either category are shown during learning
- 5. Number of test instances: 28
  - All 9 class 1
  - All 9 class 2
  - All 6 class Either
  - All 4 prototypes
  - 
  - 28 total

Observations of interest:

1. Relative classification confidence of
  - prototypes for classes 1 and 2 (2 instances)  
(Anderson calls these Class 1 instances)
  - instances of class 1 with frequency 10 during training and instances of class 2 with frequency 10 during training that are 1 value away from their respective prototypes (6 instances)  
(Anderson calls these Class 2 instances)
  - instances of class 1 with frequency 1 during training and instances of class 2 with frequency 1 during training that are 1 value away from their respective prototypes (6 instances)  
(Anderson calls these Class 3 instances)
  - instances of class 1 with frequency 1 during training and instances of class 2 with frequency 1 during training that are 2 values away from their respective prototypes (6 instances)  
(Anderson calls these Class 4 instances)
2. Relative classification recognition of them also

Some Expected results:

Both frequency and distance from prototype will effect the classification accuracy of instances. Greater the frequency, higher the classification confidence. Closer to prototype, higher the classification confidence.

#### B.4. Wisconsin Prognostic Breast Cancer (WPBC)

1. Title: Wisconsin Prognostic Breast Cancer (WPBC)

2. Source Information

a) Creators:

Dr. William H. Wolberg, General Surgery Dept., University of Wisconsin, Clinical Sciences Center, Madison, WI 53792  
wolberg@eagle.surgery.wisc.edu

W. Nick Street, Computer Sciences Dept., University of Wisconsin, 1210 West Dayton St., Madison, WI 53706  
street@cs.wisc.edu 608-262-6619

Olvi L. Mangasarian, Computer Sciences Dept., University of Wisconsin, 1210 West Dayton St., Madison, WI 53706  
olvi@cs.wisc.edu

b) Donor: Nick Street

c) Date: December 1995

3. Past Usage:

Various versions of this data have been used in the following publications:

(i) W. N. Street, O. L. Mangasarian, and W.H. Wolberg.  
An inductive learning approach to prognostic prediction.  
In A. Prieditis and S. Russell, editors, Proceedings of the Twelfth International Conference on Machine Learning, pages 522--530, San Francisco, 1995. Morgan Kaufmann.

(ii) O.L. Mangasarian, W.N. Street and W.H. Wolberg.  
Breast cancer diagnosis and prognosis via linear programming.  
Operations Research, 43(4), pages 570-577, July-August 1995.

(iii) W.H. Wolberg, W.N. Street, D.M. Heisey, and O.L. Mangasarian.  
Computerized breast cancer diagnosis and prognosis from fine needle aspirates. Archives of Surgery 1995;130:511-516.

(iv) W.H. Wolberg, W.N. Street, and O.L. Mangasarian.  
Image analysis and machine learning applied to breast cancer diagnosis and prognosis. Analytical and Quantitative Cytology

and Histology, Vol. 17 No. 2, pages 77-87, April 1995.

(v) W.H. Wolberg, W.N. Street, D.M. Heisey, and O.L. Mangasarian. Computer-derived nuclear "grade" and breast cancer prognosis. Analytical and Quantitative Cytology and Histology, Vol. 17, pages 257-264, 1995.

See also:

<http://www.cs.wisc.edu/~olvi/uwmp/mpml.html>

<http://www.cs.wisc.edu/~olvi/uwmp/cancer.html>

Results:

Two possible learning problems:

- 1) Predicting field 2, outcome: R = recurrent, N = nonrecurrent
  - Dataset should first be filtered to reflect a particular endpoint; e.g., recurrences before 24 months = positive, nonrecurrence beyond 24 months = negative.
  - 86.3% accuracy estimated accuracy on 2-year recurrence using previous version of this data. Learning method: MSM-T (see below) in the 4-dimensional space of Mean Texture, Worst Area, Worst Concavity, Worst Fractal Dimension.
- 2) Predicting Time To Recur (field 3 in recurrent records)
  - Estimated mean error 13.9 months using Recurrence Surface Approximation. (See references (i) and (ii) above)

#### 4. Relevant information

Each record represents follow-up data for one breast cancer case. These are consecutive patients seen by Dr. Wolberg since 1984, and include only those cases exhibiting invasive breast cancer and no evidence of distant metastases at the time of diagnosis.

The first 30 features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

A few of the images can be found at

<http://www.cs.wisc.edu/~street/images/>

The separation described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society,

pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in:

[K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

The Recurrence Surface Approximation (RSA) method is a linear programming model which predicts Time To Recur using both recurrent and nonrecurrent cases. See references (i) and (ii) above for details of the RSA method.

