UNIVERSITY
of
GLASGOW

Department of Mathematics

LVQ and Kohonen nets as human, for comparing ASM generated faces

Hataikan Porncharoensin

A thesis submitted to the Faculty of Science at the University of Glasgow for the degree of Master of Science (Mathematics)

Department of Mathematics
University of Glasgow
23 January 2002

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Abstract

This thesis has three main parts; Artificial Neural Networks, the Active Shape Models, and an illlustration of combining them. Computer Calculations are in the Mathematica Language. The first part is between Chapter 1 and 4. Neural Networks can be used in many areas as shown in some examples in Chapter 1. At the beginning, some simple nets are used to solve logic functions (AND, OR). More details about AND, OR problems are in Chapter 2. This chapter is concentrated on pattern classification; Hebb Net and Perceptron. In Example 2.8, the net gave the output of Character E as both E and K. Therefore, this problem is solved by using a competitive net in Chapter 3 to select the winner. Many examples are shown in this chapter because different topological structures of a Kohonen net give different results. In addition, a Kohonen net can be applied to solve a Traveling Salesman Problem. Chapter 4 gives the details of Backpropagation Neural Nets. The output propagates the error back to the previous hidden layer, it then calculates the error, and updates the weights. The backpropagation net can be used to compress data as well. In Chapter 5, the general idea of Active Shape Models is given. This method can be used to generate new shapes, which are similar to the original shapes yet of good variety. The selected shapes are hands and faces. Finally, this method is used in Chapter 6 where we compare Learning Vector Quantization, Kohonen nets and human observation, for grouping hand and face shapes.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor, Dr Stuart G. Hoggar, for all his patient advice, support, and encouragement since I had arrived to Glasgow. I feel grateful to Professor S. J. Pride for his help and support. Special mention to my flatmates, for taking photographs and encouraging me. I would like to give my pleasure to all nice friends ensuring that I had really special time in Glasgow. I am thankful for Development and Promotion for Science and Technology talents project of Thailand (DPST) for financial support through a scholarship, as well as the Department of Mathematics, University of Glasgow, for providing funding to attend conferences and facilities. Finally I would like to thank my parents for their encouragement throughout my study at Glasgow Unviersity.

CONTENTS

PART 1 ARTIFICIAL NEURAL NETWORKS

PART 3 Illustrative Projects

PART 1 ARTIFICIAL NEURAL NETWORKS

CHAPTER 1 Introduction

In the past many researchers have tried to understand the functions of the brain. For the last sixty years, they have been trying to imitate the brain's strengths to create models and use them for solving certain problems. These models, attempting to mimic the biological principles of the central nervous system, are called *Artificial Neural Networks*. A general goal of neural network models is that they should be able to learn, and adapt their action with training or experience. In fact the aim of using a neural network is to produce the "right" output when presented with an input.

McCulloch and Pitts published the first mathematical model of the biological neuron in the 1940s [McClulloch&Pitts, 1943]. They endeavored to understand the action of neural networks in terms of logic. Research in neural networks stopped in the 1960s. During these few years, Kohonen [Kohonen, 1982] had continued working on neural networks and created his own patterns. In the 1970s there was an exclusive-or (XOR) problem, which researchers could not solve using a single layer net [Rumelhart, Hinton&Williams, 1986a, 1986b; McClelland & Rumelhart, 1988]. Eventually, they could solve this problem by using the back-propagation net.

The present thesis provides a survey of some important types of Neural Network (Part 1), and an illustrative project that compares human and machine classification of hand and face shapes. In Part 1, Chapter 2, we see how even the simple perceptron can be used in pattern classification. In Chapter 3 we study Neural Networks based on competition, where we introduce the Kohonen self-organizing nets, and Learning Vector Quantization (LVQ). Our applications include the famous Travelling Salesman Problem. Chapter 4 considers Back Propagation nets, in which the difference between

desired and actual output generates a correction backwards through the net. We show how this type can be used in Data Compression.

In Part 2, we introduce principal component analysis and thence the statistical shape generation methods of [Cootes, 1995]. We use this to generate from original data of hands and faces, a variety of hand configurations and facial expressions. Then we apply Kohonen and LVQ nets to group these shapes. Finally, we compare the results and discuss which kind of net agrees better with human intuition in our (limited) data. A quantitative measure we propose gives Kohonen nets the first place.

1.1 Neurons

Each neuron is composed of *dendrites*, the *cell body* (*soma*), and an *axon*. Dendrites receive signals from other neurons and pass these signals into the soma as an accumulation (sometimes reduction) of potential. When this reaches a certain threshold the cell "fires", meaning that a wave of potential reversal (and recovery) passes out along the axon, a long fiber extending from the cell body (Figure 1.1). Meanwhile the soma potential falls back, before starting to rise again as new signals arrive. The result is a varying rate of firing.

The axon ends with branches, each of which leads to a synaptic gap (Figure 1.2) which passes on a modified signal to a dendrite of another cell. A neuron typically has many dendrites but only a single axon.

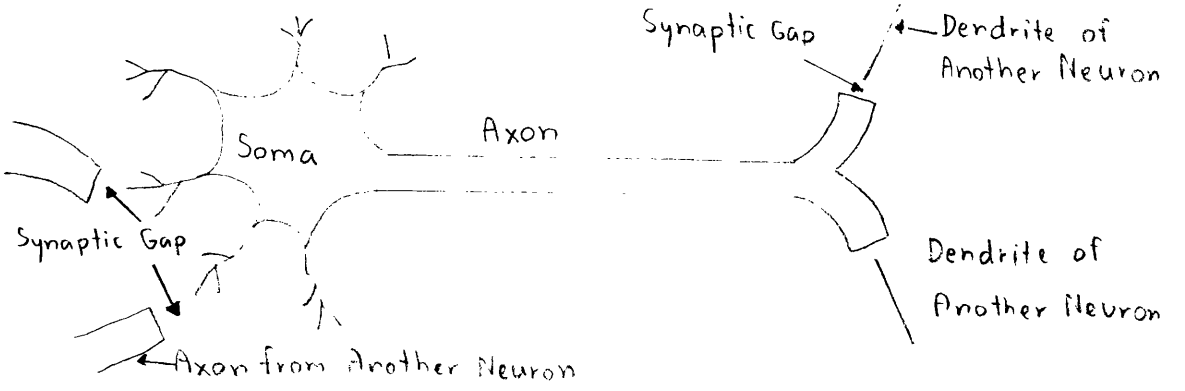


Figure 1.1 A biological neuron has three types of components (concerned with an artificial neuron): its soma, dendrites, and axon.

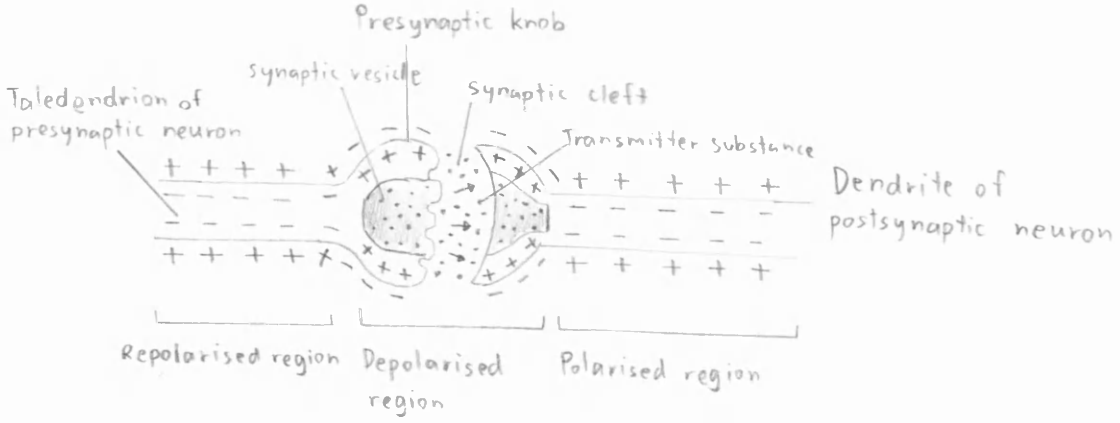


Figure 1.2 How an Axon transfers a signal to a dendrite via the synaptic gap. Hermann Von Helmboltz (1821-1894) found that this signal is not the same as electricity because it is very slow. The impulse can move because of an electrochemical reaction, causing a potential difference to travel along the fiber.

1.2 Neural Networks

Neural Networks, or **Artificial Neural Networks** (**ANNs**), are derived from the idea of biological neural networks. This means that, for example corresponding to cells dendrites and axons we have units, input connections, and output connections respectively.

An *activation function* plays a similar role to the soma of a cell. It takes the sum of weighted input values and computes a new value- the *activation*- which gives the unit's output. To each connection is associated a quantity called its *weight* which, analogously to the synaptic gap, is used to modify the signal carried by the connection. More correspondences are given in Table 1.1 below. From here there are broadly two ways to proceed by analogy. The first, and earliest understood method, is to consider a single firing and focus on a Yes/No response of a cell to input at a given moment. In this case the activation function is a step function. This is the nature of Chapter 2.

The second alternative is to take as output an analogy to the rate of *firing*, with activation function of sigmoid type (see section 1.5). This applies in Chapter 3 onwards.

Table 1.1 The ANN features correspond to the biology of neurons.

Biology	ANN Features
Cell	Unit
Dendrite	Input Connection
Synaptic Gap	Weights
Soma	Sums weighted inputs
Firing	Applying the activation function
Axon	Output Connection

However, the use of neural networks emphasizes their computational power, rather than their ability to model biological neural systems [7]. We may say that a **neural network** is a parallel, distributed information processing structure consisting of processing elements interconnected via unidirectional signal channels (*connections*). Each processing element has a single output connection that branches ("fans out") into as many collateral connections as desired.

Architecture

The *architecture* of an ANN is the pattern of connections between units. There are three kinds of units: *Input Units*, *Output Units*, and *Hidden Units*. Input Units are units which receive signals from outside. They transmit their signals to all connected units. Output Units can be interpreted as giving the response of the net. Hidden Units are units that are neither input units nor output units. Many problems can be solved only with their aid. Usually the units are divided into a list of subsets called *layers*, with the meaning that a unit is connected only to units in other layers. Prime examples of this are the input layer and output layer. In the *feedforward* type of net a unit receives input from units in previous layer (if any) and sends output only to the next layer (if any). There are three types of Architectures defined by layers used: *Single-Layer net*, *Competitive layer net*, and *Multilayer net*, which shall be illustrated in turn.

The single-layer neural net

This is a neural net with no hidden units; or equivalently, a neural net with only one layer of weight connections. Here we may establish some typical notation for units and layers. We shall label the connection from unit X_i to Unit Y_j with its weight w_{ij} in Figure 1.3, whereas the $X_i \rightarrow Y_j$ weight in Figure 1.4 is denoted by w_i. Denoting the

output from a unit X_i by x_i and the input to unit Y by y_in, we may write

$$y\_in = \sum w_{ij} x_i = \mathbf{w}_{\cdot j} \cdot \mathbf{x}, \qquad \text{(Figure 1.3)}$$

$$y\_in_j = w_1 x_1 + w_2 x_2 + w_3 x_3, \qquad \text{(Figure 1.4)}$$

where $\mathbf{w}_{\cdot j}$ is the *j'th* column of the weight matrix $[w_{ij}]$ and $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$.

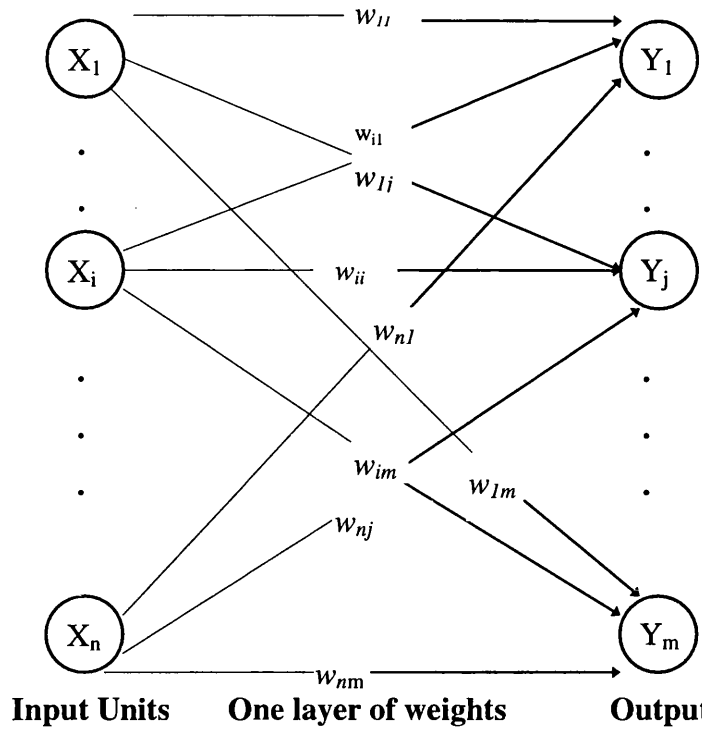


Figure 1.3 A general single-layer neural net. Only the forward direction is possible, because the X_i are input units.

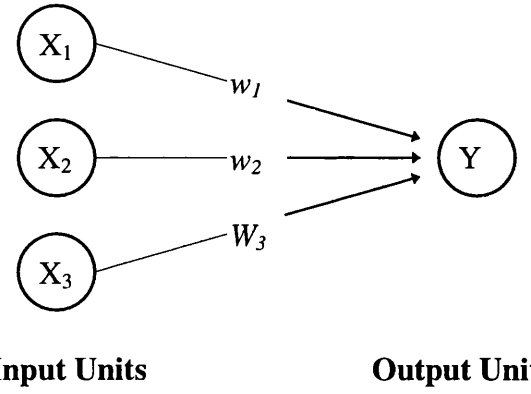


Figure 1.4 A single layer net with three input units x_i, three weights w_i, and one output unit.

The competitive neural net

This is a neural net (or subnet) in which a group of neurons competes for the right to become active (have a non-zero activation). In the most extreme (and most common)

example, the activation of the node with the largest net input is set equal to 1 and the activation of all other nodes are set equal to 0; this is often called *"winner-takes-all"*.

Fixed-weight competitive nets

Fixed weight nets are neural nets in which the weights do not change when the program runs. *MAXNET* is one type of fixed weight competitive nets. More detail is in Chapter 3.

The multilayer net

A *multilayer net* is a net which has at least one layer of hidden units. The net has at least two layers of weight connection: one between input units and hidden units and another between hidden units and output units (Figure 1.5). The number of layers in the net means the number of layers of weights.

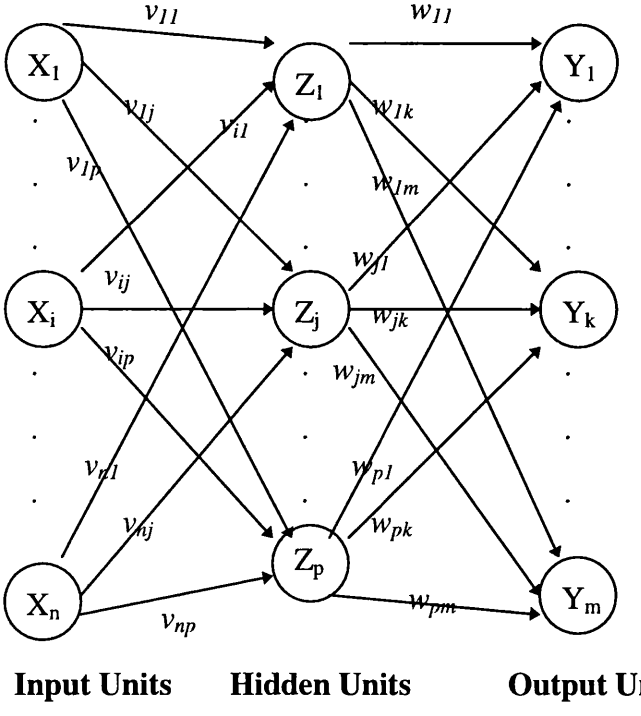


Figure 1.5 A multilayer neural net. Here the connections from X_i to Z_j and Z_j to Y_k have respective weights v_{ij} and w_{jk}.

In each layer, units normally have the same activation function (see later) and *pattern of weight connections*. In this thesis, all neural networks are fully interconnected, in the sense that every unit of a given layer is connected to every unit of any immediately preceding or following layer.

1.3 Training

Training is a method of determining the weights on the connections, also called the *learning algorithm.* In this thesis, we consider two kinds of training method - known as Supervised training and Unsupervised training.

Supervised training

Supervised training employs a sequence of training vectors corresponding to the target output. It uses a learning algorithm to improve the net until this algorithm is convergent. A pattern classification uses supervised training. There are many models using supervised training, e.g., the *Hebb rule* (the *delta rule*), *Backpropagation* (the *generalized delta rule*), and *Learning Vector Quantization* (or LVQ) as we will explain later.

Unsupervised training

Nets with this type of training are said to be self-organizing. The important thing is trying to cluster the similar patterns to the same groups. For example, each character of a, b, and c has special fonts that need a *self-organizing map* (*SOM*) to cluster the same character (not the same font) to the same group. *Kohonen* [7] discovered this model.

1.4 Activation Functions

We recall that the *activation function* f of a neural unit is a function that transforms the sum of weighted inputs $y\_in = \sum_i w_i x_i$ into the unit's activation (= output). This thesis considers three types of activation function- an *identity function*, *step functions* and *sigmoid functions*.

1. An *identity function*- It has the form $f(x) = x$ for all x. (1.1)

See Figure 1.6.