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WPBC/
```

5. Number of instances: 198

6. Number of attributes: 34 (ID, outcome, 32 real-valued input features)

7. Attribute information

1) ID number

2) Outcome (R = recur, N = nonrecur)

3) Time (recurrence time if field 2 = R, disease-free time if field 2 = N)

4-33) Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (local variation in radius lengths)

f) compactness ( $\text{perimeter}^2 / \text{area} - 1.0$ )

g) concavity (severity of concave portions of the contour)

h) concave points (number of concave portions of the contour)

i) symmetry

j) fractal dimension ("coastline approximation" - 1)

Several of the papers listed above contain detailed descriptions of how these features are computed.

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 4 is Mean Radius, field 14 is Radius SE, field 24 is Worst Radius.

Values for features 4-33 are recoded with four significant digits.

- 34) Tumor size - diameter of the excised tumor in centimeters
- 35) Lymph node status - number of positive axillary lymph nodes observed at time of surgery
- 8. Missing attribute values:
  - Lymph node status is missing in 4 cases.
- 9. Class distribution: 151 nonrecur, 47 recur



## Appendix C

### PROGRAM LISTING

The following program files are presented in this appendix.

- C.1. Neural network in Matlab (ann.m)
- C.2. Encoder in Matlab (encode.m)
- C.3. Neural network in Java (ann.java)

## C.1. Neural network in Matlab (ann.m)

```

%      * Author      : Beum-Seuk, Lee
%      * Created date : Oct. 21, 1998
%      * Last updated date : Oct. 31, 1998
%      * Require file : none except this file itself (Ann.m) - Matlab ver 5.2
%      * Function    : Backpropagation neural network with different transfer functions
%                    and learning rule
%      * Reference   : Duane Hanselman (1997) Matlab Version 5 User Guid, Prentice-Hall
%                    Inc.

% clear all the previous variables
clear all
% the input and output type
sCodeType = 'place';
% Transfer function in the hidden layer
%RDBAS : Radial basis transfer function
%TANSIG : Hyperbolic tangent sigmoid transfer function
%LOGSIG : Log sigmoid transfer function
%TRIBAS : Triangular basis transfer function
sTransFunc = 'TANSIG';

% Learning rule for the backpropagation
sLearnMethod = 'TRAININGDX';
%TRAINBFG : BFGS quasi-Newton backpropagation.
%TRAINCGB : Conjugate gradient backpropagation with Powell-Beale updates.
%TRAINCGF : Conjugate gradient backpropagation with Fletcher-Reeves updates.
%TRAINCGP : Conjugate gradient backpropagation with Polak-Ribiere updates.
%TRAININGDX : Gradient descent w/momentum & adaptive lr backpropagation.
%TRAINOSS : One step secant backpropagation.
%TRAINRP : RPROP backpropagation.
%TRAINSCG : Scaled conjugate gradient backpropagation.
%TRAINWB : By-weight-and-bias network training function.

12/2/98%
% Initialization of input and output data
%
% load the input and output file and save the data to the each variables
load hay_in.dat
load hay_out.dat
OriginInput = hay_in';
OriginOutput = hay_out';

% initialize the evaluation variables
GeneralizationError = 0;
OverallError = 0;
TrEpoch = 0;
TrPerf = 0;

% set the number of learning steps
iStep = 10;

% get the data number and output nodes number
[iOutputNode, iDataNum] = size(OriginOutput);
[iInputNode, iDataNum] = size(OriginInput);
iSeparate = iDataNum/iStep;

aLayer = [3 iOutputNode];
%shuffle the OriginInput and OriginOutput
for kIndex = 1:iDataNum
    iRand = round((rand(1,1)*iDataNum));
    if iRand <= 0
        iRand = 1;
    elseif iRand >= iDataNum
        iRand = iDataNum;
    end
    cInTmp = OriginInput(:, iRand);

```

```

    cOutTmp = OriginOutput(:,iRand);
    OriginInput(:,iRand) = [];
    OriginOutput(:,iRand) = [];
    OriginInput = [cInTmp OriginInput];
    OriginOutput = [cOutTmp OriginOutput];
end

%
% main function of network
%

for iIteration = 1:iStep
    iIteration
    input = [];
    output = [];
    UntrainedInput = [];
    UntrainedOutput = [];
    iTrainIndex = 1;
    iTestIndex = 1;
    % separate input data into training data and test data
    for iIndex = 1:iDataNum
        if iIndex > iSeparate*(iIteration-1) & iIndex <= iSeparate*iIteration
            UntrainedInput(:,iTestIndex) = OriginInput(:,iIndex);
            UntrainedOutput(:,iTestIndex) = OriginOutput(:,iIndex);
            iTestIndex = iTestIndex + 1;
        else
            input(:,iTrainIndex) = OriginInput(:,iIndex);
            output(:,iTrainIndex) = OriginOutput(:,iIndex);
            iTrainIndex = iTrainIndex+1;
        end
    end
    input = encode(sCodeType,input,5);
    UntrainedInput = encode(sCodeType,UntrainedInput,5);
    %
    % initialize the network
    % 1) maximum and minimum input range
    % 2) Construction of a network with 1 hidden layer with 2 hidden nodes
    % 3) transfer functions and learning rule
    %
    net = newff([min(input(:,:))-1;
max(input(:,:))+1],aLayer,{sTransFunc,'purelin'},sLearnMethod);

    %net
    net.trainParam.show = 1000;           % output the status of net every 100 epoch
    net.trainParam.epochs = 4000;        % maximum epochs to stop the training
    net.trainParam.min_grad = 5e-004;    % minimum gradient value to stop the training
the net
    net.trainParam.lr=0.1;                % Learning rate
    net.trainParam.mc=0.9;                % Momentum constant.