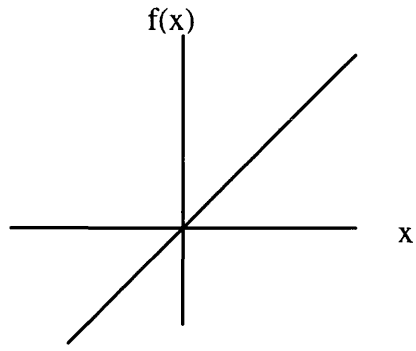


Figure 1.6 Identity function.

2. The *step function* (Figure 1.7). There are two kinds of step function (or threshold function), namely the *binary* step function and the *bipolar* step function. The binary step function has the form (for some threshold θ)

$$f(x) = \begin{cases} 1 & \text{If } x \geq \theta \\ 0 & \text{If } x < \theta, \end{cases} \qquad (1.2)$$

(The bipolar step function has some form with 1,0 replaced respectively by 1, -1). Then we can replace the threshold statement $\sum_i w_i x_i \geq \theta$ by the activation statement $f(\sum_i w_i x_i) = 1$.

Many single-layer nets use step functions as activation functions.

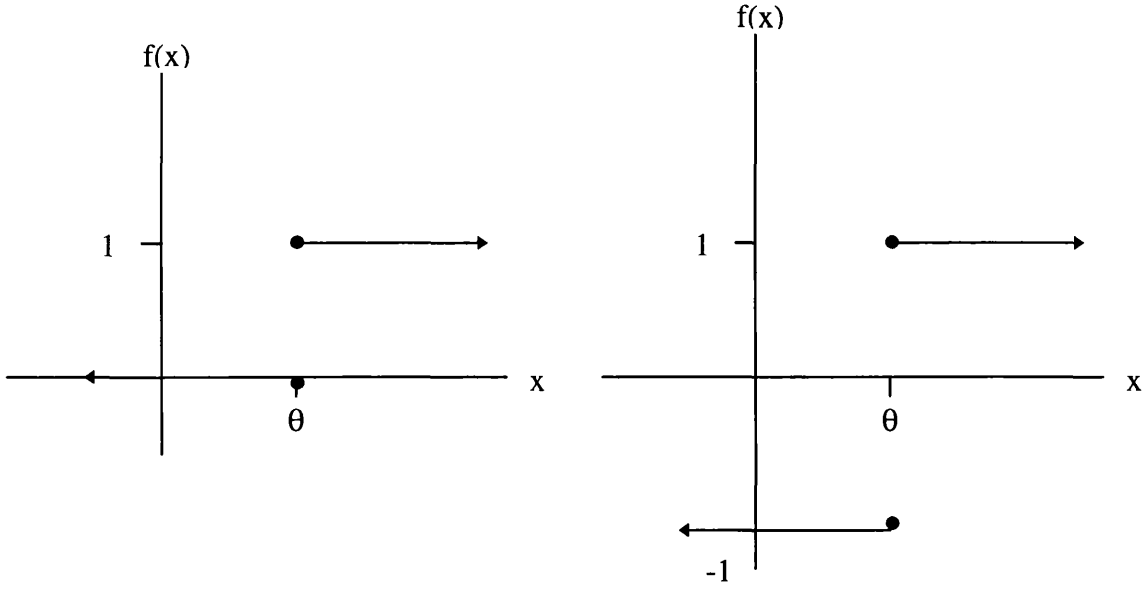


Figure 1.7 Binary step function and bipolar step function, respectively.

3. The *sigmoid function*- This is a real function giving an S-shaped curve lying between two lines. It is used in backpropagation nets (e.g. multi-layered perceptrons) where we use the gradient of the error function of each iteration step to apply the method of gradient descent (see Chapter 4). We would like the sigmoid function to be continuous, differentiable, monotonical non-decreasing. Moreover, it is expected that the activation should converge to finite maximum and minimum values. More details about the backpropagation net are given in Chapter 4.

Binary sigmoid The form of the binary sigmoid is

$$f(x) = 1 / (1+ \exp(-\sigma x), \qquad (\sigma > 0) \qquad (1.3)$$

and therefore satisfies $$f'(x) = \sigma f(x)[1-f(x)]. \qquad (1.4)$$

The case $\sigma = 1$ is called the *logistic function* and as σ increases, $f(x)$ approaches the binary step function, as shown in Figure 1.8 below.

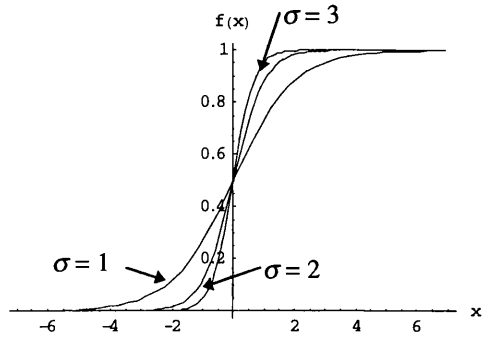


Figure 1.8 This figure shows the effect of σ when σ= 1, 2, 3, respectively σ = 3 is the most like a step function.

Bipolar sigmoid The form of the bipolar sigmoid is

$$g(x) = 2f(x)-1 = 2/[1+\exp(-\sigma x)] - 1, \qquad (1.5)$$

and so, $$g'(x)= \sigma/2\ [1+g(x)][1-g(x)]. \qquad (1.6)$$

Because the hyperbolic tangent is

$$\tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x}), \qquad (1.7)$$

Equation (1.5) may be written

$$g(x)= \tanh (\sigma x/2). \qquad (1.8)$$

If σ = 2 then this gives simply the hyperbolic tangent (tanh x). Also this sigmoid function approaches the bipolar step function as σ increases (Figure 1.9).

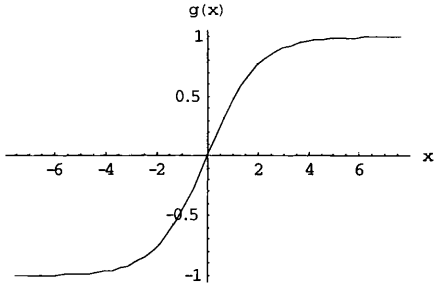


Figure 1.9 Bipolar sigmoid.

Note As σ is increases, the graph becomes more like a step function as mentioned earlier (see Figure 1.8). Notice that Equation (1.10) transforms the range (0,1) of f(x) into the range (-1,1) of a bipolar sigmoid. The binary and bipolar sigmoid functions are related respectively to corresponding step functions.

The meaning of *Epoch* is one presentation of each training pattern.

1.5 Some more History

Warren McCulloch and Walter Pitts [7] started to use the first neural networks around 1943. They found that if they used the simple logic function (*AND/OR* function) with many simple units in neural systems, they could increase computational power. They thought of the idea that if an input is greater than the threshold, the response is positive. Most of their works involved logic circuits. The name of this model is a McCulloch-Pitts neuron.

Donald Hebb [Hebb, 1949] determined the first learning law for artificial neural networks- that the strength of the connection between two units should be increased if they fired at the same time (more details in [7]). We study such nets in Chapter 2.

Johnson, Brown (1988), Anderson, and Rosenfield (1988) used ideas of Warren McCulloch and John von Neumann to improve the computer technology. Block, Minsky & Papert, and Frank Rosenblatt, found the really important model - the

perceptron which is superior to the Hebbian net in its problem solving capability (see Chapter 2).

After 1960, Bernard Widrow and Marcian Hoff found the delta rule, which is similar to the perceptron learning except that the delta rule improves weights to decrease the difference between the net output and the desired output by using the smallest mean squared error.

Teuvo Kohonen [Kohonen, 1982] developed self-organizing maps by using topological structures (linear array, rectangular grid, and hexagonal grid) for cluster units. Around 1988, he succeeded in solving the Traveling Salesman Problem by his method as mentioned earlier (see Chapter 3).

In the 1970s the development of neural networks was very slow because single layer perceptrons could not solve the XOR problem and because of the lack of a training method for multilayer nets. David Parker (1985) and LeCun (1986) then found a method, and when Parker joined with David Rumelhart, James McClelland published this idea, called *backpropagation* (see Chapter 4) [Rumelhart, Hinton, & Williams, 1986a, 1986b; McClelland & Rumelhart, 1988].

1.6 More on Applications

Neural networks can apply in many areas. One important area is that of pattern recognition where nets are used to recognize characters. Most approaches to pattern recognition attempt to copy the way a human recognizes things. For example, in Character Recognition (Chapter 3), we represent a set of alphabets and allow a net to classify them. Any misclassifications are corrected (supervised training). It learns well and gives good results. The backpropagation net (Chapter 4) can be used as a tool for recognizing, for example, handwritten zip codes.

Speech production means to read English with the correct pronunciation from a text file. One production which does this is NETtalk. We train it with

approximately 1,000 English words after which it can recognize words, and has only few errors.

Speech recognition Kohonen developed one program called "phonetic typewriter" by using a self-organizing map. This is one application from speech recognition. The topological structure is a rectangular grid or hexagonal grid. After training, it can often spell correctly.

Medical diagnosis Anderson developed one program called "Instant Physician" [7]. It stores a huge number of medical records (e.g., symptoms, diagnosis, and treatment) and the input is a set of symptoms. After training, it can advise the best diagnosis and treatment.

Optical Character Recognition (*OCR*) is the process of converting bitmapped characters into a standard font set or text format. By using a scanner and OCR software, the computer can be taught how to read. The first step is the optical scan of the test using a standard desktop scanner. During this process, the test will be divided into millions of image points each of which can be assigned a grayscale or colour value. The resolution of the scanner is measured in dpi (dots per inch) and determines the number of points the page is divided into. The result of the scanning process is an image that can be printed or modified by using any image processing software. To transfer the image into the word processor as text, we use an OCR program to find the individual characters on the image and convert them into a text format [17]. An example is *Quicktionary! English-Hebrew Translator* (Figure 1.10). It is a hand-held, scanner with an integrated OCR module.

Figure1.10 Quicktionary, an English-Hebrew Translator.

Machine Vision is very important for commercial areas, also medical science. For example, a small robot arm needs to know where the objects are so it can move them correctly. Hardware requirements are a TV camera, a very fast computer, and so on.

1.7 Implementation

Our examples are implemented in the system Mathematica with all detailed coding listed in Appendix. In context, we specify algorithms in the usual pseudocode.

Chapter 2 Pattern Classification

This Chapter is concentrated on pattern classification- a type of pattern recognition. We consider two methods of pattern classification: the *Hebb's rule* and the *Perceptron learning rule*. Note that each set of characters, which represents the same letter, is called a *class*, for example, letters A, B, C, D, E, J, and K have an 'A class', a 'B class', etc. We give the neural network input pattern, and the net produces an output which specifies the class. It is a common practice that for each class X, we have an output unit which decides whether the input belongs to class X. The unit outputs are 1 for YES, otherwise 0 if we use binary and -1 if bipolar.

To begin with, we assume that we already know the correct classification of the training vectors: that is, we can say which training vectors belong to which class (or classes). First, we consider single output nets. Having done so, we can extend to consider many groups and classify whether each pattern belongs to those classes or not.

From threshold to bias. Instead of a threshold θ we may equivalently use a so-called *bias* $b = -\theta$, which acts as a weight when the input vector is extended to $\mathbf{x} = (1, x_1, x_2,..., x_n)$. The argument is that:

$$\begin{aligned} \sum w_i x_i \geq \theta &\Leftrightarrow -\theta + \sum w_i x_i \geq 0 \\ &\Leftrightarrow b + \sum w_i x_i \geq 0 \\ &\Leftrightarrow (b, w_1, w_2,..., w_n) \,.\, (1, x_1, x_2, ..., x_n) \geq 0. \end{aligned}$$

This argument is similar for the case $\sum w_i x_i < \theta$.

We can include both bipolar and binary cases, saying that the activation now satisfies

$$f(net) = 1 \text{ if and only if } net \geq 0, \qquad (2.1)$$

where

$$net = b + \sum_i x_i w_i. \qquad (2.2)$$

Decision Boundary and *Linear Separability.*

We discuss the case $n = 2$, for simplicity so that the input pairs (x_1, x_2) represent points of the plane. The key idea is that the points satisfying $net > 0$ and those satisfying $net < 0$ lie on opposite sides of the line

$$(net =)\ b + w_1 x_1 + w_2 x_2 = 0, \qquad (2.3)$$

which we call the *decision boundary.* The opposite sides are called *decision regions.* If a decision boundary exists for a classification problem, we say the problem is *linearly separable.* What we need is an algorithm using training vectors, which will find such a boundary.

Note The equation $b + \sum_i x_i w_i = 0$ can be interpreted as the equation of a *hyperplane* in N-dimensional space, with variables x_i and parameters w_i. The sum is positive on one side and negative on the other.

2.1 Some Examples from LOGIC

The input (x_1, x_2) is a pair of logic values, with 1 for TRUE and 0 for FALSE. The output is similarly coded. The objective is to find a decision boundary which, for a given logic function, say AND, divides the True pairs (x_1, x_2) from the False.

Example 2.1 The AND function. Inputs and target outputs are shown below.

Input (x_1, x_2)	*Target output (t)*
(1, 1)	1
(1, 0)	0
(0, 1)	0
(0, 0)	0

Clearly no line passing through any of the input points can separate the points. In particular, a decision boundary cannot pass through the origin, and so must have a nonzero constant in its equation. This implies that we require a nonzero bias. It is easily seen that one possible decision boundary is the line shown in Figure 2.1, namely $x_2 = -x_1 + 3/2$ with $(b, w_1, w_2) = (-3/2, 1, 1)$.

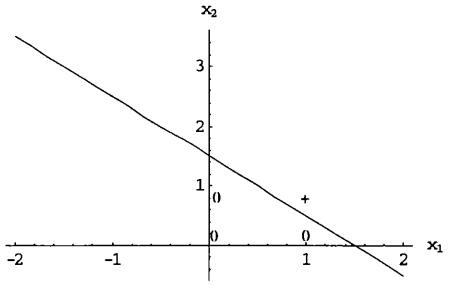


Figure 2.1 The decision boundary separates regions for Yes (+) response and No (0) responses.

However, some logic functions, e.g. XOR as shown below, are not linearly separable.

Example 2.2 Binary inputs and target outputs with XOR function.

Input (x_1, x_2)	*Target output* (t)
(1, 1)	0
(1, 0)	1
(0, 1)	1
(0, 0)	0

See Figure 2.2.

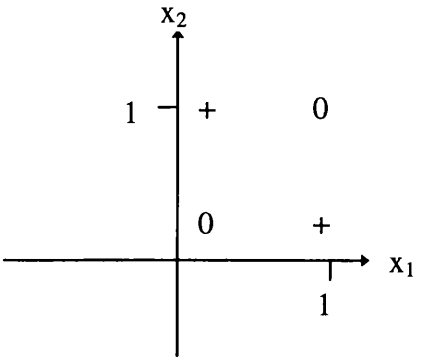


Figure 2.2 We cannot find a separating line for the XOR problem.

It is clear from Figure 2.2 that no separating line exists, so this problem cannot be solved using a single-layer neural net. The proof is also given below.

Example 2.3 Proof that the XOR problem is not linearly separable.

Suppose there is a separating line $w_1x_1 + w_2x_2 + b = 0$. Then without loss of generality we have the following table.

XOR input	Target Output	Inequality with bias b	
(x_1, x_2)			
1 1	< 0	$x_1 + x_2 + b < 0$	(I)
1 0	> 0	$x_1 + b > 0$	(II)
0 1	> 0	$x_2 + b > 0$	(III)
0 0	< 0	$b < 0$	(IV)

From (I) and (IV) $\quad x_1 + x_2 + 2b < 0$

From (II) and (III), $\quad x_1 + x_2 + 2b > 0.$

This is a contradiction, so there is no separating line for a single layer neural net.

2.2 Hebb nets

Hebb described his theory by using the term *connectionism*. He hypothesized a basic mechanism called *Hebb's rule*: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." [Hebb, 1949]. By a Hebb net we shall mean a single layer net with a training algorithm that is a slight generalisation of Hebb's original (Faussett, 1994).

Algorithm for Hebb net

Set input patterns (**x**) and target outputs (t) $\mathbf{x} = (1, x_1, x_2, \ldots, x_n)$.

Include bias in the weight vector $\mathbf{w} = (b, w_1, w_2, \ldots, w_n)$.

For each pair **x**, t do $\mathbf{w} = \mathbf{w} + t\,\mathbf{x}$. (2.4)

When we use binary target outputs with the AND function (by using the algorithm above), weights do not change so the net cannot learn, when $t = 0$. When we use bipolar target outputs, the net can learn more. However, the net still may not find a decision boundary.

Example 2.4 We compare the binary input patterns $(1, x_1, x_2)$ with the bipolar input patterns for the AND function.

(1) The binary case

x	t	Δ**w**	**w**
(binary)			0 0 0
1 1 1	1	1 1 1	1 1 1
1 0 1	-1	-1 0 -1	0 1 0
1 1 0	-1	-1 -1 0	-1 0 0
1 0 0	-1	-1 0 0	-2 0 0

Although the net can learn more because (b, w_1, w_2) has been changed, it does not separate Yes and No responses correctly (see Figure 2.3).

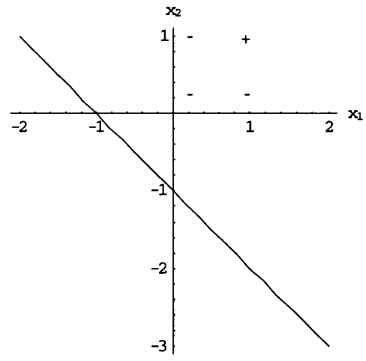


Figure 2.3 Decision boundary for the AND function by using binary inputs and bipolar target outputs.