    % train the network with input and output on the initialized net
    [net,tr]=train(net,input,output);

    %
    % simulation the network with the untrained input (test samples)
    %
    check = sim(net,UntrainedInput);
    % get the MSE (Mean Square Error) to calculate generalization error
    iDiffer = UntrainedOutput-check; % difference
        iDiffer = iDiffer.^2; % square
        iSum = sum(iDiffer,1); % sum
        iSum = sum(iSum,2)/iOutputNode; % sum

    GeneralizationError = GeneralizationError + iSum/iSeparate;

    % print out epoch and the MSE for trained samples after each step
    TrEpoch = TrEpoch + tr.epoch(end);
    TrPerf = TrPerf + tr.perf(end);

```

```
end %iIteration
% print out overall result
%
sCodeType
sLearnMethod
sTransFunc
aLayer
TrEpoch = TrEpoch/iStep % Epoch
TrPerf = TrPerf/iStep % Error
GeneralizationError = GeneralizationError/iStep % Generalization error
```

## C.2. Encoder in Matlab (encode.m)

```

% * Author      : Beum-Seuk, Lee
% * Created date : Oct. 28, 1998
% * Last updated date : Nov. 05, 1998
% * Require file : none except this file itself (encode.m) - Matlab ver 5.2
% * Function    : Normalize and encode the array
% * Reference   : Duane Hanselman (1997) Matlab Version 5 User Guid,
%               : Prentice-Hall Inc.

function aOutput = encode(sKind,aInput,iStep)
%function aOutput = encode(sKind,aInput,iStep)
OriginInput = aInput';
iDataNum =size(OriginInput,1);
iColNum =size(OriginInput,2);
%get mean
for iIndex = 1:iColNum
    iMinTmp(iIndex) = min(OriginInput(:,iIndex));
    iMaxTmp(iIndex) = max(OriginInput(:,iIndex));
end

for iIndex = 1:iDataNum
    for jIndex = 1:iColNum
        if iMaxTmp(jIndex) == iMinTmp(jIndex)
            OriginInput(iIndex,jIndex) = OriginInput(iIndex,jIndex)*100;
        else
            OriginInput(iIndex,jIndex) =
round((OriginInput(iIndex,jIndex)-iMinTmp(jIndex))*iStep/(iMaxTmp(jIndex)-iMinTmp(jIndex))
);
        end
    end
end

Last = -ones(iDataNum,iColNum*iStep);
for kIndex = 1:iDataNum
    for lIndex = 1:iColNum
        iOriginInput = -ones(1,iStep);
        for iIndex = 1:iStep
            switch sKind
            %place
                case 'place'
                    if OriginInput(kIndex,lIndex) >= 1
                        Last(kIndex,(lIndex-1)*iStep+OriginInput(kIndex,lIndex)) = 1;
                    end
                otherwise
                    %therm
                        for jIndex = 1:OriginInput(kIndex,lIndex)
                            if jIndex >= 1
                                Last(kIndex,(lIndex-1)*iStep+jIndex) = 1;
                            end
                        end
                    end % if iKind
            end
        end
    end
end
aOutput = Last';

```

## C.3. Neural network in Java (ann.java)

```

/*
 * Author      : Beum-Seuk, Lee
 * Created date : Feb. 07, 1998
 * Last updated date : Oct. 31, 1998
 * Require file : none except this file itself (Ann.java) - JDK 1.0
 * Function : Backpropagation neural network with momentum constant
             Neurons are trained by Gradient descent method
             1. the number of Hidden layers can be changed
             2. the number of Hidden nodes for each Hidden layers can
                be changed with different values
             3. Rescaling the input values by using
                Value = (Value - Min)/(Max - Min)
             4. Cross evaluation
                Seperating test sets with training sets
             5. Back propagation learning
             6. It can have more than one Error functions
             7. Shuffling training set at every epoch
             8. The Error function is Mean-Square function
 * Global variables :
             1. cNode   : a class with information of each node
             2. cWeight : a class with information of weights of each connection
             3. cIO     : a class for input and output nodes
             4. Ann     : the main java applet with all neural network function
 * Reference :
             1. Russell Norvig (1995) Artificial Intelligence : A Modern Approach
                Prentice-Hall, Inc.pp. 563-584
             2. Lippmann R. P. (1987) An Introduction to Computing with Neural Nets.
                IEEE ASSP Magazine, Vol.4. No.2, pp. 4-22.
 */
// include some library files
import java.applet.*;
import java.awt.*;
import java.lang.Math;
import java.io.*;
import java.util.Vector;
import java.lang.Math;
//class for each node
class cNode
{
    // synapse(weight) toward input layers
    Vector vSynapse = new Vector(0);
    //current output node value ai = g(ini) = g(dSumOfWa)
    double dSoma = 0.0; // aj = step0(Wjk*ak)
    //the error factor
    double dError = 0.0;
    // the sum of all the multiplication of weight and node values
    double dSumOfWa = 0.0; // Sum of Wjk*ak
}
// class for weight between two nodes
class cWeight
{
    // current weight
    double dWeight = 0.0;
    // change of the weight
    double dChangeWeight = 0.0;
    // constructor
    public cWeight(double dWeight)
    {
        this.dWeight = dWeight;
        this.dChangeWeight = 0.0;
    }
}

/ set of input and output vector
class cIO
{

```

```

boolean bTest = true;
boolean bRecur = false;
int iRecur = 0;
Vector vIn = new Vector(0);
Vector vOut = new Vector(0);
}
// main class to perform the neural network
public class Ann extends Applet
{
    //sum of the test errors
    private static double dTestError = 0;
    //average ephoche
    private static int iAveEpoche = 0;
    //the number of item to be split to training and test set
    private static int iSplitItems = 5;
    //total average error of the test sets
    private static double dTotalError = 0;
    // hidden layer vector which contains all the hidden layer
    private static Vector vNumOfHidNode = new Vector();
    // count number for error limit
    private static int iCumulativeCount = 0;
    // current error value which is Average Sum of square of all errors at the output nodes
    private static double dError = 0;
    // contain all the set of input and output values
    private static Vector vIO = new Vector();
    // the number of test set in all the set
    private static int iMaxGroupSet = 0;
    //traing set or test set
    private static boolean bTraining = false;
    // graphic object
    private static Graphics m_Graphics;
    // Base path of directory where Ann.html resides
    private static String sBase = "";
    // data file name
    private static String m_FileName = "";
    // the number of input nodes
    private static int m_iInput = 0;
    // the number of output nodes
    private static int m_iOutput = 0;
    // the number of hidden layers
    private static int m_iHiddenLayer = 0;

    // the current training set index of the training set
    public static int iCurrentSet = 0;
    // count of the every set
    public static int iCount = 0;
    // count for error graph to draw dots
    public static int iErrorCount = 0;
    //momentum for backpropagation
    public static double dMomentum = 0.0;
    //learning rate
    public static double dLearningRate = 0.0;
    // hidden layers and output layer in the network
    public static Vector vNodeLayer = new Vector(0);
    // current output vlues(Target values)
    public static Vector vOutput = new Vector(0);
    //current input values (input nodes)
    public static Vector vInput = new Vector(0);
    // Training set
    public static Vector vTrainingSet = new Vector(0);
    //test set
    public static Vector vTestSet = new Vector(0);

    // Parameter names. To change a name of a parameter, you need only make
    // a single change. Simply modify the value of the parameter string below.
    //-----
    private final String PARAM_sMyFileName = "sMyFileName";
    private final String PARAM_iMyInput = "iMyInput";

```

```

private final String PARAM_iMyOutput = "iMyOutput";
private final String PARAM_iMyHiddenLayer = "iMyHiddenLayer";
private final String PARAM_iMyHiddenNode = "iMyHiddenNode";
private final String PARAM_iSeperate = "iSeperate";
public Ann()
{
}
// applet information
public String getAppletInfo()
{
    return "Name: Ann\r\n" +
        "Author: Rom\r\n" +
        "Created with Microsoft Visual J++ Version 1.1";
}
/*
    function : get variable from the HTML parameter tags
    parameters : none
    return value : each information
*/
public String[][] getParameterInfo()
{
    String[][] info =
    {
        { PARAM_sMyFileName, "String", "Parameter description" },
        { PARAM_iMyInput, "int", "Parameter description" },
        { PARAM_iMyOutput, "int", "Parameter description" },
        { PARAM_iMyHiddenLayer, "int", "Parameter description" },
        { PARAM_iMyHiddenNode, "String", "Parameter description" },
        { PARAM_iSeperate, "int", "Parameter description" },
    };
    return info;
}
/*
    function : initialize the applet
    parameters : none
    return value : none
*/
public void init()
{
    String param;
    //get parameters from html file
    param = getParameter(PARAM_sMyFileName);
    if (param != null)
        m_FileName = param;

    param = getParameter(PARAM_iMyInput);
    if (param != null)
        m_iInput = Integer.parseInt(param);

    param = getParameter(PARAM_iMyOutput);
    if (param != null)
        m_iOutput = Integer.parseInt(param);

    param = getParameter(PARAM_iMyHiddenLayer);
    if (param != null)
        m_iHiddenLayer = Integer.parseInt(param);

    param = getParameter(PARAM_iMyHiddenNode);
    //parse the number of nodes in each hidden layer
    if (param != null)
    {
        String sAll = param;
        String sCur = "";
        int iIndex = 0;
        for(int i = 0; i < m_iHiddenLayer; i++)
        {
            iIndex = sAll.indexOf(',');
            if(iIndex== -1)

```



```

        iIndex = sAll.length();
        sCur = sAll.substring(0,iIndex);
        if(sCur.length()==0)
            return;
        if(iIndex != sAll.length())
            sAll = sAll.substring(iIndex+1,sAll.length());
        else
            sAll = "";
        vNumOfHidNode.addElement(new Integer(sCur));
    }
}
param = getParameter(PARAM_iSeperate);
iSplitItems = Integer.parseInt(param);
//get the base address of the html for file access
sBase = "+getDocumentBase();
int iBase = sBase.lastIndexOf('/');
sBase = sBase.substring(0,iBase+1);
iBase = sBase.indexOf('/');
sBase = sBase.substring(iBase+1,sBase.length());
resize(700, 340);
}
/*
    function : called when the applet is deleted
    parameters : none
    return value : none
*/
public void destroy()
{
}
/*
    function : paint the applet
when the applet starts, it
will call this function to start main function
    parameters : Graphics class
    return value : none
*/
public void paint(Graphics g)
{
    if(!bTraining)
    {
        m_Graphics = g;
        main();
    }
}
/*
    function : called when the applet starts
    parameters : none
    return value : none
*/
public void start()
{
}
/*
    function : called when the applet stops
    parameters : none
    return value : none
*/
public void stop()
{
}
/*
    function : main program of learning process
    parameters : none
    return value : none
*/

```