(2) The bipolar case

The solution is to use the bipolar representation for training patterns as well as targets. The result is:

x	t	Δw	w
(bipolar)			0 0 0
1 1 1	1	1 1 1	1 1 1
1 1 -1	-1	-1 -1 1	0 0 2
1 -1 1	-1	-1 1 -1	-1 1 1
1 -1 -1	-1	-1 1 1	-2 2 2

The separating line after the first step is $x_2 = -x_1-1$, but it does not separate the (+) and (-) responses (Figure 2.4). However, separation is achieved after the third step (Figure 2.5).

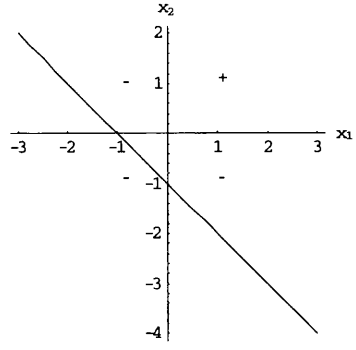


Figure 2.4 The separating line for the first input pair does not separate Yes and No response for the AND function.

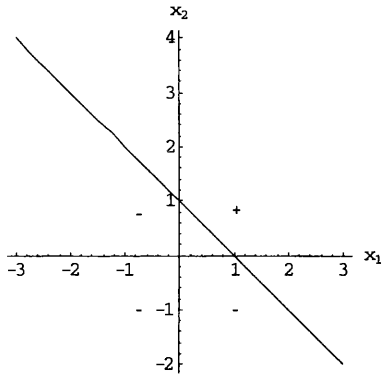


Figure 2.5 After only the third training step, we have a valid decision boundary. Therefore, on comparing the binary representation and bipolar representations, we find that the bipolar representation is more powerful than the binary representation.

2.3 The Perceptron and Pattern Recognition

The first person using the term *perceptron* was Frank Rosenblatt [10]. He used it for describing a number of different types of neural networks, including one inspired by the human retina, with unit types described as sensory, associator, and response. He also invented the *Mark I Perceptron* which functioned as a character recognizer [11]. The perceptron learning rule is more powerful than Hebb's rule because unlike the latter, it always finds correct weights when they exist. Minsky and Papert proved the *perceptron learning rule convergence theorem* in 1989.

Notation In the present context a perceptron is a single-layered net, utilizing the perceptron learning rule which we shall describe. We begin with the case of a single output, which we shall distinguish when necessary as the *simple* perceptron. As the pattern classification, we use +1 for this output belongs to, and -1 for does not belong to the class. Note that the weight vector is to include bias.

The Perceptron Learning Rule

We define a finite set of P input training vectors $\mathbf{x}(p)$, $p = 1, \ldots, P$,

and each $\mathbf{x}(p)$ has an associated target value $t(p)$, $p = 1, \ldots, P$,

which is either +1 or –1.

Set weight vector $\mathbf{w} = \mathbf{0}$. Set the learning rate α ($0 < \alpha \leq 1$). Choose $\theta > 0$ and define

$$f(y\_in) = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta. \end{cases} \qquad (2.5)$$

Repeat (epoch)

For each training pair (**x**, t) do

$y = f(\mathbf{w}.\mathbf{x})$.

If $y \neq t$ then $\mathbf{w} = \mathbf{w} + \alpha t \mathbf{x}$.

Until no change occurs during the last epoch.

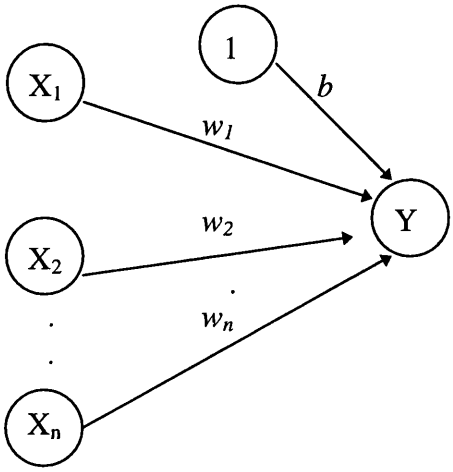


Figure 2.6 The simple architecture represents n input units (or one training n-vector) and one output unit, plus a bias (case of $w_0 = b$).

Note The variable *y_in* in Equation (2.5) is the same as *net* in Equation (2.2).

An *epoch* is a single presentation of each training pattern.

Remark (i) This algorithm can be rather slow to converge because changes to the weights are made only when errors occur, and correcting an error may result in fresh errors. However, the proof below shows that it *will* converge, and provides an upper limit on the number of iterative steps required.

(ii) Notice how this proof depends on the positive - negative symmetry gained by the choice of activation function (2.5).

(iii) The algorithm provides for a single output unit; if there are several, we simply apply the algorithm to each output unit in turn. Thus, we determine the weight matrix column by column.

The Perceptron Learning Rule Convergence Theorem

"If there is a weight vector $\mathbf{w}^*$ such that $f(\mathbf{x}(p).\ \mathbf{w}^*) = t(p)$ for all p, then for any starting vector $\mathbf{w}$, the perceptron learning rule will converge to a weight vector (not necessarily unique and not necessarily $\mathbf{w}^*$) that gives the correct response for all training patterns, and it will do so in a finite number of steps."

In other words, if the perceptron has a set of weights, which gives the desired responses to all input patterns, then the preceding adaptation rule will give a set of weights that produces the desired responses within a finite number of iterations. This set of solution weights is not unique. The perceptron is trying to find the straight line (or hyperplane in general) that separates classes.

Proof The input patterns are assumed to come from a space which has two classes; $\mathbf{F}^+$ and $\mathbf{F}^-$, where

$$\mathbf{F}^+ = \{\mathbf{x} \text{ such that the target value is } +1\}$$

and

$$\mathbf{F}^- = \{\mathbf{x} \text{ such that the target value is } -1\}.$$

We define a new training set:

$$\mathbf{F} = \mathbf{F}^+ \cup -\mathbf{F}^-,$$

where

$$-\mathbf{F}^- = \{-\mathbf{x} \text{ such that } \mathbf{x} \text{ is in } \mathbf{F}^-\}.$$

We assume, without loss of generality, that $\theta = 0$ and $\alpha = 1$ in the proof. The existence of a weight vector $\mathbf{w}^*$ for which

$$\mathbf{x} \cdot \mathbf{w}^* > 0 \qquad \text{if } \mathbf{x} \text{ is in } \mathbf{F}^+ \qquad \text{(I)}$$

and

$$\mathbf{x} \cdot \mathbf{w}^* < 0 \qquad \text{if } \mathbf{x} \text{ is in } \mathbf{F}^-, \qquad \text{(II)}$$

is equivalent to the existence of $\mathbf{w}^*$ such that

$$\mathbf{x} \cdot \mathbf{w}^* > 0 \qquad \text{if } \mathbf{x} \text{ is in } \mathbf{F} \text{ (because } \mathbf{x} \cdot \mathbf{w}^* < 0 \text{ implies } (-\mathbf{x}) \cdot \mathbf{w}^* > 0),$$

which we use instead of (I) and (II). Thus the new training set has all target values +1. Seen from this new viewpoint, the weights are updated as follows. If the response of the net is incorrect for a training input $\mathbf{x}$, we take

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \mathbf{x}.$$

We study the sequence of input training vectors for which a weight change occurs and must show that this sequence is finite. Let the initial weights be $\mathbf{w}(0)$, and the first new weights be $\mathbf{w}(1)$. If $\mathbf{x}(0)$ is the first training vector for which an error has occurred, then we update $\mathbf{w}(0)$ by using

$$\mathbf{w}(1) = \mathbf{w}(0) + \mathbf{x}(0), \quad \text{where } \mathbf{x}(0).\mathbf{w}(0) \leq 0.$$

If the error occurs again, then we update

$$\mathbf{w}(2) = \mathbf{w}(1) + \mathbf{x}(1), \quad \text{where } \mathbf{x}(1).\mathbf{w}(1) \leq 0.$$

The vector $\mathbf{x}(1)$ may be the same as $\mathbf{x}(0)$ if no errors have occurred for any other training vectors. We assume that at any stage k the weights are changed if and only if the current weights fail to produce the correct (positive) response for the current input vector, i.e., if $\mathbf{x}(k-1).\mathbf{w}(k-1) \leq 0$. We have:

$$\mathbf{w}(k) = \mathbf{w}(0) + \mathbf{x}(0) + \mathbf{x}(1) + \ldots + \mathbf{x}(k-1).$$

We have to show that k cannot be arbitrarily large. Let $\mathbf{w}^*$ be a weight vector such that $\mathbf{x} \,.\, \mathbf{w}^* > 0$ for all training vectors in $\mathbf{F}$. Let $m = \min\{\mathbf{x}.\, \mathbf{w}^*\}$, where the minimum is taken over all training vectors $\mathbf{x}$ in $\mathbf{F}$. Then

$$\mathbf{w}(k).\, \mathbf{w}^* = [\mathbf{w}(0) + \mathbf{x}(0) + \mathbf{x}(1) + \ldots + \mathbf{x}(k-1)].\, \mathbf{w}^* \geq \mathbf{w}(0).\, \mathbf{w}^* + km$$

because $\mathbf{x}(i).\, \mathbf{w}^* \geq m$ for each i, $1 \leq i \leq P$. For any vectors $\mathbf{a}$ and $\mathbf{b}$ we use the Cauchy-Schwartz inequality

$$(\mathbf{a} \,.\, \mathbf{b})^2 \leq \|\mathbf{a}\|^2 \, \|\mathbf{b}\|^2,$$

or, equivalently

$$\|\mathbf{a}\|^2 \geq (\mathbf{a} \,.\, \mathbf{b})^2 / \|\mathbf{b}\|^2 \qquad (\text{for } \|\mathbf{b}\|^2 \neq 0).$$

Hence,

$$\|\mathbf{w}(k)\|^2 \geq (\mathbf{w}(k) \,.\, \mathbf{w}^*)^2 / \|\mathbf{w}^*\|^2$$

$$\geq (\mathbf{w}(0) \,.\, \mathbf{w}^* + km)^2 / \|\mathbf{w}^*\|^2, \qquad \text{(III)}$$

where k is the number of times the weights have changed. From (III) the square length of the weight vector grows faster than k^2, but we will prove that the length cannot grow indefinitely. We have by definition that $\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{x}(k-1)$, where $\mathbf{x}(k-1).\mathbf{w}(k-1) \leq 0$, and therefore

$$\|\mathbf{w}(k)\|^2 = \|\mathbf{w}(k-1)\|^2 + 2\,\mathbf{x}(k-1).\mathbf{w}(k-1) + \|\mathbf{x}(k-1)\|^2$$
$$\leq \|\mathbf{w}(k-1)\|^2 + \|\mathbf{x}(k-1)\|^2.$$

Let $\mathbf{M} = \max\{\|\mathbf{x}\|^2$ for all $\mathbf{x}$ in the training set$\}$. Then, by repeatedly applying the above result,

$$\|\mathbf{w}(k)\|^2 \leq \|\mathbf{w}(k-1)\|^2 + \|\mathbf{x}(k-1)\|^2$$
$$\leq \|\mathbf{w}(k-2)\|^2 + \|\mathbf{x}(k-2)\|^2 + \|\mathbf{x}(k-1)\|^2$$
$$\leq \|\mathbf{w}(0)\|^2 + \|\mathbf{x}(0)\|^2 + \ldots + \|\mathbf{x}(k-1)\|^2.$$
$$\leq \|\mathbf{w}(0)\|^2 + k\mathbf{M}. \qquad \text{(IV)}$$

From (IV) the square length grow less rapidly than linearly in k.
In fact, using (III) and (IV), we have

$$(\mathbf{w}(0) \,.\, \mathbf{w}^* + km)^2 / \|\mathbf{w}^*\|^2 \leq \|\mathbf{w}(k)\|^2 \leq \|\mathbf{w}(0)\|^2 + k\mathbf{M}.$$

We assume, without loss of generality, that $\mathbf{w}(0) = \mathbf{0}$. Thus,

$$(km)^2 / \|\mathbf{w}^*\|^2 \leq k\mathbf{M},$$

or

$$k \leq \mathbf{M}\|\mathbf{w}^*\|^2 / m^2.$$

If $\mathbf{w}^*$ is multiplied by a positive scalar λ, then so is 'm' (from its definition) so this bound is actually independent of the size of $\|\mathbf{w}^*\|$. Therefore we may take $\|\mathbf{w}^*\| = 1$ and obtain the simpler formula $k \leq \mathbf{M} / m^2$.

We can use binary or bipolar input vectors provided the target vectors are in the bipolar form. We use both a fixed non-negative threshold and a bias. There are n inputs and m output targets. We begin with examples in the case of a single output.

Example 2.5 We use binary inputs and bipolar target outputs for pattern classification by a simple perceptron in the case of the AND function (note the Hebbian method, Example 2.4, failed on this case).

Input $(1, x_1, x_2)$	*Target Output* (t)
(1, 1, 1)	1
(1, 1, 0)	-1
(1, 0, 1)	-1
(1, 0, 0)	-1

Set initial weights and bias equal to zero, the learning rate α equals 1, fixed threshold equals 0.2. We recall that a Mathematica implementation is given in Appendix.

Result

We recall that y_in = **w.x**.

Epoch 1	*y_in*	*y*	**w** = {b,w_1, w_2)
1			{0,0,0}
	0	0	{1,1,1}
	2	1	{0,1,0}
	1	1	{-1,0,0}
	-1	-1	{-1,0,0}
.			
.			
.			
10			{-4,2,3}
	1	1	{-4,2,3}
	-2	-1	{-4,2,3}
	-1	-1	{-4,2,3}
	-4	-1	{-4,2,3}

Learning is complete by the end of the first complete epoch during which the weight vector **w** was unchanged. In the first epoch, not every weight vector is the same so that the net needs to learn more. Figure 2.7 shows an early failure of separation. Therefore, the net continues to be trained.

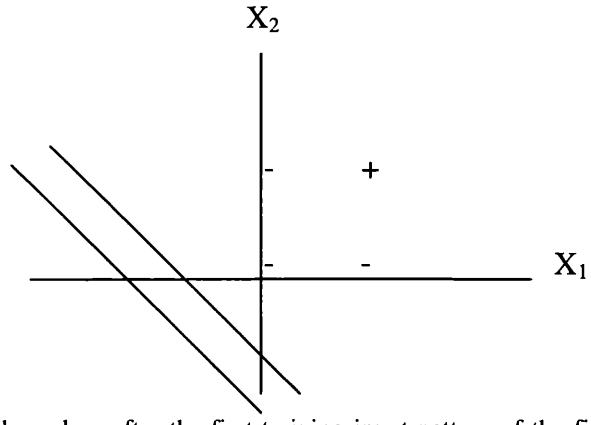


Figure 2.7 Decision boundary after the first training input pattern of the first epoch. The response of the net is correct for this pattern only.

Finally, the net converges within the tenth epoch and the weight vector solution is {2,3,-4}. Therefore, the net has found the correct decision boundaries, which are the separating lines $2x_1 + 3x_2 - 4 = 0.2$, and $2x_1 + 3x_2 - 4 = -0.2$ as shown in Figure 2.8.

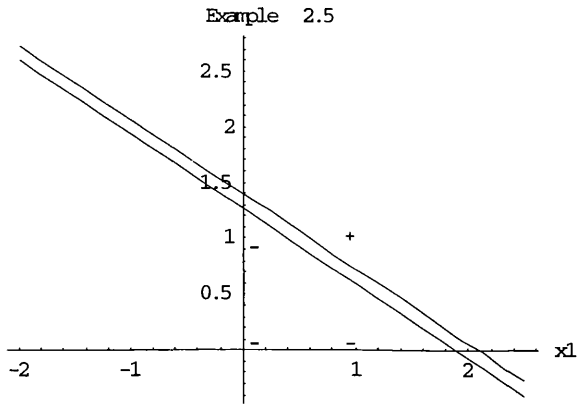


Figure 2.8 Final decision boundaries for the AND function. After the tenth epoch the net gives the correct response to every point because separating lines distinguish the YES and NO response.

It is interesting to note that if we use bipolar rather than binary input vectors, then this net is faster, using only two epochs.

Example 2.6 We use binary input vectors and bipolar target outputs as shown below. In this example an epoch consists of four training vectors. Those with target -1 are distinguished by having exactly one zero coordinate. We will find separating planes by using the perceptron net. Let initial weights and bias b equal zero, the learning rate α equals 1, and fixed threshold θ equals 0.1.

Input $(1, x_1, x_2, x_3)$	*Target* (t)
(1, 1, 1, 1)	1
(1, 1, 1, 0)	-1
(1, 1, 0, 1)	-1
(1, 0, 1, 1)	-1

Result

Epoch	*y_in*	*y*	$\mathbf{w} = \{b, w_1, w_2, w_3\}$
1			{0,0,0,0}
	0	0	{1,1,1,1}
	3	1	{0,0,0,1}
	1	1	{-1,-1,0,0}
	-1	-1	{-1,-1,0,0}
.			
.			
.			