```

public static void main()
{
    //get data from input data file
    if(!bGetData())
        return;
    //do for loop until get to the last one
    for(int i = 0;i<iMaxGroupSet;i++)
    {
        // seperate set of data to training and test set
        vSeperateTrainigAndTestSet(i);
        while(true)
        {
            //shuffle the training set after training of all the set
            if(!bGetTraininSet())
                break;
            //foward
            vRunNetwork();
            //backward propagation
            if(!bTraining)
                vUpdateWeight();
            //calculate error
            vErrorCalculate();
            //print out the result
            if(bTraining||iCurrentSet==vTrainingSet.size())
                vPrintOutput();
        }
    }
    //for
    iAveEpoche = iAveEpoche /iMaxGroupSet;
    dTotalError = dTotalError/iMaxGroupSet;
    dTestError = dTestError/iMaxGroupSet;

    vPrintToFile("Average Test Set Error : "+dTestError);
    vPrintToFile("Average Training Set Error : "+dTotalError);
    vPrintToFile("Average Training Set Epoche : "+iAveEpoche);
}

/*
    function : calculate error ... square, sum and average
    parameters : none
    return value : none
*/
public static void vErrorCalculate()
{
    if(vTrainingSet.size()==0)
        return;

    double tError = 0.0;
    //get the output layer
    Vector vHidLay = (Vector)vNodeLayer.elementAt(vNodeLayer.size()-1);
    // Everage of Sum(Error**2)
    for(int i = 0;i<vHidLay.size();i++)
    {
        double tmpError =((Double)vOutput.elementAt(i)).doubleValue()
            - ((cNode)vHidLay.elementAt(i)).dSoma;
        tmpError = Math.pow(tmpError,2);
        tError += tmpError;
    }
    dError += tError;
}

/*
    function : seperating training set and test set
    parameters : the index of the part to seperated
    return value : none
*/
public static void vSeperateTrainigAndTestSet(int iIndex)
{
    //initialize some values for a new learning
    vInitValues();
}

```

```

//seperating from the index until get the set of size of iMaxGroupSet
for(int i=0;i<vIO.size();i++)
{
    cIO io = new cIO();
    io.vOut = (Vector)((Vector)((cIO)vIO.elementAt(i)).vOut).clone();
    io.vIn = (Vector)((Vector)((cIO)vIO.elementAt(i)).vIn).clone();
    if(i >= (iIndex * iSplitItems)&& i < ( (iIndex+1) * iSplitItems))
        vTestSet.addElement(io);
    else
        vTrainingSet.addElement(io);
}
}
}
/*
function : get the training set data from a file
parameters : none
return value : return true when there is no error in reading the file
*/
public static boolean bGetData()
{
    m_FileName = sBase+m_FileName;

    try
    {
        //check if there exist the data file
        if(new File(m_FileName).exists())
        {
            RandomAccessFile raf = new
RandomAccessFile(m_FileName, "r");

            String sFile = "";
            String sCur = "";
            int iIndex = 0;
            while((sFile = raf.readLine())!=null)
            {
                //if no input or malformat, quit
                if(sFile.length()<2)
                    break;

                cIO io = new cIO();

                //get input values
                for(int i = 0;i < m_iInput;i++)
                {
                    iIndex = sFile.indexOf(' ');
                    if(iIndex==-1)
                        iIndex = sFile.length();
                    sCur = sSubString(sFile,0,iIndex);
                    if(sCur.length()==0)
                        return false;
                    if(iIndex != sFile.length())
                        sFile =
sSubString(sFile, iIndex+1, sFile.length());

                    else
                        sFile = "";
                    io.vIn.addElement(new Double(sCur));
                }

                //get output values
                for(int i = 0;i < m_iOutput;i++)
                {
                    iIndex = sFile.indexOf(' ');
                    if(iIndex==-1)
                        iIndex = sFile.length();
                    sCur = sSubString(sFile,0,iIndex);
                    if(sCur.length()==0)
                        return false;
                    if(iIndex != sFile.length())

```

```

sSubString(sFile, iIndex+1, sFile.length());

Double(sCur));

sFile =
else
sFile = "";
io.vOut.addElement(new

}

vIO.addElement(io);

} //while
raf.close();
} //exist
else
{
System.err.println(m_FileName+" Not found!");
return false;
}
} //try
catch (FileNotFoundException e)
{
System.err.println( e);
}
catch (IOException e)
{
System.err.println( e);
}
//calculate the iMaxGroupSet by dividing it by 5
if(iSplitItems!=0)
iMaxGroupSet = vIO.size() / iSplitItems;
else
iMaxGroupSet = 1;

if(iMaxGroupSet==0)
iMaxGroupSet = 1;

return true;
}
/*
function : reset the input and output value in scale
parameters : vec is the vector to be scaled.... Test or Training set
return value : none
*/
public static void vScaleReset(Vector vec)
{
if(vec.size()==0)
return;
Vector vMax = new Vector();
Vector vMin = new Vector();
int iInputSize = ((cIO)vec.elementAt(0)).vIn.size();
//initialize the max as the smallest num and min as the biggest one
for(int i = 0; i<iInputSize; i++)
{
vMax.addElement(new Double(Double.MIN_VALUE));
vMin.addElement(new Double(Double.MAX_VALUE));
}
//get the max and min for each column
for(int i = 0; i<vec.size(); i++)
{
cIO io = (cIO)vec.elementAt(i);
for(int j = 0; j<io.vIn.size(); j++)
{
double dTmp = ((Double)io.vIn.elementAt(j)).doubleValue();
if( dTmp > ((Double)vMax.elementAt(j)).doubleValue())
vMax.setElementAt(new Double(dTmp), j);
else if( dTmp < ((Double)vMin.elementAt(j)).doubleValue())
vMin.setElementAt(new Double(dTmp), j);
}
}
}

```

```

} //for i

//rescale the input values
//get the max and min for each column
for(int i = 0; i < vec.size(); i++)
{
    cIO io = (cIO)vec.elementAt(i);
    double dMin = 0;
    double dMax = 0;
    double dVal = 0;
    for(int j = 0; j < io.vIn.size(); j++)
    {
        dVal = ((Double)io.vIn.elementAt(j)).doubleValue();
        dMin = ((Double)vMin.elementAt(j)).doubleValue();
        dMax = ((Double)vMax.elementAt(j)).doubleValue();
        if((dMax-dMin) != 0)
            dVal = (dVal - dMin)/(dMax-dMin);

        io.vIn.setElementAt(new Double(dVal), j);
    }
    vec.setElementAt(io, i);
}
}
/*
function : initiate values
parameters : none
return value : none
*/
public static void vInitValues()
{
    vNodeLayer.removeAllElements();
    vInput.removeAllElements();
    vOutput.removeAllElements();
    vTestSet.removeAllElements();
    vTrainingSet.removeAllElements();

    iCumulativeCount = 0;
    iCount = 0;
    iCurrentSet = 0;
    iErrorCount = 0;

    bTraining = false;

    dLearningRate = 0.1;
    dMomentum = 0.9;

    int iMaxInput = m_iInput;//3;
    int iLastNodeSize = m_iOutput;//7;

    int iNodeLayer = m_iHiddenLayer+1;
    int iNode = 0;

    //init input layer
    for(int i = 0; i < iMaxInput; i++)
    {
        vInput.addElement(new Double(0.0));
    }
    //the bias node
    vInput.addElement(new Double(0.0));

    //init output layer
    for(int i = 0; i < iLastNodeSize; i++)
    {
        vOutput.addElement(new Double(0.0));
    }

    //init Nodelayer
    for(int i=0; i<iNodeLayer; i++)

```

```

    {
        Vector vTempNode = new Vector(0);
        //for the output
        if(i==(iNodeLayer-1))
            iNode = iLastNodeSize;
        else
            iNode = ((Integer)vNumOfHidNode.elementAt(i)).intValue();
        for(int j = 0;j<iNode;j++)
        {
            cNode cTempNode = new cNode();
            //update the vSynapse
            //the first Node layer set the synapse with input nodes
            if(i==0)
                vSetSynapse(vInput.size(),cTempNode);
            else

vSetSynapse(((Vector)vNodeLayer.elementAt(i-1)).size(),cTempNode);

                vTempNode.addElement(cTempNode);
            }
            vNodeLayer.addElement(vTempNode);
        }
    }

/*
    function : run network with given inputs and weights
    parameters : none
    return value : none
*/
public static void vRunNetwork()
{
    if(vTrainingSet.size()==0)
        return;

    for(int i = 0;i<vNodeLayer.size();i++)
    {
        Vector vTmpNode = (Vector)vNodeLayer.elementAt(i);
        for(int j = 0;j<vTmpNode.size();j++)
        {
            cNode cHid = (cNode)vTmpNode.elementAt(j);
            cHid.dSumOfWa = 0;

            for(int k = 0;k<cHid.vSynapse.size();k++)
            {
                double dWeight =

((cWeight)cHid.vSynapse.elementAt(k)).dWeight;
                //the first hidden node
                if(i==0)
                    cHid.dSumOfWa +=

dWeight*((Double)vInput.elementAt(k)).doubleValue();
                else
                    cHid.dSumOfWa += dWeight*

((cNode)((Vector)vNodeLayer.elementAt(i-1)).elementAt(k)).dSoma;

                }//k
                //omit the last node of hidden layers to be updated, because
it just for bias

                if(i != vNodeLayer.size()-1 && j==vTmpNode.size()-1)
                    cHid.dSoma = 1.0;
                else
                    cHid.dSoma = dActivationFunction(cHid.dSumOfWa);
            }
        }
    }
}

```

```

        vTmpNode.setElementAt(cHid,j);
    } //j
    vNodeLayer.setElementAt(vTmpNode,i);
} //i
}
/*
function : update weights with the error values
parameters : none
return value : none
*/
public static void vUpdateWeight()
{
    double tLearningRate = dLearningRate;

    //Node layers
    for(int k = vNodeLayer.size()-1;k>-1;k--)
    {
        Vector vHidLay = (Vector)vNodeLayer.elementAt(k);
        for(int i = 0;i<vHidLay.size();i++)
        {
            cNode hid = (cNode)vHidLay.elementAt(i);
            //the last layer gets the error from the output layer
            //if it is the output layer (T-O)*g'(Ini)
            if(k == vNodeLayer.size()-1)
            {
                hid.dError = dErrorFunction(true,k,i);
            }
            // Error(i) = [Result(i) - OutPut(i)]*g'(SumOfWa(i))
            else
            {
                // Error(j) = g'(SumOfWa(j))*SumOf(W(i,j)*Error(i))
                hid.dError = dErrorFunction(false,k,i);

                for(int j = 0; j< hid.vSynapse.size();j++)
                {
                    //Wk,j = Wk,j +a *Ik*Ej
                    double dSoma = 0.0;
                    if(k==0)
                        dSoma =
((Double)vInput.elementAt(j)).doubleValue();
                    else
                        //get the node input value of toinput
                        dSoma
= ((cNode)((Vector)vNodeLayer.elementAt(k-1)).elementAt(j)).dSoma;