Epoch	*y_in*	*y*	$\mathbf{w} = \{b, w_1, w_2, w_3\}$
26			{-8,2,3,4}
	1	1	{-8,2,3,4}
	-3	-1	{-8,2,3,4}
	-2	-1	{-8,2,3,4}
	-1	-1	{-8,2,3,4}

Again, the weight vectors vary during the first epoch. The net needs to learn more. After 26 epochs, the weights converge to the vector {-8,2,3,4}. This means that the net has found two decision boundary planes separating the input patterns: $2x_1 + 3x_2 + 4x_3 - 8 = \pm 0.1$, or

$$x_2 = 41/15 - 2/3\ x_1 - 4/3\ x_3,$$

and

$$x_2 = 13/5 - 2/3\ x_1 - 4/3\ x_3.$$

Figure 2.9 shows the small gap between these planes.

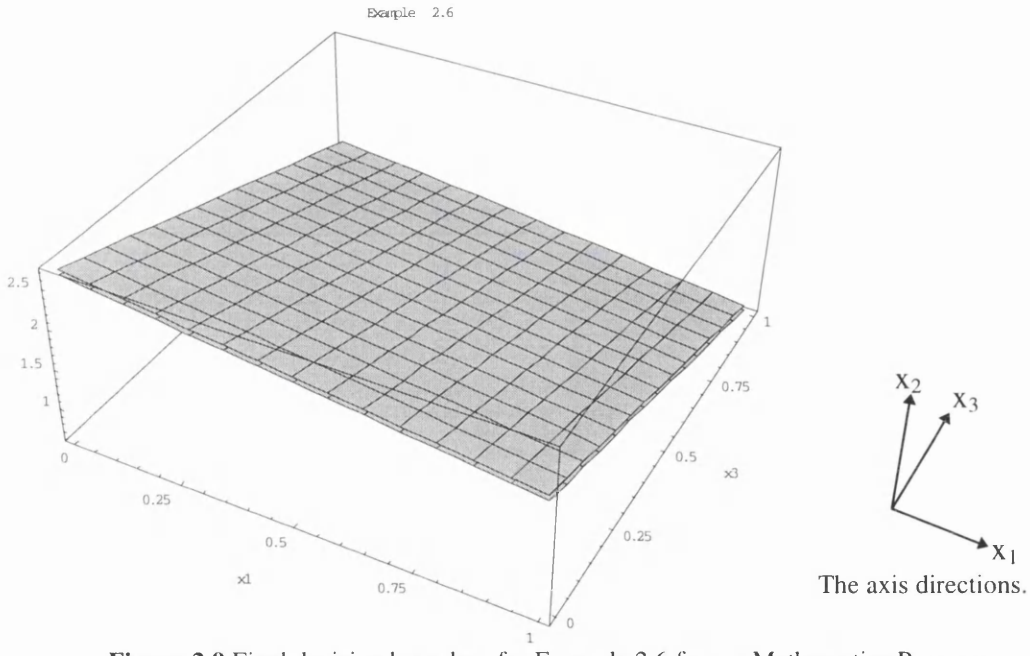


Figure 2.9 Final decision boundary for Example 2.6 from a Mathematica Program.

Character Classification

We will shortly (Example 2.8) use the perceptron to classify letters drawn from three fonts. We begin with a simpler task (Example 2.7).

Example 2.7 We use the simple perceptron to classify letters as A or not A. The output will give the Yes response '1' if that letter (the same letter in some font) is in

Class A; otherwise, the output will give the No response '-1' if this letter is in the 'not A' class, which we will denote by A'.

One letter uses a 9 x 7 grid of pixels (nine rows of seven) as illustrated in Figure 2.10. Each input character is represented by a vector of length 64 (x_0 = 1 followed by successive rows). For example, a character A is represented by the vector

$\mathbf{x} = (1,x_1,x_2,\ldots ,x_{63}) = (1,1,1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,-1,1,1,-1,1,1,1,1,-1,1,-1,-1,-1,-1,1,1,1,-1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,1,-1,-1,-1,1,-1,-1,-1,-1)$.

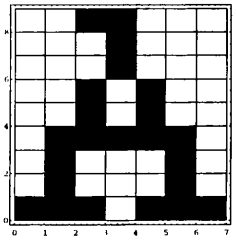


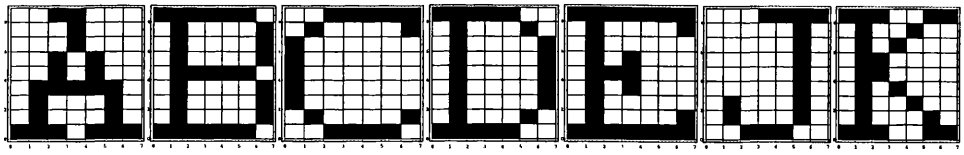
Figure 2.10 This figure represents the A character on a 9 x 7 grid.

Table 2.1 This table shows the training vectors with their target outputs (where target -1 means class A') When trained, the net will distinguish between A and A' characters for the limited testing we note below .

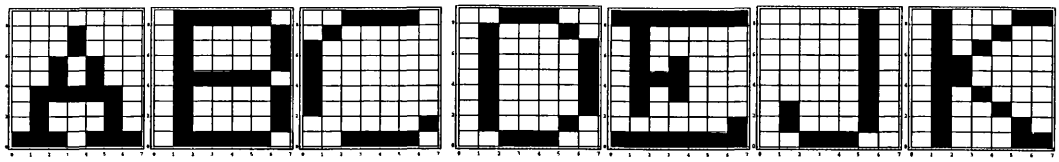
Letter	The training vector	target output
A	(1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,...,-1,-1,-1,1,-1,-1,-1)	1
B	(1,-1,-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,-1,...,-1,-1,-1,-1,-1,-1,1)	-1
C	(1,1,1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,1,-1,...,1,1,-1,-1,-1,-1,1)	-1
D	(1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,1,1,1,1,-1,1,...,-1,-1,-1,-1,-1,1,1)	-1
E	(1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,1,-1,...,-1,-1,-1,-1,-1,-1,-1)	-1
J	(1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,...,1,1,-1,-1,-1,1,1)	-1
K	(1,-1,-1,-1,1,1,1,-1,-1,1,-1,1,11,-1,1,1,...,-1,-1,-1,1,1,1,-1,-1)	-1

Result

Training There are seven letters- A, B, C, D, E, J, and K as shown in Table 2.1 and (I) below. From this table, the perceptron net is trained to recognize a letter A and reject the rest.

(I) 

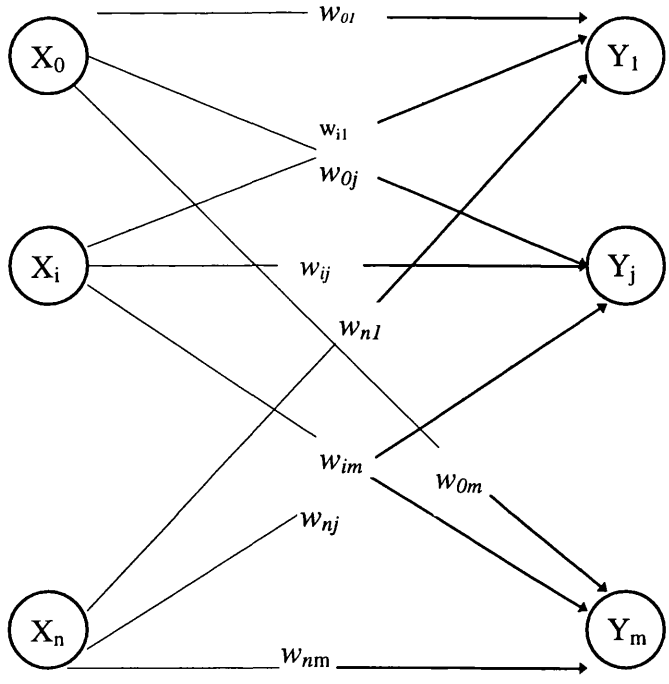
Testing We offered the net the seven variant letters shown in (II) below, of which exactly one is an 'A', and the net picked out this unique A from the rest.

(II) 

The A we test gives the only +1 response because it is closer to 'A' (in some sense) than to any other. It is important to note that the example above used only one output However, Example 2.8 below is a multi-output net, which tests for more than one letter.

2.4 The General Perceptron

Example 2.8 We train a general perceptron to recognize seven letter classes.
Each output represents one class. The net will allocate an input vector to this class or another. The architecture is shown in the Figure below where $n = 63$ and $m = 7$.



This figure is shown earlier when $i = 1, \ldots, 63$ and $j = 1, \ldots, 7$.

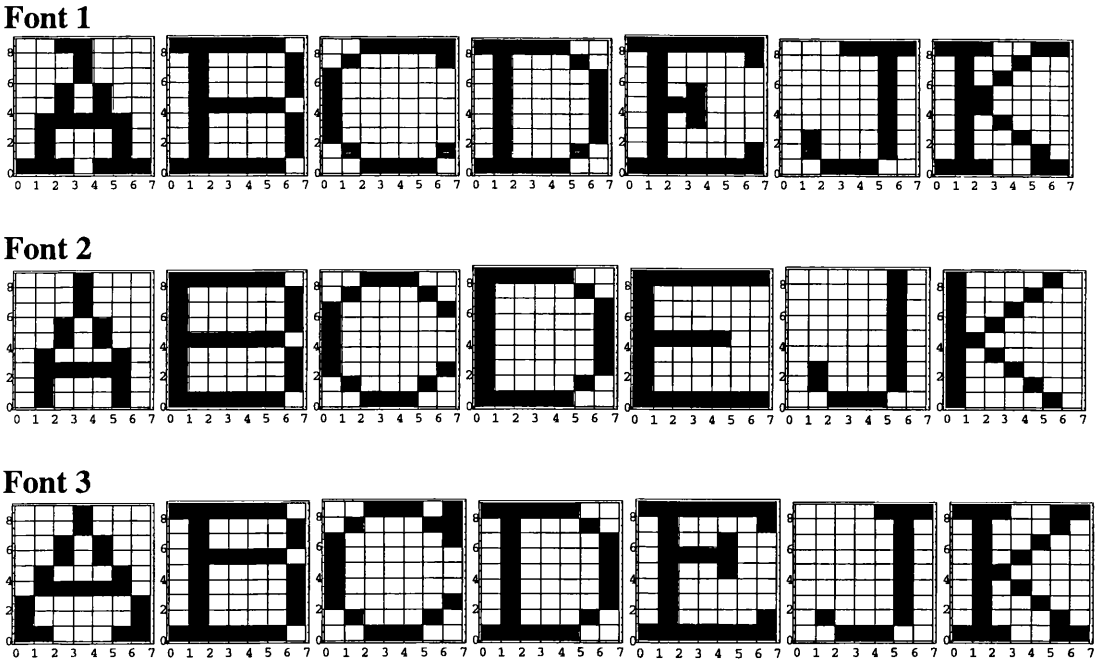


Figure 2.11 The 21 training vectors represent 7 letters, each in three different font versions.

The vector **x**, representing letter **A** in Font 1, is (1,-1,-1,1,1,-1,-1,-1, ... ,1,1,1). The target vector for the input A is represented by $\mathbf{t_a}$ = (1, -1, -1, -1, -1, -1, -1) with a '1' in just the first position. The target vector for the input B is represented by $\mathbf{t_b}$ = (-1, 1, -1, -1, -1, -1, -1) with a '1' in just second position, etc. From this example, the bipolar representation is better than the binary representation again.

After training this net to recognize letters from Figure 2.11, we use it to classify letters from Figure 2.12 below, which are from the same fonts but with 'noise' distortion.

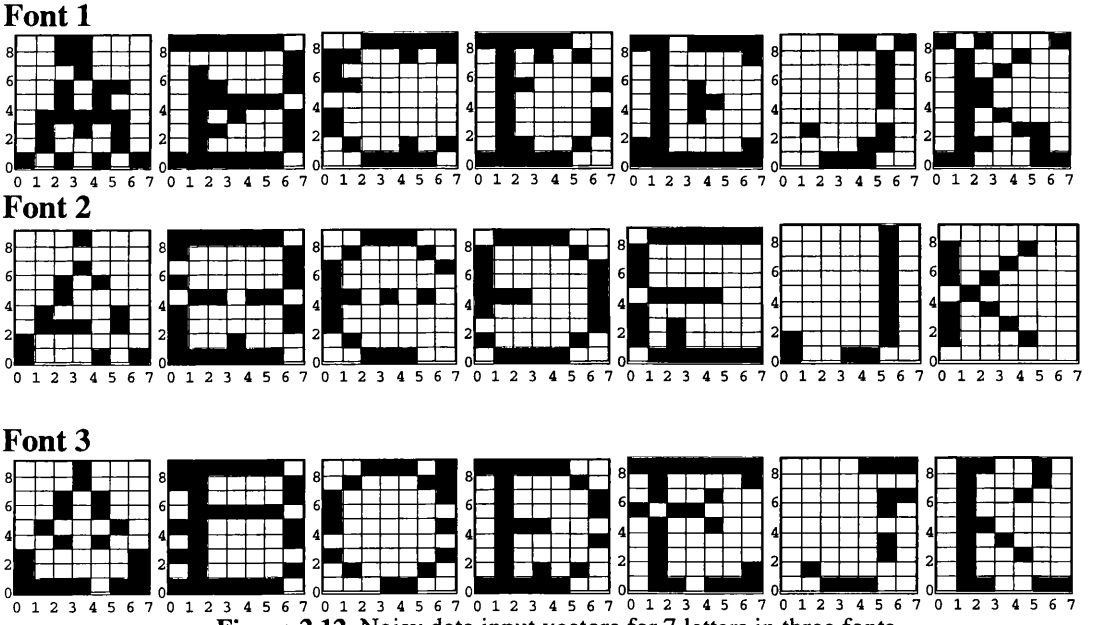


Figure 2.12 Noisy data input vectors for 7 letters in three fonts.

Result

The results of the output are shown in Table 2.2.

Table 2.2 This table shows the results of Example 2.8 classifying each noisy letter with three fonts belongs to class A, B, C, D, E, J, or K. The entries are to be "interpreted as follows.

Correct: the input letter X was correctly identified by an output vector with -1 in just the X position

X: the X position of the output letter was -1, but the input was not an X.

Unsure: some output vector positions were zero. For example 'unsure D' means a '0' in the fourth output vector position, (-1, -1, -1, 0, -1, -1, -1).

	Font 1	Font 2	Font 3
A	Correct	K	No classes
B	Correct	Correct	Correct
C	C,K	C	Correct
D	Correct	Correct	Unsure D
E	Correct	Correct	E,K
J	Correct	J,K	Correct
K	UnsureA,K	Correct	Correct

Conclusion The letters which give the correct results are A1 (Letter A Font 1) B1 D1 E1 J1 B2 C2 D2 E2 K2 B3 C3 J3 K3. Some letters are allocated to more than one class. For example C1 is both C and K classes. In fact we may use a competitive net (Chapter 3) to solve this problem. Overall, our net was correct in 13 out of 21 cases, a success rate of about 62%.

Chapter 3 Neural Networks Based on Competition

Example 2.8 in Chapter 2 gives the results in which for example an output of E responds to both E and K classes. This output unit should respond to either E or K, but not both. We can achieve this by including additional structure which forces the net to make a decision. Each input vector elicits a response from exactly one output unit. This method is called *competition*, the topic of the present chapter. One form of competition is the *winner-take-all* type, where we choose the winner as the output unit with the largest input signal or whose weight vector is closest to the input vector. If the weights are fixed, the net is called *Fixed-weight-competitive*. Maxnet is the example which we will introduce in Section 3.1.

Supervised learning Alternatively we can combine competition with a learning algorithm to adjust the weights. One possibility is supervised learning, in which the target output unit is known for each training vector. We exemplify this by *Learning Vector Quantization* (LVQ) in Section 3.2, which has a single layer of weights from the input to the output units. In addition, it uses a parameter called the *learning rate* α, which must be systematically reduced as learning proceeds.

Unsupervised learning The self-organising map (SOM) of Kohonen, whilst retaining the idea of competition, performs *unsupervised* learning by grouping input vectors into *clusters* (or classes), each cluster associated with an output unit. In common with LVQ, there is a single weight layer and use of a diminishing learning rate. The learning algorithm is based upon a structuring of output units into sets of mutual neighbours (a topology). An actual update for output/cluster unit j is given in terms of input vector $\mathbf{x}$ and weight vector $\mathbf{w}_j$ (the j^{th} column of the weight matrix $[w_{ij}]$) by

$$\mathbf{w}_j(\text{new}) = \alpha\,\mathbf{x} + (1-\alpha)\,\mathbf{w}_j(\text{old}). \qquad (3.1)$$

More details will be given when we study Kohonen nets in Section 3.3. We remark that after learning, in both cases above, an output class has the corresponding weight vector as an exemplar. Such vectors are sometimes viewed as forming a codebook. The net maps an input vector to the codebook entry.

3.1 Fixed-weight competitive nets

MAXNET

One example of a neural net based on competition is *Maxnet* whose weights are fixed. We recall that we have many units of a competing group and find only one unit, which has the largest activation (= 'on'), to be a winner. Here, the m units are not divided into layers but are fully interconnected, with symmetric weights (Figure 3.1). The net will stop calculating when the winner is "on", and other units in the competing group are "off". Maxnet is a simple net which acts as a subnet in the Hamming Net (Fausett, 1994).

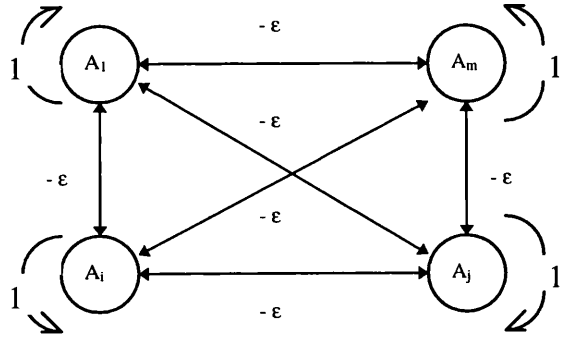


Figure 3.1 Max Net with $m = 4$.