                    // W(j,k) = W(j,k)
                    +a*SumOf(g'(SumOfWa(j))*SumOf(W(i,j)*Error(i)))
                    cWeight wt = (cWeight)hid.vSynapse.elementAt(j);
                    double dChangeWeight = wt.dChangeWeight*dMomentum +
tLearningRate * dSoma * hid.dError;

                    wt.dWeight = wt.dWeight+dChangeWeight;
                    wt.dChangeWeight = dChangeWeight;
                    hid.vSynapse.setElementAt(wt,j);
                } //j
                vHidLay.setElementAt(hid,i);
            } //i
            vNodeLayer.setElementAt(vHidLay,k);
            tLearningRate /= 2;
        } //k
    }
}
/*
function : set the bias and synnections with weight
parameters : iNum is the Synapes
hid is the hidden node
return value : none
*/
class

```

```

public static void vSetSynapse(int iNum,cNode hid)
{
    for(int i = 0;i<iNum;i++)
    {
        //random from -1.0 to 1.0
        hid.vSynapse.addElement(new
cWeight(Math.random()-0.5)); //(newDouble((Math.random()*2.0)-1.0));
    }
}

/*
    function      : if reach the end of the testset, shuffle it
    parameters    : none
    return value  : mark bit to check if the training should be finished or not
*/

public static boolean bGetTraininSet()
{
    if(iCurrentSet >= vTrainingSet.size())
    {
        //if it finished the checking the training set,
        if(!bErrorGraph())
            return false;

        //if Trainingset no need to shuffle
        if(!bTraining)
        {
            Vector vec = new Vector(0);
            while(vTrainingSet.size()!=0)
            {
                int iIndex =
(int)(Math.random()*(vTrainingSet.size()));
                cIO test = new cIO();
                test = (cIO)vTrainingSet.elementAt(iIndex);
                vec.addElement(test);
                vTrainingSet.removeElementAt(iIndex);
            }
            vTrainingSet= new Vector();
            vTrainingSet = (Vector)vec.clone();
        }
        else
        {
            dError = 0;
            if(vTrainingSet.size()==0)
                return false;
        }

        iCurrentSet = 0;
    }
    if(vTrainingSet.size() <= iCurrentSet)
        return false;
    cIO cur = (cIO)vTrainingSet.elementAt(iCurrentSet);
    for(int i = 0;i<cur.vIn.size();i++)
    {
        vInput.setElementAt((Double)cur.vIn.elementAt(i),i);
    }
    for(int i = 0;i<cur.vOut.size();i++)
    {
        vOutput.setElementAt((Double)cur.vOut.elementAt(i),i);
    }
    iCurrentSet++;
    return true;
}

/*
    function      : print out the input and output after do net working
    parameters    : none
    return value  : none
*/

```



```

public static void vPrintOutput()
{
    if (vTrainingSet.size() == 0)
        return;

    String sTraining = (!bTraining) ? "Train : " : "Test : ";
    String st = "\r\n" + sTraining + iCount++ + "\r\nInput : ";
    String sTmp = "";
    //input nodes
    for (int i = 0; i < vInput.size() - 1; i++)
    {
        if (((Double)vInput.elementAt(i)).toString().indexOf('-') == -1)
            sTmp = ((Double)vInput.elementAt(i)).toString() + "
";
        else
            sTmp = ((Double)vInput.elementAt(i)).toString() + "
";

        sTmp += " ";
        st += sTmp.substring(0, 5) + " ";
    }

    st += "\r\nOutput Result\r\n";
    //Target values of output nodes
    Vector vLastNode =
    (Vector)vNodeLayer.elementAt(vNodeLayer.size() - 1);
    for (int i = 0; i < vOutput.size(); i++)
    {
        cNode nod = (cNode)vLastNode.elementAt(i);
        String sDouble = ((Double)vOutput.elementAt(i)).toString();
        if (sDouble.indexOf('-') == -1)
            sTmp = " " + sDouble + " ";
        else
            sTmp = " " + sDouble + " ";
        sTmp += " ";
        st += sTmp.substring(0, 5) + " " + nod.dSoma + "\r\n";
    }
    st += "\r\nError : " + dError / vTrainingSet.size();
    vPrintToFile(st);
}
/*
function      : print out a string to a default output file
parameters    : String to print out
return value   : none
*/
public static void vPrintToFile(String st)
{
    //store the value to the output file
    try
    {
        RandomAccessFile fOut = new
        RandomAccessFile(sBase + "output", "rw");
        //move the file pointer to the last of the file to append
        rather than to overwrite
        fOut.seek(fOut.length());
        fOut.writeBytes(st + "\r\n");
        fOut.close();
    } //try
    //check the exceptions
    catch (FileNotFoundException e)
    {
        System.err.println(e);
    }
    catch (IOException e)
    {
        System.err.println(e);
    }
}

```