The activation function is as follows

$$f(x) = \begin{cases} x & \text{if } x > 0; \\ 0 & \text{otherwise.} \end{cases} \qquad (3.2)$$

Notation $\mathbf{w}_j$ is as usual the j^{th} column of the weight matrix $[w_{ij}]$.

Algorithm

Set $0 < \varepsilon < 1/m$.

If $i = j$, then $w_{ij} = 1$, otherwise $w_{ij} = -\varepsilon$.

Initialise the activation vector **a**(old).

Repeat

For $j = 1, ..., m$ do $a_j(\text{new}) = f\,[\mathbf{w}_j \,.\, \mathbf{a}(\text{old})]$, (3.3)

a(old) = **a**(new) (3.4)

Until only one unit has nonzero activation.

Example 3.1 Using the Maxnet to find the winning unit.

Let the fixed weights be $\varepsilon = 0.1$, with four units whose initial activations $a_i(0)$ are

$a_1(0) = 0.25 \quad a_2(0) = 0.45 \quad a_3(0) = 0.65 \quad a_4(0) = 0.85$.

By using the algorithm above, we obtain the result shown in Table 3.1.

Table 3.1 Maxnet

Epoch	*Result* (**a**)
1	(0.055,0.275,0.495,0.715)
2	(0,0.1485,0.3905,0.6325)
3	(0,0.0462,0.3124,0.578)
4	(0,0,0.24992,0.54274)
5	(0,0,0.195646,0.517748)
6	(0,0,0.143871,0.498183)
7	(0,0,0.0940529,0.483796)
8	(0,0,0.0456732,0.474391)
9	(0,0,0,0.469824)

Therefore, the winner in this example is the fourth unit.

3.2 Learning Vector Quantization

Kohonen discovered the LVQ net [7]. The function of this net when trained is to allocate each input vector **x** to its correct class, namely the output unit whose weight vector is nearest to **x**.

Architecture

A Learning Vector Quantization (LVQ) neural net has no topological structure (we shall discuss such structure in Section 3.3). Its architecture is shown in Figure 3.2 below.

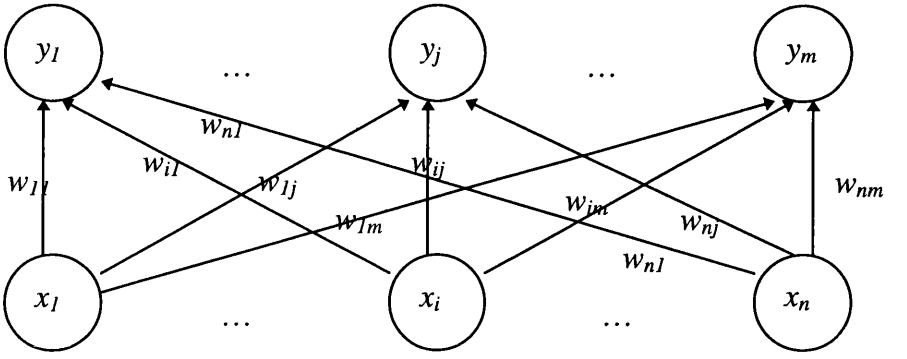


Figure 3.2 The architecture of a Learning Vector Quantization (LVQ) net.

Algorithm

Let T be the specified class of the training vector $\mathbf{x} = (x_1, ..., x_n)$, and C_j the class represented by the j^{th} output unit. Here $\|\mathbf{x}-\mathbf{w}_j\|$ is the Euclidean distance between the input vector and the weight vector for the j^{th} that unit. We use the first m training vectors for initializing the weight vectors, and the remaining vectors for training. We initialize the learning rate $\alpha(0)$, and then:

Repeat (t = the epoch number)

For each training vector $\mathbf{x}$ do

Find index J for which $\|\mathbf{x}-\mathbf{w}_J\|$ is a minimum.

If $T = C_J$, then $\mathbf{w}_J = \mathbf{w}_J + \alpha(\mathbf{x}-\mathbf{w}_J)$,

else $\mathbf{w}_J = \mathbf{w}_J - \alpha(\mathbf{x}-\mathbf{w}_J)$.

Update the learning rate.

Until $\alpha < 0.01$.

Example 3.2 Training the LVQ net with five vectors.

Two reference vectors are given, and the class for each input vector is already known.

	Input vector	*Class*
Reference vectors	(1, 0, 0, 0)	$C = 1$
	(0, 0, 1, 0)	$C = 2$
Training vectors	(1, 1, 0, 0)	$T = 1$
	(1, 0, 0, 1)	$T = 1$
	(0, 1, 1, 0)	$T = 2$

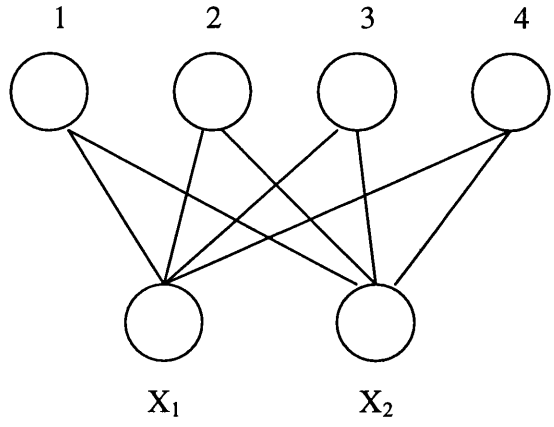
The first two vectors are the reference vectors in Class 1 ($C = 1$), and Class 2 ($C = 2$) respectively, and the remaining vectors are the training vectors. We update the learning rate α by using the decreasing function $\alpha(t) = \alpha(t\text{-}1) - t/100$, where t is the epoch number and $\alpha(0) = 0.1$ (this means $\alpha = \alpha(0)$ during the first epoch). The training stops when the learning rate α is less than 0.01.The result is shown in Table 3.2.

Table 3.2 The results of Example 2.6.

Epoch	j	**w**	
1	1	{1,0.1,0,0}	
	1	{1,0.09,0,0.1}	
	2	{0,0.1,1,0}	
$\alpha = 0.1$			
2	1	{1,0.1719,0,0.091}	
	1	{1,0.156429,0,0.17281}	
	2	{0,0.181,1,0}	
$\alpha = 0.09$			
3	1	{1,0.215479,0,0.160713}	
	1	{1,0.200395,0,0.219463}	
	2	{0,0.23833,1,0}	
$\alpha = 0.07$			
4	1	{1,0.23238,0,0.210685}	
	1	{1,0.223084,0,0.242257}	(I)
	2	{0,0.268797,1,0}	(II)
$\alpha = 0.04$			

Result = two trained class vectors (I), (II) as shown in the last row of Table 3.2.

Example 3.3 We use the LVQ net for a geometric example in the plane. The net is to allocate points (x_1, x_2) of the unit square $\{(x_1, x_2): 0 \leq x_1, x_2 \leq 1\}$ to one of four classes labelled 1, 2, 3, 4. Hence the architecture is that shown below.



The initial weights for these classes are the respective corners (0,0),(1,1),(1,0),(0,1). We use 81 training vectors $(0.a, 0.b)$, where a, b run through the digits 1 to 9. These points form a 9 x 9 grid on the unit square and we write at point $(0.a, 0.b)$ its *predefined* class as shown below

$x_2 = 0.9$ ⟶ 4 4 4 4 4 2 2 2 2
. 4 4 4 4 4 2 2 2 2
. 1 1 1 1 1 1 1 2 2
. 1 1 1 1 1 1 1 2 2
. 1 1 1 1 1 1 1 3 3
. 1 1 1 1 1 1 1 3 3
1 1 1 1 1 1 1 3 3
$x_2 = 0.2$ ⟶ 1 1 1 1 1 1 1 3 3
$x_2 = 0.1$ ⟶ 1 1 1 1 1 1 1 3 3.

↑

$x_1 = 0.1$

For example (0.1, 0.2) is in Class T = 1. The learning rate is updated by the equation

$$\alpha(t) = \alpha(t-1) - t/625,$$

where t is the epoch number and $\alpha(0) = 0.1$. The training stops when the learning rate α is less than 0.01.

Result

When t =1, the weight matrix **w** = {{0.42,0.49},{0.87,0.89},{0.96,0.17},{0.14,0.95}}, listed by columns, and the result is the following

$x_2 = 0.9$ ⟶ 4 4 4 4 4 2 2 2 2
. 4 4 1 1 1 1 2 2 2
. 4 1 1 1 1 1 1 2 2
. 1 1 1 1 1 1 1 1 2
. 1 1 1 1 1 1 1 1 3
. 1 1 1 1 1 1 1 3 3
1 1 1 1 1 1 1 3 3
$x_2 = 0.2$ ⟶ 1 1 1 1 1 1 3 3 3
$x_2 = 0.1$ ⟶ 1 1 1 1 1 3 3 3 3.

When t = 10 (α = 0.02), the weight matrix is

w ={{0.35,0.40},{0.82,0.84},{0.97,0.31},{0.33,0.93}},

and the resulting classification is

$x_2 = 0.9$ ⟶ 4 4 4 4 4 2 2 2 2
. 4 4 4 4 4 2 2 2 2
. 4 4 4 4 4 2 2 2 2
. 1 1 1 1 1 2 2 2 2
. 1 1 1 1 1 1 3 3 3
. 1 1 1 1 1 1 3 3 3
1 1 1 1 1 1 3 3 3
$x_2 = 0.2$ ⟶ 1 1 1 1 1 1 3 3 3
$x_2 = 0.1$ ⟶ 1 1 1 1 1 1 3 3 3.

Conclusion After a hint by the training vectors, the net is moving towards allocating each quarter of the square to its rightful class.

3.3 Kohonen Self-Organizing Maps

The self-organizing neural nets are also called *topology preserving maps* because they use a topological structure which we will explain later. It defines when units are 'close' to each other. Kohonen nets are self-organizing networks used as *pattern classifiers*. Training involves grouping similar patterns in close proximity in this pattern space so that a cluster of similar patterns cause the same unit to respond.

The winner is the cluster which has the minimum square Euclidean distance between the input pattern and the weight vector as mentioned earlier. After the net has found the winner, both the winner and its *neighboring* units will update their weights (a special feature of the Kohonen net).

The output is the weighted sum of inputs, possibly after the operation of an activation function. When the input is applied, the output will be a single *1* in the region of the pattern space, which corresponds to a particular class of patterns. The exemplar vectors are stored in such a way that similar exemplar vectors are found in units which are close to each other. Exemplar vectors which are very different are in units situated far apart.

A preview we shall see how we can use a Kohonen net to solve the Traveling Salesman Problem (Section 3.5).

Architecture

The SOM architecture is the same as the LVQ architecture (Figure 3.2), with n - vector input. However, the m clustering output units are formed into an array/grid in one or two dimensions (e.g. a linear, rectangular, or hexagonal array), which is used to specify the neighbourhoods of various radii for any unit. This neighbourhood specification or *topology*, is a key feature of the Kohonen SOM. In the case of a linear array of units 1, 2, ..., m the neighbour units j within a radius R of unit J are given by

$$\max(1, J - R) \leq j \leq \min(J + R, m).$$

Our second example is rectangular (Figure 3.4). There are 9 units within radius 1 of the winning unit #, and 25 units within radius 2. Here, and in the subsequent two Figures, we represent the boundaries of successively larger neighborhoods, of radius R = 0, 1, 2 by dotted, continuous, then dashed lines.

The next is a hexagonal array with hexagonal neighbourhoods (see Figure 3.5). There are 7 units within radius 1 of # and 19 units within radius 2. Finally in Figure 3.6, we

show a rectangular array with diamond neighbourhoods. There are 5 units within radius 1 and 13 units within radius 2.

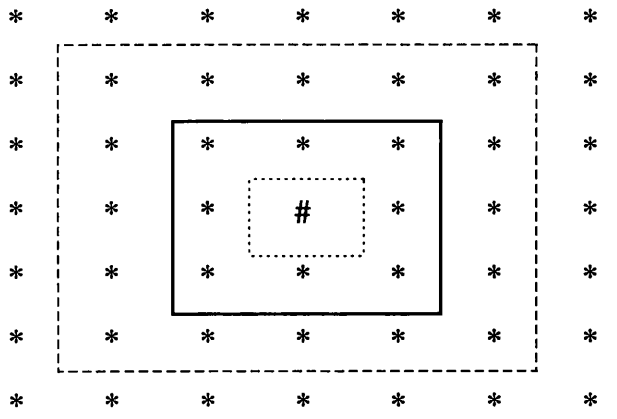


Figure 3.4 Neighbourhoods of the unit # for rectangular grid.

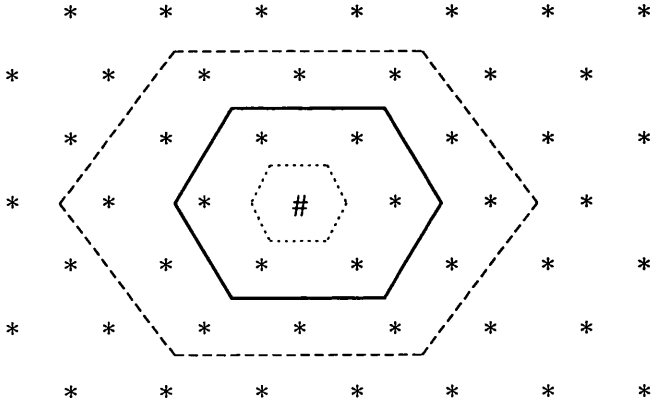


Figure 3.5 Neighbourhoods of the unit # for hexagonal grid.

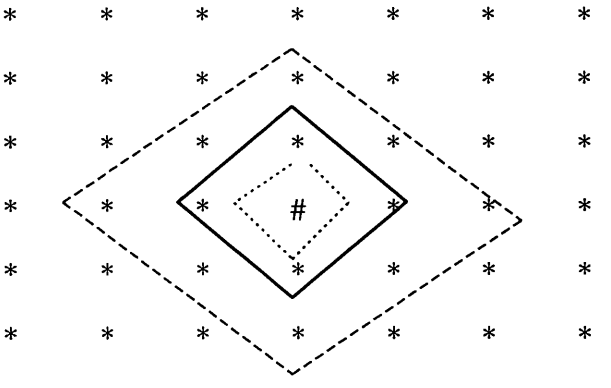


Figure 3.6 Diamond neighbourhoods on a rectangular grid..

We randomise values of initial weights in the same range as input, for example, if input vectors are in the *bipolar* form, then we take random values between -1 and 1.

Algorithm

Set initial weight vectors $\mathbf{w}_j$, number of cluster units m, learning rate α, and initial radius R.
Set $0 < \lambda < 1$.
Repeat (t = the epoch number)
For each training vector **x** do
If $R \geq 1$, then R is reduced at specified epoch t.
Find the index J for which $\|\mathbf{w}_j - \mathbf{x}\|$ is minimum.
For all j in the nbd of J with radius R do
(*Update the weight of winning unit and its neighbourhood. See Example below*)

$$\mathbf{w}_j = \mathbf{w}_j + \alpha(\mathbf{x} - \mathbf{w}_j). \qquad (3.5)$$

$$\alpha(t) = \alpha(0)\,\lambda^t, \quad \text{(for the geometric function)} \qquad (3.6)$$

or, $$\alpha(t) = \lambda + \alpha t, \quad \text{(for the linear function)} \qquad (3.7)$$

Until $\alpha < 0.05$.

Example Suppose the topological structure is of diamond type (Figure 3.6) with $R = 2$, the units arranged in rows of length ρ and numbered row by row. At the first specified epoch R is reduced to 1 and updating is confined to units $j = J\text{-}\rho, J\text{-}1, J, J\text{+}1, J\text{+}\rho$. After the next radius reduction to $R = 0$, we update only the winning unit J.

Example 3.4 Using the Kohonen SOM to cluster four vectors. The input vectors are ((1, 0, 0, 0), (0, 1, 0, 1), (1, 0, 1, 0), (0, 0, 1, 1)). The initial weight vectors are ((0.1,0.8,0.5,0.9),(0.2,0.5,0.1,0.9)). The number of clustering units is $m = 2$ and the radius of topological neighbourhood is $R = 0$. Set the learning rate $\alpha(0) = 0.6$ and $\alpha(t) = 0.6 \times 0.96^t$. This training stops when the learning rate is less than 0.01. The result is shown in Table 3.3.

Table 3.3 Four vector are clustered by using SOM.

Epoch	norm	closest to unit
1	(2.06,0.78) (0.46,2.40) (3.11,0.56) (1.35,1.93)	2 1 2 1
...	.....	....
101	(2.5,0.26) (0.52,3.3) (2.5,0.25) (0.49,2.2)	2 1 2 1

The weight vectors converge to ((0, 0.49, 0.51, 1), (1, 0, 0.51, 0)). This means that the first and third input vectors are in Cluster 2, the second input vector and fourth input vector are in Cluster 1.

3.4 Using the Kohonen SOM for Character Recognition

We can use the *Kohonen SOM* to cluster seven characters, A, B, C, D, E, J, and K with three fonts for which we refer back to Figure 2.11. There are 15 cases in this Section. Cases 1-5 have No Topological Structure $R = 0$. Then Cases 6-9 use the Linear Array (Figure 3.3). Next, Cases 10-13 use the Diamond neighbourhoods (Figure 3.6). Finally, Cases 14-15 use the Rectangular neighbourhoods (Figure 3.4).

The cases have these conditions in common:

1. The maximum number of cluster units $m = 25$.
2. The learning rate α decreases linearly from 0.6 to 0.01. The training then stops.
3. The training input vectors are the input vectors from Example 2.8 but without a component $x_0 = 1$.

No Topological Structure ($R = 0$)

Case 1 Binary input vectors. The initial weights are randomised between 0.1 and 0.9. The learning rate is geometric decreasing function $\alpha(t) = 0.6*0.96^t$, where t is the epoch number. The result is shown in Table 3.4.

Table 3.4 Kohonen SOM with no topological structure and *binary* input vectors. The learning rate is reduced by using the geometric decreasing function.

No. of cluster unit	*Patterns*
5	A1, A2
1	A3
21	B1, B3, D1, D3, E1, E3, K1, K3
23	B2, D2, E2
4	C1, C2, C3, J1, J2, J3
8	K2

Discussion Notice only 6 clusters were formed. The outputs of A1 and A2 give the response correctly, but A3 is in the other clustering unit. Notice that C1, C2, C3 and J1, J2, J3 are in the same clustering unit. B1, B3, D1, D3, E1, E3, K1, K3 are in the

same clustering net. If we use the bipolar input vectors and change the parameters or equations, then the result may change as we show below.

Case 2 Bipolar input vectors. The initial weights are randomized between 0 and 1. The learning rate is the geometric decreasing function $\alpha(t) = 0.6*0.96^t$. The result is shown in Table 3.5.

Table 3.5 Kohonen SOM with no topological structure and *bipolar* input vectors. The learning rate is reduced by using the geometric decreasing function.

No. of cluster unit	*Patterns*
19	A1, A2
1	A3
7	B1, B3, D1, D3, E1, E3, K1, K3
9	C1, C2, C3, B2, D2, E2
23	J1, J2, J3
2	K2

Discussion This result is better than Case 1 because the outputs of J1, J2, J3 respond correctly. However, we still have a problem because B2 D2 E2 are put with C1 C2 C3.

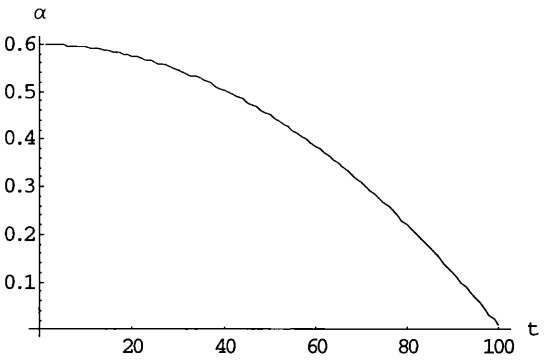
Case 3 Binary input vectors. The initial weights are randomized between 0 and 1. The learning rate is the non linear decreasing function $\alpha(t) = \alpha(t\text{-}1) - 0.59/5050\, t$ ($t \geq 1$, $\alpha(0) = 0.6$). The result is shown in Table 3.6.

Table 3.6 Kohonen SOM with no topological structure and *binary* input vectors. The learning rate is reduced by using the non linear decreasing function.

No. of cluster unit	*Patterns*
3	A1, A2
6	A3
1	B1, B3, D1, D3, E1, E3, K1, K3
10	B2, D2, E2
25	C1, C2, C3
7	J1, J2, J3
20	K2

Discussion The result of Case 3 is better than that of Cases 1 and 2 because A1, A2, C1, C2, C3, J1, J2, J3 do not mix each other.

Derivation of the learning function



The expression for $\alpha(t)$ is obtained from the chosen form $\alpha(t) = \alpha(t\text{-}1) - t\text{A}$ ($t = 1, ..., 100$, A is constant) by specifying $\alpha(0) = 0.6$ and $\alpha(100) = 0.01$. We have

$$\alpha(1) = \alpha(0) - 1\text{A}.$$

$$\alpha(2) = (\alpha(0) - 1\ \text{A}) - 2\text{A}$$

.....

$$\alpha(t) = \alpha(0) - (1+2+ ... + t)\ \text{A} = 0.6 - (t / 2)(t+1)\ \text{A}$$

Putting $t = 100$ $\quad 0.01 = \alpha(100) = 0.6 - 50\ (101)\ \text{A}$

Hence $\quad \text{A} = 0.59 / 5050.$

Case 4 Binary input vectors. The initial weights are randomized between 0 and 1. The learning rate is the linear decreasing function $\alpha = 0.6 - (0.0059 * t)$. The result is shown in Table 3.7.

Table 3.7 Kohonen SOM with no topological structure and *binary* input vectors. The learning rate is reduced by using the linear decreasing function.

No. of cluster unit	*Patterns*
22	A1, A2
25	A3
12	B1, B3, D1, D3, E1, E3, K1, K3
4	C1, C2, C3, B2, D2, E2
10	J1, J2, J3
15	K2

Discussion This result is again similar to Case 2.

Case 5 Binary input vectors. The initial weights are randomised between 0 and 1. The learning rate is

$$\alpha(t) = 0.6\, e^{-t/\delta},$$

where $\delta = 1/21 \sum_i \|x_i - \mu\|^2$ and $\mu = 1/21(\sum_i x_i)$ ($i = 1, ..., 21$ because of 21 characters). The result is shown in Table 3.8.

Table 3.8 Kohonen SOM with no topological structure and *binary* input vectors. The learning rate is reduced by using the exponential decreasing function.

No. of cluster unit	*Patterns*
1	A1, A2
18	A3
2	B1, B3, D1, D3, E1, E3, K1, K3
9	B2, D2, E2
8	C1, C2, C3
17	J1, J2, J3
20	K2

Discussion This program has 185 epochs and the best result is similar to Case 3 as shown above. This program can separate A1, A2, A3, C1, C2, C3, J1, J2, J3 classes.

Conclusion (no topological structure) Cases 3, 5 give the best result when we compare to Cases 1, 2, 4. However, B1, B3, D1, D3, E1, E3, K1, K3, B2, D2, E2 are mixed. Therefore, we will use other structures to match these letters. The next structure is the Linear Structure as shown below.

The Linear Array ($R \geq 1$)

Case 6 Bipolar input vectors. The initial weight elements are randomized between 0 and 1. The learning rate is the geometric decreasing function $\alpha = 0.6^* 0.96^t$. We *fix* the radius $R = 1$. The result is shown in Table 3.9.

Table 3.9 Kohonen SOM with the linear array and *bipolar* input vectors. The learning rate is reduced by using the geometric decreasing function.

No. of cluster unit	*Patterns*
1	A2
2	A1
4	A3
11	B2, E2
21	B1, B3
6	C1
7	C2
8	C3
9	D2
19	D1, D3
23	E1, E3
15	J2
17	J1, J3
25	K1, K3
13	K2

Discussion Each pattern separates each patterns very clearly, but the problem is how we can cluster the same character (but different fonts) to the same cluster. This program uses 101 epochs. Notice that B2, E2 are in the same cluster. We probably should change some parameters to obtain a better result.

Case 7 Binary input vectors. The initial weights are randomized between 0.1 and 0.9. The learning rate is the linear decreasing function $\alpha = 0.6 - 0.0059 * t$. We reduce the radius from $R = 1$ to 0 when the epoch $t > 60$. The result is shown in Table 3.10.

Table 3.10 Kohonen SOM with the linear array and *binary* input vectors. The learning rate is reduced by using the linear decreasing function.

No. of cluster unit	*Patterns*
17	A1
19	A2
15	A3
2	B1, B3
13	B2, E2
7	C1, C2, C3
3	D1, D3
11	D2
1	E1, E3
9	J1, J2, J3
5	K1, K3
14	K2

Discussion This program is much better than Case 6 because B1, B3, E1, E3, D1, D3, K1, K3 are separated to different clusters. In addition, C1, C2, C3 and J1, J2, J3 are matched correctly. Kohonen (1989a, p. 133) noticed that the linearly decreasing function gives a good result for computation, and a geometric decrease produces similar results. However, B2, E2 are still in the same cluster.

Case 8 Bipolar input vectors. The initial weight elements are randomized between 0 and 1. The learning rate is updated by

$$\alpha(t) = 0.6\, e^{-t/\delta},$$

where δ (new) = δ (old) + 1/21 $\sum_i \|x_i - \mu\|^2$, $\delta(0) = 0$, $\mu = 1/21(\sum_i x_i)$, $i = 1, ..., 21$. We fix the radius $R = 1$. We remark that in Epoch 286, $\delta = 12913.5$ so that α (286) cannot decrease and α never drops below 0.52, i.e. this training is non-stop. The result of Epoch 286 is shown in Table 3.11.

Table 3.11 Kohonen SOM with the linear array and *bipolar* input vectors. The learning rate is reduced by using the exponential decreasing function.

No. of cluster unit	*Patterns*
17	A1, A2
15	A3
3	B1, B3
19	B2
9	C1
11	C2, C3
1	D1, D3
21	D2
5	E1, E3
19	E2
23	J1, J3
25	J2
7	K1, K3
13	K2

Discussion Case 7 gives a result better than this program although the number of clustering units of A1, A2, A3, J1, J2, J3, C1, C2, C3 close each other. However, this training separates B2 from E2.

Case 9 Bipolar input vectors. The initial weights are randomized between 0.1 and 0.9. The learning rate α is decreased by the equation

$$\alpha(t) = 0.6\, e^{-t/\delta},$$

where $\delta_i = 1/21 * \|x_i - \mu\|^2$ ($\mu = 1/21(\sum_i x_i)$, $i = 1, ..., 21$). When $t > 70$, we reduce the radius $R = 1$ to 0. The result is shown in Table 3.12.

Table 3.12 Kohonen SOM with the linear array and *bipolar* input vectors. The learning rate is reduced by using the exponential decreasing function. The radius R is reduced during training.

No. of cluster unit	*Patterns*
14	A1, A2
12	A3
20	B1, B3, D1, D3
25	B2, E2
22	C1
23	C2, C3
19	E1, E3
24	D2
18	K1, K3
8	K2
16	J1, J2, J3

Discussion There are 185 epochs. This program has a problem because B2 and E2 are in the same cluster unit, and also B1, B3, D1, D3.

Conclusion Case 7 gives the best result in the sense that C1, C2, C3 are in the same cluster as well as J1, J2, J3 although B2, E2 are still in the same cluster. Case 8 does not mix different letters in the same group, but it still needs to match these letters in the same cluster.

The Diamond neighbourhoods

Case 10 Bipolar input vectors. The initial weights are randomised between 0 and 1. The geometric decreasing function $\alpha(t) = 0.6 \times 0.96^t$. We *fix* the radius $R = 1$. The result is shown in Table 3.13.

Table 3.13 Kohonen SOM with the diamond neighbourhoods and *bipolar* input vectors. The learning rate is reduced by using the exponential decreasing function. The radius R is fixed during training.

No. Of cluster unit	*Patterns*
19	A1
21	A2
14	A3
6	B1
1	B2, D2, E2
7	B3, E1, E3
3	C1, C2, C3
5	D1, D3
11	J1, J2, J3
9	K1, K3
16	K2

Discussion Number of clustering units 1 and 7 have different patterns. The problem is similar to Case 9. However characters C1, C2, and C3 are in the same clustering net. Patterns J1, J2, J3 are also in the same clustering net. Patterns A1, A2, A3 are not in the same clustering net, but at least they are not with other characters.

Case 11 Bipolar input vectors. The initial weights are randomized between 0.1 and 0.9. The learning rate is the exponential decreasing function $\alpha(t) = 0.6 * e^{(-t/\delta)}$, where $\delta = 1/21 \sum_i \|\mathbf{x}_i - \mu\|^2$, $\mu = 1/21(\sum_i \mathbf{x}_i)$, and $i = 1, \ldots, 21$. We start with $R = 2$, reducing R to 1 when $t > 70$ and to 0 when $t \geq 100$. The result is shown in Table 3.14.

Table 3.14 Kohonen SOM with the diamond neighbourhoods and *bipolar* input vectors. The learning rate is reduced by using the exponential decreasing function. The radius R is reduced during training.

No. Of cluster unit	*Patterns*
1	A1, A2
8	A3,
24	B1, B3
16	B2, E2
20	C1
19	C2, C3
25	D1, D3
18	D2
23	E1, E3
3	J1
6	J2
4	J3
22	K1, K3
14	K2

Discussion When we run this program again, the result (as shown below) is changed because we randomize the value of weights. The result is shown in Table 3.15.

Table 3.15 Kohonen SOM with the diamond neighbourhoods and *bipolar* input vectors. The learning rate is reduced by using the exponential decreasing function. The radius R is reduced during training.

No. Of cluster unit	*Patterns*
6	A1
4	A2
25	A3
19	B1, B3
8	B2, E2
13	C1
11	C2
12	C3
17	D1, D3
10	D2
21	E1, E3
15	J1, J2, J3
23	K1, K3
2	K2

The second result is better than the first result because J1, J2, J3 are in the same cluster. However, A1, A2, A3 are not in the same cluster.

Case 12 Bipolar input vectors. The initial weights are randomised between 0 and 1. The learning rate is the geometric decreasing function $\alpha(t) = 0.6^* 0.96^t$. We start with $R = 2$, reducing R to 1 when $t > 60$ and to 0 when $t \geq 80$. The result is shown in Table 3.16.

Table 3.16 Kohonen SOM with the diamond neighbourhoods and *bipolar* input vectors. The learning rate is reduced by using the geometric decreasing function. The radius R is reduced during training.

No. Of cluster unit	*Patterns*
21	A1, A2
19	A3
16	B1, B3
6	B2, E2
8	C1, C2, C3
15	D1, D3
4	D2
17	E1, E3
10	J1, J3
11	J2
13	K1, K3
23	K2

Discussion This training can match C1, C2, C3 to be in the same cluster. Most of the patterns, i.e. A1, A2 are in the same cluster of letter A (also B1, B3, D1, D3, E1, E3, J1, J3, K1, K3). However, A3, D2, J2, K2 are separated to the clusters of their letters. Notice that B2 and E2 are in the same cluster. Therefore, this training should be improved.

Case 13 Binary input vectors. The initial weights are randomised between 0.1 and 0.9. The learning rate is the exponential decreasing funciton $\alpha(t) = 0.6 * e^{t(-441/19912)}$. We start with $R = 2$, reducing R to 1 when $t > 70$ and to 1 when $t \geq 100$. The result is shown in Table 3.17.

Table 3.17 Kohonen SOM with the diamond neighbourhoods and *binary* input vectors. The learning rate is reduced by using the exponential decreasing function. The radius R is reduced during training.

No. Of cluster unit	*Patterns*
14	A1, A2
16	A3
10	B1, B3
7	B2
1	C1, C2, C3
9	D1, D3
8	D2
11	E1, E3
6	E2
13	J1, J2, J3
12	K1, K3
5	K2

Discussion We use this initial learning rate because Case 11 will stop when $t = 49$, and we would like to know whether the result gives a better result or not. This program gives the best result because C1, C2, C3 and J1, J2, J3 are classified correctly. Moreover, A1 A2 are in the same cluster (the same as B1 B3, D1 D3, E1 E3, and K1 K3).

Conclusion Case 13 gives the best result in the sense that different letters do not mix in the same cluster. Moreover, C1, C2, C3 are in the same cluster as well as J1, J2, J3.

The Rectangular Neighbourhoods

Case 14 Bipolar input vectors. The initial weights are randomized between 0 and 1. We update the learning rate α by using the linear decreasing function $\alpha = 0.6\text{-}0.0059t$. We start with $R = 2$, reducing to 1 when $t > 60$ and to 0 when $t \geq 80$. The result is shown in Table 3.18.

Table 3.18 Kohonen SOM with rectangular neighbourhoods and *bipolar* input vectors. The learning rate is reduced by using the linear decreasing function. The radius R is reduced during training.

No. of cluster unit	*Patterns*
1	A1, A2
22	A3
5	B1, B3
8	B2
11	C1
10	C2, C3
6	D1, D3
9	D2
4	E1, E3
8	E2
13	J1, J3
15	J2
3	K1, K3
20	K2

Conclusion Each cluster unit has no different letters. This training cannot group C1, C2, C3 to the same cluster unit so does J1, J2, J3.

Case 15 Bipolar input vectors. The initial weights are randomised between 0 and 1. The learning rate is the geometric decreasing function $\alpha(t) = 0.6 \times 0.96^t$. We start with $R = 2$, reducing to 1 when $t > 60$ and to 1 when $t \geq 80$. The result is shown in Table 3.19.

Table 3.19 Kohonen SOM with rectangular neighbourhoods and *bipolar* input vectors. The learning rate is reduced by using the geometric decreasing function. The radius R is reduced during training.

No. of cluster unit	*Patterns*
1	A1, A2
23	A3
8	B1, B3
3	B2
14	C1
16	C2, C3
6	D1, D3
4	D2
10	E1, E3
3	E2
18	J1, J2, J3
12	K1, K3
20	K2

Conclusion Case 15 matches J1, J2, J3 to the same cluster so that Case 15 is better than Case 14 because Case 14 matches only J1, J2 to the same cluster. The other letters of Case 15 are classified to the similar way as Case 14. These letters do not mix with different letters.

Compare each topological structure of SOM

Table 3.20 The results of each topological structure which used SOM to classify letters. *Mixed clusters* are those which contain examples of more than one letter.