```

    }
    /*
        function      : Activation function
        parameters    : dSum is the Sum of Weighted inputs
        return value  : double value of the result of activation function
    */
    public static double dActivationFunction(double dSum)
    {
        double dResult = 0;
        //standard sigmoid
        dResult = (1/(1+Math.exp(-dSum)));
        return dResult;
    }
    /*
        function      : Derivate Activation function
        parameters    : dSum is the Sum of Weighted inputs
        return value  : double value of the result of derivative activation function
    */
    public static double dDerivativeActivationFunction(double dSum)
    {
        double dActiveFunc = dActivationFunction(dSum);
        //g' = g(1-g)
        return dActiveFunc*(1-dActiveFunc);
    }
    /*
        function      : Error function
        parameters    : bLast means the last layer, iLayer is the layer index, iNode
is node index
        return value  : double value of the result of error function
    */
    public static double dErrorFunction(boolean bLast,int iLayer,int iNode)
    {
        double dSum = 0.0;
        Vector vToOutput = new Vector();
        vToOutput = (Vector)vNodeLayer.elementAt(iLayer);
        cNode cError = (cNode)vToOutput.elementAt(iNode);
        if(bLast)
            dSum = (((Double)vOutput.elementAt(iNode)).doubleValue() -
                cError.dSoma)*
dDerivativeActivationFunction(cError.dSumOfWa);
        else
        {
            vToOutput = (Vector)vNodeLayer.elementAt(iLayer+1);
            for(int i = 0;i<vToOutput.size();i++)
            {
                cNode hid = (cNode)vToOutput.elementAt(i);
                // Error(j) = g'(SumOfWa(j))*SumOf(W(j,i)*Error(i))
                //Sum of (Wji*Errori)
                dSum += hid.dError*
                ((cWeight)hid.vSynapse.elementAt(iNode)).dWeight;
            }
            dSum *= dDerivativeActivationFunction(cError.dSumOfWa);
        }
        return dSum;
    }
    /*
        function      : draw a dot after learning one set
        parameters    : none
        return value  : mark bit to check if this is the last graph print or not
    */
    public static boolean bErrorGraph()
    {
        drawLines();
        iErrorCount++;
        dError /= vTrainingSet.size();
        double dTmpError = dError;
        dError *= 300;//300,2000
    }

```

```

int ix = 100;
int iy = 300;
if(dError > iy)
    dError = iy;

if(!bTraining)
{
//m_Graphics.drawString(""+dTmpError,iErrorCount/4+ix+10,(int)(iy-dError+10));

m_Graphics.drawLine(iErrorCount/4+ix,(int)(iy-dError),iErrorCount/4+ix-1,(int)(iy-dError
));
}
else
    m_Graphics.drawLine(100,(int)(iy-dError),700,(int)(iy-dError));

//don't need to shuffle after finishing Trainingset
if(bTraining)
{
    dTestError += dTmpError;
    return false;
}
//if the error reach a certain value, stop and check the trainingset
//if the error reach a certain value, stop and check the trainingset
if(dTmpError < 0.001 || iErrorCount>1000)//28,40
{
    if(iCumulativeCount > 8 || iErrorCount>1000)//8
    {
        iAveEpoche += iErrorCount;
        dTotalError += dTmpError;
        dError = 0;
        bTraining = true;
        vGetTrainingSet();
        //don't need to shuffle after get a trainingset
        return true;
    }
    iCumulativeCount++;
}
else
    iCumulativeCount = 0;
dError = 0;
return true;
}
/*
function      : put the traing set into test set
parameters   : none
return value  : none
*/
public static void vGetTrainingSet()
{
    vTrainingSet.removeAllElements();
    vTrainingSet = new Vector();
    vTrainingSet = (Vector)vTestSet.clone();
    //reset the currentSet number as 0
    iCurrentSet = 0;
    m_Graphics.setColor(Color.red);
}
/*
function      : draw graph axis
parameters   : none
return value  : none
*/
public static void drawLines()
{
    m_Graphics.setColor(Color.black);

    m_Graphics.drawLine(100,0,100,304);
    m_Graphics.drawLine(100,304,700,304);
}

```

```

m_Graphics.drawString("Error",40,10);
for(int i=0;i<10;i++)
{
    m_Graphics.drawLine(95,i*30,100,i*30);
    double itmp = 10-i;
    itmp /= 10;
    m_Graphics.drawString(" "+itmp,60,i*30);
}
m_Graphics.drawString("Iteration",650,335);
for(int i=0;i<12;i++)
{
    m_Graphics.drawLine(i*50+100,304,i*50+100,310);
    int itmp = i*200;
    m_Graphics.drawString(" "+itmp,i*50+100,316);
}
}
/*
    function      : message event process
    parameters    : evt is the event class
    return value  : bit mark to propagate this error to the parent event
*/
public boolean handleEvent (Event evt)
{
    //for the dubble click, move the pushabel block while moving
    //or if the clicked point is the Me, release the traped cells from it
    if(evt.id == Event.MOUSE_DOWN)
    {
        if(evt.clickCount == 2)
        {
            dError=0;
            iCumulativeCount= 20;
        }
    }
    return false;
}
/*
    function      : Get the substring
    parameters    : sIn is the input string
                                iStart is the start
                                iEnd is the end index
index of the substring
of the substring
    return value : the substring of the input string
*/
public static String sSubString(String sIn,int iStart, int iEnd)
{
    String sTmp = "";
    for(int j = iStart;j<iEnd;j++)
        sTmp += sIn.charAt(j);
    return sTmp;
}
} // end of ANN class

```

VITA <sup>2</sup>

Beum-Seuk Lee

Candidate for the Degree of

Master of Science

Thesis: COMPARISON OF PLACE CODING AND THERMOMETER CODING  
IN NEURAL NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Chunbook, South Korea, On April 21, 1973, son of Mr. Seong-Hee Lee and Mrs. Hyeon-Duck Hong.

Education : Received Bachelor of Science Degree from Konkuk University, Seoul, South Korea in February 1996; completed the requirements for the Master of Science Degree at Oklahoma State University in May 1999.

Professional Experience: From December 1995 to August 1997 worked as Research and Development Coordinator for Choongwae Medical Corporation, Seoul, South Korea.