Topological Structure	**No. of mixed clusters**	**C1, C2, C3 are a cluster.**	**J1, J2, J3 are a cluster.**	**Total cluster units**
No topological structure (Case 3 or 5)	**11**	**Yes**	**Yes**	**7**
Linear Array (Case 7)	**2**	**Yes**	**Yes**	**12**
Diamond nbds. (Case 13)	**-**	**Yes**	**Yes**	**12**
Rectangular nbds. (Case 15)	**-**	**No**	**Yes**	**13**

The result from Table 3.20 above are discussed into the following order:

Step 1 The best results of each method (i.e. Cases 5, 7, 13, and 15) has been chosen from all above cases.

Step 2 The classification of "Step 1" results depends in the following *criteria*:

(I - most important, II - important, and III - less important)

(I) The different letters (A, B, C, etc.) should not be put to the same cluster.

(II) Three fonts of each letter (e.g. A1, A2, A3) should be put to the same cluster as many cluster units as possible.

(III) The number of cluster units should be as small as possible.

Step 3 The obtained results from "Step 1" are compared by using *criteria* from "Step 2" in order to get the best topological structure.

Discussion

By using criteria from "Step 2", the results indicate that No Topological Structure is not a good structure because there are many letter combinations. The Linear array is better than No Topological Structure because there are less letter combinations. The Rectangular grid has no letter combinations the same as the Diamond neighbourhoods but (C1 C2 C3) are not grouped to the same cluster unit. In addition, the Diamond neighbourhoods (C1 C2 C3) and (J1 J2 J3) are grouped to their own cluster. Moreover, its cluster unit numbers are less than the Rectangular grid.

Conclusion

The Diamond structure is the *best* topological structure of SOM method for these experiments. This structure is better than the Rectangular grid, Linear array, and No Topological Structure, respectively.

3.5 The Kohonen SOM and the Traveling Salesman Problem

We can use the Kohonen SOM net to solve this problem. The idea of the problem is that the salesman will visit every city only one time and come back to the first city. He would like to know the order which gives him the shortest way. The distances are shown below. See the graph in Figure 3.7.

Table 3.21 The distance between cities. This aids comparison of the total distance covered, for different orders of visiting the cities.

	AB	C	D	E	F	G	H	I	J	
A	.0000	.3361	.3141	.3601	.5111	.5176	.2982	.4564	.3289	.2842
B	.3361	.0000	.1107	.6149	.8407	.8083	.5815	.6418	.4378	.3934
C	.3141	.1107	.0000	.5349	.7919	.8207	.5941	.6908	.4982	.4501
D	.3601	.6149	.5349	.0000	.3397	.6528	.5171	.7375	.6710	.6323
E	.5111	.8407	.7919	.3397	.0000	.4579	.4529	.6686	.7042	.6857
F	.5176	.8083	.8207	.6528	.4579	.0000	.2274	.2937	.4494	.4654
G	.2982	.5815	.5941	.5171	.4529	.2274	.0000	.2277	.2690	.2674
H	.4564	.6418	.6908	.7375	.6686	.2937	.2277	.0000	.2100	.2492
I	.3289	.4378	.4982	.6710	.7042	.4494	.2690	.2100	.0000	.0498
J	.2842	.3934	.4501	.6323	.6857	.4654	.2674	.2492	.0498	.0000

Example 3.4 We use a Kohonen net to find the most economical tour of the cities positioned as in Figure 3.7. The input vectors are the coordinates of these cities. The output units, identified by their randomised initial exemplar weight vectors, are put in some linear order, with additionally the last succeeded by the first, i.e. their order is cyclic. There is one output for each city and the objective is to assign each city to an output unit. The order of visiting cities is then the order of their associated units. We argue that the neighbourhood structure (with $R = 1$) causes the long term Kohonen solution to minimise the length of the journey around the cities/output units. The initial weight matrix

$\mathbf{w}$ = ((0.72,1),(0.92,0.66),(0.4,0.39),(0.78,0.55),(0.47,0.74),(0.74,0.49),(0.82,0.59),(0.7,0.34), (0.37, 0.27),(0.74, 0.13)).

We select the initial learning rate 0.5. If $t \leq 100$, then $R = 1$ and the learning rate α is reduced by using the linearly decreasing function

$$\alpha = 0.5 - 0.001\, t.$$

If $t > 100$, then $R = 0$ and the linearly decreasing function is

$$\alpha = 0.4 - 0.002\,(t - 100).$$

This training stops when the learning rate α is less than 0.02, with unit positions and order as shown in Figure 3.8. The result is shown in Table 3.22 below and combined with city positions in Figure 3.9.

Table 3.22 The TSP results by using SOM (Linear Array) method.

No. of cluster unit	*Pattern*
4	A
10	B, C
3	D
1	E
6	F
6	G
7	H
8	I, J

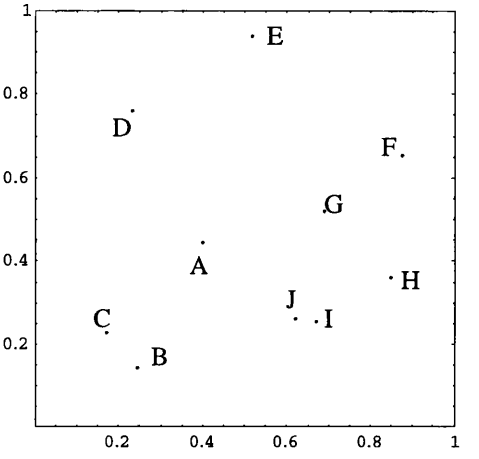


Figure 3.7 Positions of the cities.

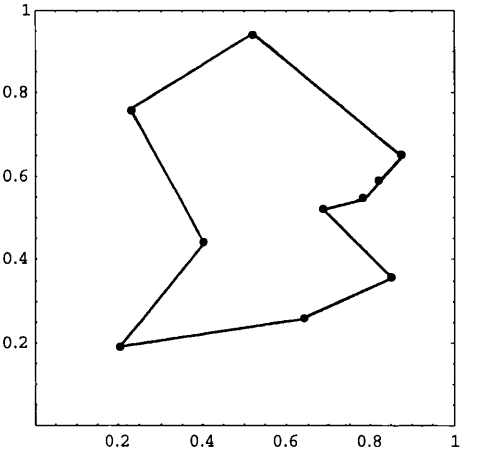


Figure 3.8 The output exemplar vectors after 31 epochs.

This man needs to select B or C city by himself because these cities are in the same clustering net and also I or J city. The graph is shown below.

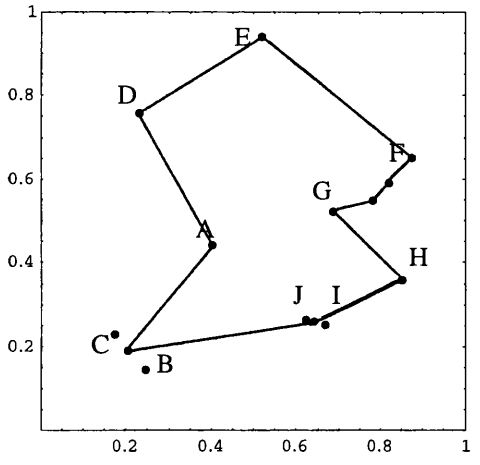


Figure 3.9 This figure combines the two figures above.

Therefore, there are four possibilities which this man can select.

(1) A D E F G H I J B C.

(2) A D E F G H I J C B.

(3) A D E F G H J I B C.

(4) A D E F G H I J C B.

Chapter 4 The Backpropagation Neural Net

4.1 Introduction

The supervised learning algorithm based on minimizing the mean squared error for multilayer feedforward neural nets is called *backpropagation*, or the *generalized delta rule.* This rule uses a *gradient descent method* (more details in Appendix A), and was suggested by Rumelhart, McClelland, and Williams in 1986. A neural net using this rule is called a *backpropagation neural net.* This net calculates the value of the error function, and backpropagates error information from one layer to the previous one. The weights for each unit are then updated. The training of the net will conclude when the net produces responses to the training input which are sufficiently close to the targets.

The training method includes three steps: the *feedforward* of the input training pattern, the *backpropagation* of the associated error, and the *adjustment* of weights. Notice that the outputs of the first layer are the inputs to the second layer.

4.2 Architecture

We can use more than one hidden layer, but we discuss only the case of one hidden layer in this Chapter. There are the input units (X units), one layer of hidden units (Z units), and the output units (Y units). The bias is in the hidden units (v_{0j}) and the output units (w_{0k}). The first step is the forward direction (the feedforward step), but we reverse the direction during the backpropagation step (Figure 4.1).

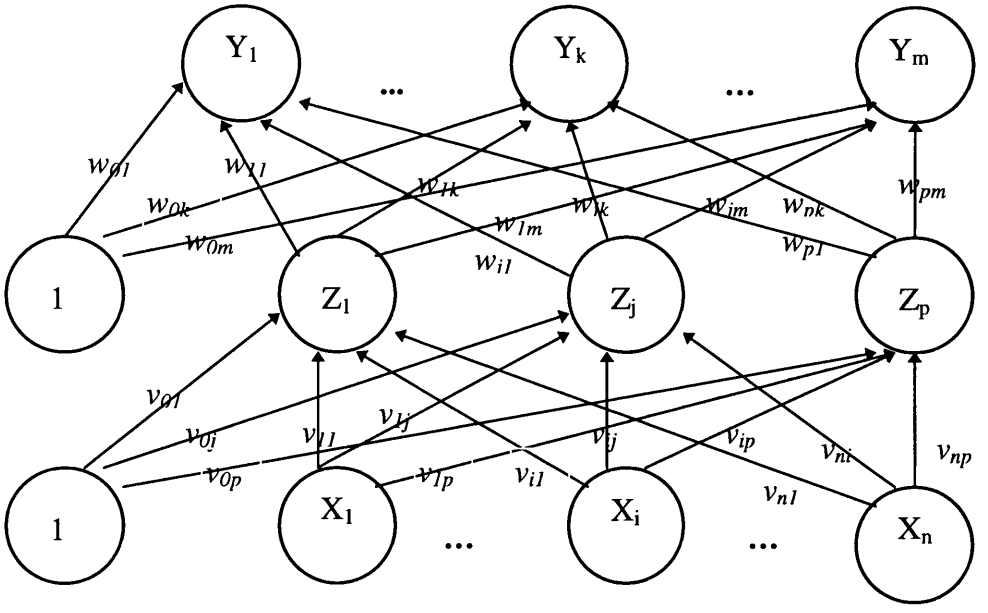


Figure 4.1 Backpropagation neural network (one hidden layer).

4.3 Three steps of Backpropagation

Feedforward step We refer to Figure 4.1. The input values from a training vector are fed forwards to the hidden units, which calculate their activations z_j. These numbers are passed on to the output units to determine their output values y_k.

Backpropagation of error Each unit compares its output y_k with target t_k and adds a term $(t_k - y_k)^2$ to an accumulating total squared error for the current epoch. For backpropagation it computes an error factor

$$\delta_k = (t_k - y_k)\, f'(y\_in_k). \qquad (4.1)$$

The δ_k are passed back to the hidden units to determine update weight increments Δw_{jk}, then they are passed further back and used along with the derivatives $f'(z\_in_j)$ to calculate increments Δv_{ij}. Details are given in Section 4.5.

The updating weights Only at this stage are weights updated by thir computed increments, when the 'old' values are finished with.

Summary of notation

x_i = The output signal from input unit.

z_j = The output signal from hidden unit.

y_k = The output signal from output unit.

z_in_j = The net input to the hidden unit Z_j.

y_in_k = The net input to the output unit Y_k.

etotal = The total squared error.

α = The learning rate.

w = The weight matrix $[w_{jk}]$ from hidden layer to output layer.

v = The weight matrix $[v_{ij}]$ from input layer to hidden layer.

ε_j = The associated error factor from input layer to the hidden unit Z_j.

δ_k = The associated error factor from hidden layer to the output unit Y_k.

4.4 Activation functions

We recall that the binary sigmoid function (whose values lie between 0 and 1), and its derivative, are given by

$$f_1(x) = 1/ [1+ e^{-x}], \tag{4.2}$$

$$f'_1(x) = f_1(x) [1 - f_1(x)]. \tag{4.3}$$

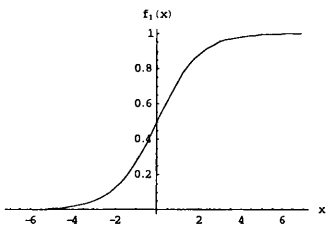
Its bipolar relative f_2, with values between ± 1, satisfies

$$f_2(x) = 2 f_1(x) - 1 = \tanh (x/2), \tag{4.4}$$

$$f'_2(x) = 2 f'_1(x). \tag{4.5}$$

Notice that the derivative function is related to gradient descent because it gives the slope, and we use it to find the error of weights. Because of (4.4) we will wish to compare the case of a third sigmoid $f_3 = \tanh x = f_2 (2x)$, for which $f_3'(x) = 2 f_2'(2x)$.

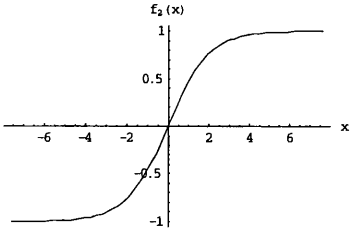
Activation Function Comparisons



Binary Sigmoid

f_1

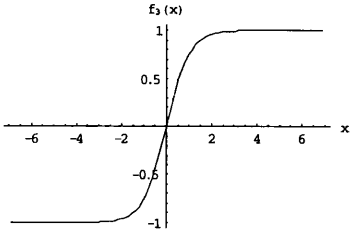
$f'_1(0) = ¼$



Bipolar Sigmoid

f_2

$f'_2(0) = ½$



Hyperbolic Tangent

$f_3 = f_2(2x)$

$f'_3(0) = 2 f'_2(0) = 1$

As graphs shown above, the slope of bipolar sigmoid is steeper than binary sigmoid. This affects the speed of training, as we discuss in Section 4.6.

4.5 Algorithm

We recall that unit Z_j has input z_in_j and output z_j, and similarly for Y_k. We write as usual $\mathbf{w} = [w_{jk}]$ and $\mathbf{v} = [v_{ij}]$ with j^{th} column $\mathbf{v}_j$. This algorithm is conveniently stated in terms of a procedure *Epoch* which runs through the training vectors in turn.

Set the activation function f = f_1, f_2 or f_3. Set the learning rate α.
Repeat Epoch *Until etotal* < 0.05.

Procedure Epoch
Set *etotal* = 0
Repeat for each training vector
(**Feedforward step**)
For $j = 1, ..., p$ do
$z\_in_j = \sum_i v_{ij} x_i$, $(i = 0, \ldots, n)$
$z_j = f(z\_in_j)$.
For $k = 1, \ldots, m$ do
$y\_in_k = \sum_j w_{jk} z_j$, $(j = 0, ..., p)$
$y_k = f(y\_in_k)$.
(**Backpropagation of error step**)
For $k = 1$ to m do
$etotal = etotal + (t_k - y_k)^2$.
$\delta_k = (t_k - y_k) f'(y\_in_k)$,
$\Delta w_{jk} = \alpha \delta_k z_j$, $(j = 0, ..., p)$
For $j = 1, ..., p$ do
$\delta\_in_j = \sum_k \delta_k w_{jk}$.
$\varepsilon_j = \delta\_in_j f'(z\_in_j)$
$\Delta v_{ij} = \alpha \varepsilon_j x_i$, $(i = 0, ..., n)$
(**Adjustment of weights and biases**)
$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$, $\mathbf{v} = \mathbf{v} + \Delta\mathbf{v}$.

4.6 Speed of Convergence

For the purpose of rough argument about influences on the number of epochs required for convergence, in the algorithm, let us summarise some relations whilst ignoring subscripts

$z = f(x)$, $y = f(z)$

Δw = error x $f'(y)$ x z

Δv = (the above) x $f'(z)$ x x.

(i) *Initial weights* (bipolar case)

Observe first that if x is small then so is $f(x)$. Thus, if the initial weights are small then so for example are the z_j, hence also the increments Δw_{jk}, and learning is slow. On the other hand if the weights are large then the derivatives defining δ_k are small, hence also the Δw_{jk}, and again learning is slow. One method of mitigating these considerations is to randomise initial weights between $\pm$ 0.5. a stronger effect is obtained by Nguyen - Widrow Initialization:

1. Choose the v_{ij} randomly between ± 0.5,
2. Choose scaling factor $\beta = (0.7)\, p^{1/n}$,
3. Recalculate $v_{ij} \rightarrow \beta\, v_{ij} / | \mathbf{v}_j |$.

(ii) *Sigmoids*

We have from Section 4.4 that $f_2'(x) = 2\, f_1'(x)$ and $f_3'(x) = 2\, f_2'(2x) > 2\, f_2'(x)$. Thus a given change in the argument (written above as x, though the same applys to z) tends to produce a greater weight update in bipolar cases (f_2) than in binary (f_1) , and still greater in hyperbolic. This is particularly important when x is close to zero and the net is in danger of freezing.

(iii) *Targets*

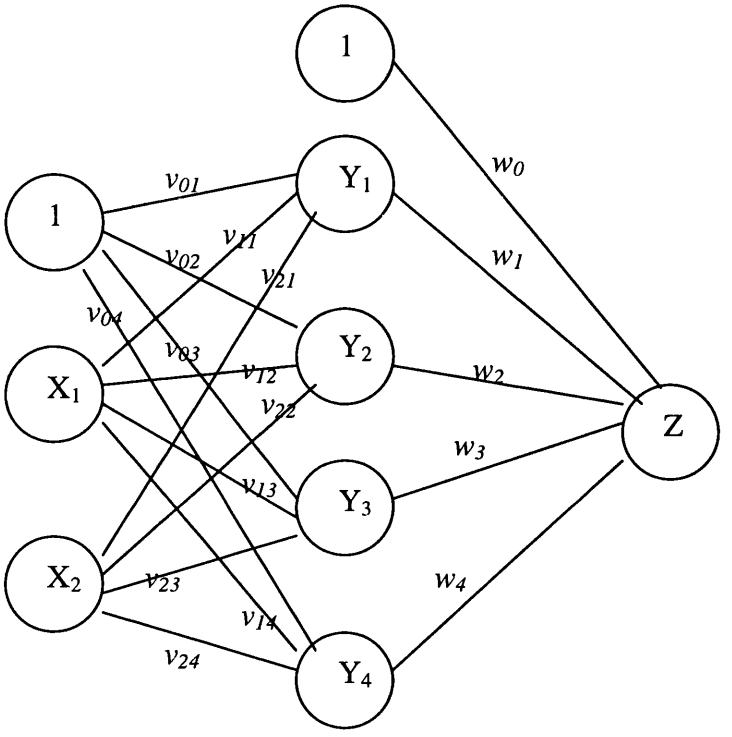
Consider the bipolar case. For f to be close to a target $\pm$ 1, the argument (x or z) must be rather large, and so it may be more realistic to use modified bipolar targets $\pm$ 0.8. Similar considerations apply to other sigmoids.

4.7 Numerical Experiments

We explore backpropagation for the XOR problem, studying the impact of various activation functions and weight initialisations (Cases 1-7). Our experiments reveal as expected that the choice of both initial weights and activation functions is crucial to the speed of convergence of the backpropagation algorithm.

The cases have these conditions in common:

1. Architecture



The four training input vectors are ((1,1,1),(1,1,0),(1,0,1),(1,0,0)) and for bipolar input we replace 0 by -1. *Modified bipolar* means changing the targets from ± 1 to ± 0.8. There are five hidden units in one hidden layer, and one output unit.

2. Learning rate $\alpha = 0.2$.

3. In Cases 1 - 3, the *initial weights* are randomised between ± 0.5 as shown below.

v = (0.4919, -0.2913, -0.3979, 0.3581, -0.1401).

w = ((0.1970,0.3099,-0.3378),(0.3191,0.1904,0.2771),(-0.1448,-0.0347,0.2859), (0.3594,-0.4861,-.3329)).

In Cases 4-7, this is extended to Nguyen - Widrow Initialisation (or N-W for short).

4. Stop condition *etotal* < 0.05.

4.7.1 Random initial weights

We compare binary, bipolar, and modified bipolar techniques leading to successively faster convergence.

Case 1 Binary input and sigmoid function (targets ± 1).

The results are shown in Table 4.1 and Figure 4.4.

Table 4.1 The XOR problem with binary sigmoid activation function and random weights.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.215246	0.297605	0.297926	0.222083	1.03286
.					
.					
.					
2891	0.0107437	0.0116026	0.0129801	0.014661	0.0499874

This training is relatively slow. It took 2,891 epochs as illustrated in the graph of Figure 4.4. The total squared error starts at 1.03286. The error is fairly constant over the first 1,000 epochs of training, and after 1,500 epochs, the error decreases very quickly. For this example, this training freezes when the total squared error is less than 0.05 (Figure 4.4).

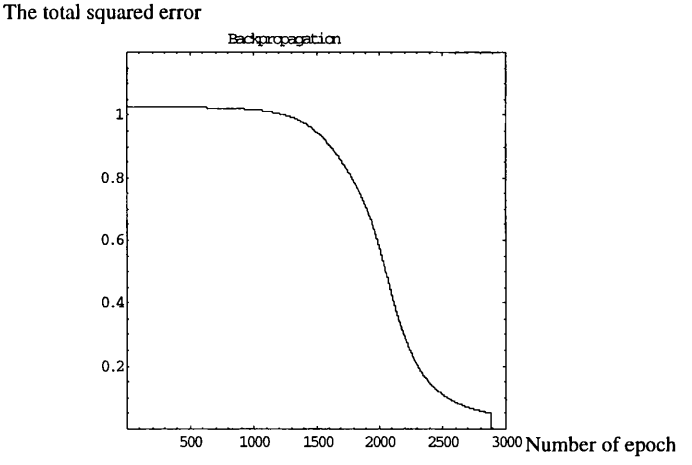


Figure 4.4 The total squared error of a backpropagation net solving the XOR problem: binary representation.

Case 2 Bipolar sigmoid, input vectors, and targets.

The result is shown in Table 4.2.

Table 4.2 The XOR problem with bipolar sigmoid activation function and random weights.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.716406	1.3736	1.54981	0.7976	4.4372
.					
.					
.					
387	0.00887451	0.0151397	0.0164533	0.00945053	0.049918

This training took 387 epochs. The starting total squared error is more than 4; we recall that the starting total squared of the binary case is 1. The net using the bipolar sigmoid function and the bipolar representation learns faster than the binary representation. After 100 epochs, the total squared error reduced very quickly. Next, it reduces slowly after 250 epochs (Figure 4.5). The total epoch is 387 epochs.

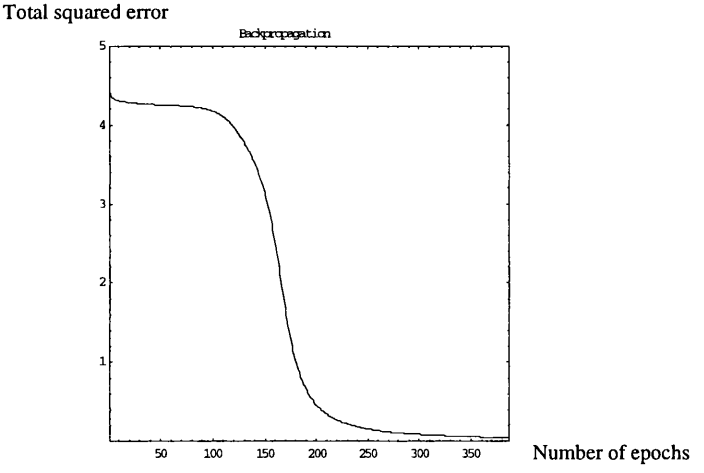


Figure 4.5 This graph shows the total squared error of a backpropagation net solving the XOR problem with the bipolar representation.

Case 3 Bipolar sigmoid function.

We use the bipolar training input vectors as before, but targets of ± 0.8 called *modified bipolar targets* (as mentioned earlier). The result is shown in Table 4.3.

Table 4.3 The XOR problem with bipolar sigmoid activation function, random weights, and modified bipolar targets.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.417843	0.923666	1.08301	0.458727	2.88324
.					
.					
.					
264	0.00763681	0.0149499	0.0184417	0.00812198	0.0491504

A total of 264 epochs was used. This net learns faster than the net using the binary representation (Case 1), and the bipolar representation (Case 2). The starting total squared error is almost 3. Hence, this error of the modified bipolar is less than the error of the binary and bipolar values. For the first 100 epochs, the total squared error reduces very slowly, then it learns quicker after 150 epochs. Finally, it starts to reduce the error slowly again after 200 epochs. To summarise, the net using the modified bipolar gives the better result comparing to the net using the binary and bipolar representations (Figure 4.6).

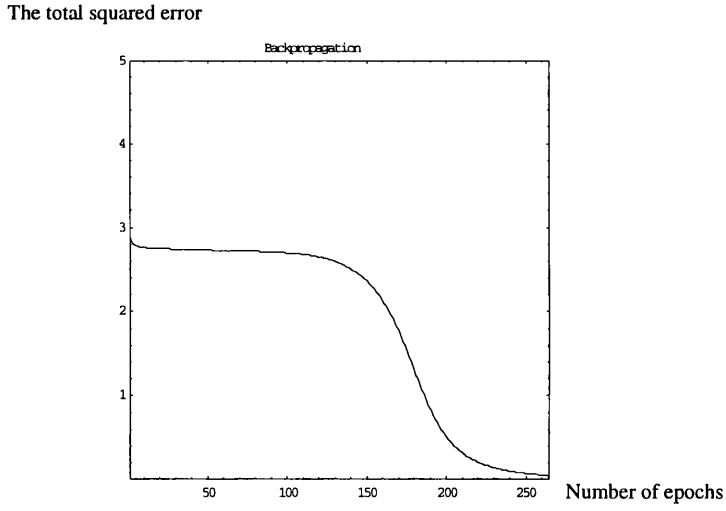


Figure 4.6 The total squared error of a backpropagation net solving the XOR problem with modified bipolar representation.

4.7.2 Nguyen - Widrow weight initialization

Case 4 Hyperbolic tangent sigmoid, binary input vectors and bipolar targets.

The result as shown in Table 4.4.

Table 4.4 The XOR problem with Hyperbolic tangent, and Nguyen - Widrow weight initialization.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.321354	2.36025	1.65856	0.0153233	4.35549
.					
.					
.					
77	0.00296968	0.0141104	0.0316902	0.0010327	0.049803

This training took 77 epochs (Figure 4.7). At the beginning of this training, the total squared error reduces very fast from (approximately) the total squared error 4.4 to 1.2.

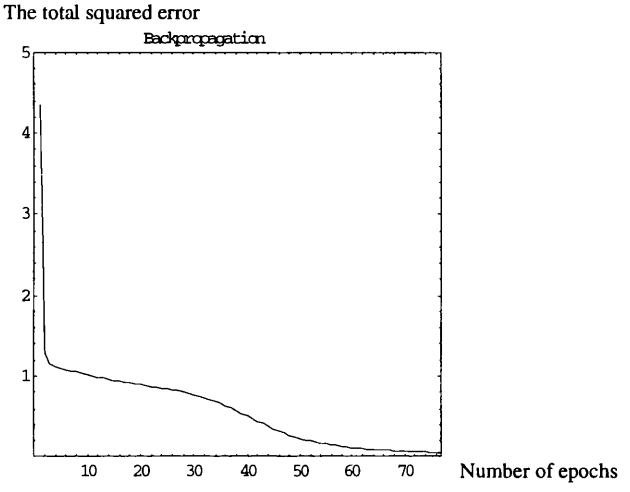


Figure 4.7 The total squared error of the backpropagation net. The activation function is the Hyperbolic tangent with the binary XOR problem.

Case 5 Bipolar sigmoid, input vectors, and targets.

The result is shown in Table 4.5

Table 4.5 The XOR problem with bipolar sigmoid function, and Nguyen - Widrow weight initialization.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.560661	1.714	2.04653	0.590102	4.91129
.					
.					
.					
248	0.0131279	0.00419114	0.0176815	0.0149265	0.0499271

This training took 248 epochs. This result is much better than the result of Case 2. We may say that the Nguyen-Widrow weight initialization can improve the training speed. The total squared error reduces very fast after thirty epochs and decreases very slowly after 100 epochs (Figure 4.8).

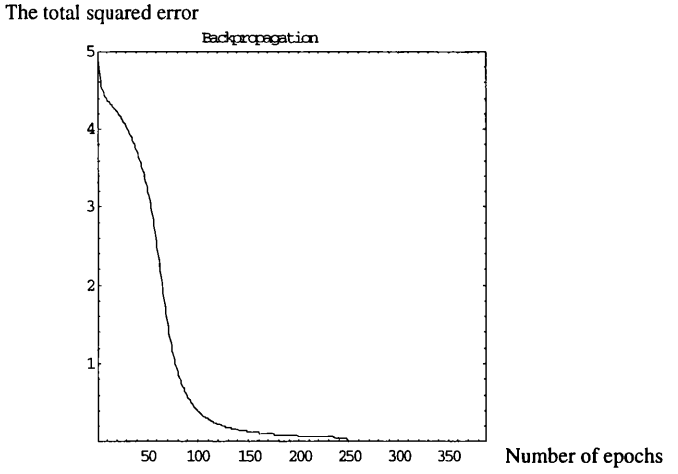


Figure 4.8 The total squared error of a backpropagation net with bipolar XOR, using the Nguyen-Widrow weight initialization to set the initial weights.

Case 6 Hyperbolic tangent sigmoid, bipolar input vectors and bipolar targets.

The result as shown in Table 4.6.

Table 4.6 The XOR problem with Hyperbolic tangent, and Nguyen - Widrow weight initialization.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.187592	2.87575	3.26446	0.413912	6.74172
.					
.					
.					
52	0.0142838	0.00287339	0.0190102	0.0134874	0.0496548

This training took 52 epochs (Figure 4.9). At the beginning, the total squared error is about 6.74. The graph decreases very quickly two times- the first five epochs and between 10 and 15 epochs. The training speed of this example is slower than the 25 epochs of Example 4.4. This figure is similar to Figure 4.7 (Example 4.4). The difference between this example and Example 4.6 is that this example uses the Hyperbolic tangent as its activation function rather than the bipolar sigmoid function.

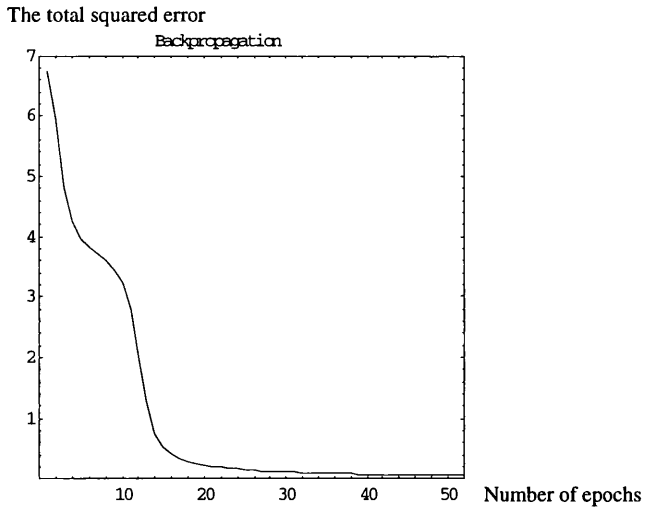


Figure 4.9 The total squared error of the backpropagation net using the Nguyen-Widrow weight initialization. The activation is Hyperbolic tangent with the bipolar XOR problem.

Case 7 Hyperbolic tangent, bipolar input vectors and modified bipolar targets.

We use Nguyen - Widrow weight initialization to initialise weights. The result is shown in Table 4.7.

Table 4.7 The XOR problem with Hyperbolic tangent, Nguyen - Widrow weight initialization, and modified bipolar data.

Epoch	e_1	e_2	e_3	e_4	Total squared error
1	0.0543445	2.18123	2.53693	0.171688	4.9442
.					
.					
.					
25	0.0133891	0.000442885	0.0193827	0.0103376	0.0435523

A total of 25 epochs was used (Figure 4.10). The total squared error reduces quickly within the first five epochs and again from epochs 10 to 15. Notice that this training uses the smallest epoch number of all our examples.

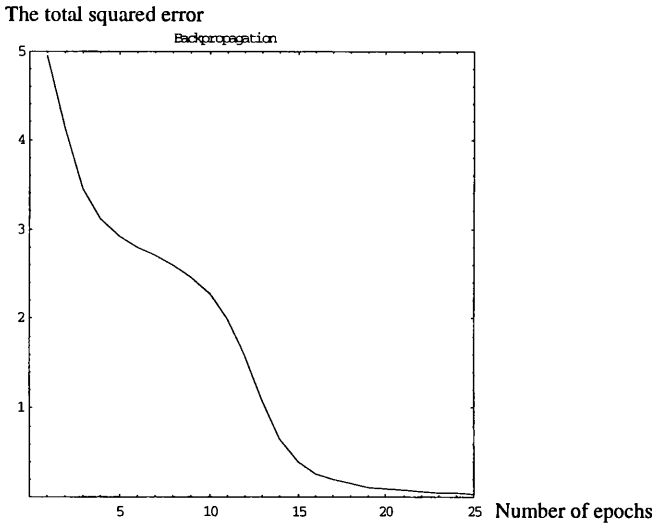


Figure 4.10 The total squared error of a backpropagation net solving the XOR problem. This net uses Nguyen-Widrow weight initialization, bipolar input vectors and the modified bipolar targets. The activation function is Hyperbolic tangent.

4.8 Summary

Table 4.8 below shows the total number of epochs for each case. There are two methods of choosing initial weights.

Table 4.8 We summarize all results of experiments which compare the activation function and the initialization weights in solving the XOR problem.

	Initialization weights	
	Random [Section **4.7.1**]	*Nguyen-Widrow* [Section **4.7.2**]
f_1, binary input vectors and binary targets.	2,891 epochs (Case 1)	-
f_2, bipolar input vectors and bipolar targets.	387 epochs (Case 2)	248 epochs (Case 5)
f_2, bipolar input vectors and modified bipolar targets ± 0.8.	264 epochs (Case 3)	-
f_3, binary input vectors and binary targets.	-	77 epochs (Case 4)
f_3, bipolar input vectors and bipolar targets.	-	52 epochs (Case 6)
f_3, bipolar input vectors and modified bipolar targets ± 0.8.	-	25 epochs (Case 7)

Case 4 versus Case 6: Difference is in input binary versus bipolar.
Case 5 versus Case 6: Difference is in activation function.
Case 6 versus Case 7: Difference is in targets only.

Conclusion For the XOR problem bipolar training is faster than binary, and is further speeded up by the use of modified (bipolar) targets. The Nguyen-Widrow weight initialization can improve the speed of training. If the net uses Hyperbolic tangent sigmoid with modified bipolar targets, and Nguyen-Widrow weight initialization, then the speed is very fast indeed.

4.9 Data compression

In this section, we use the backpropagation neural network to compress data. The training input vector is the same as the target output vector. This example will use data from the set of characters. We represent each character by 8x7 pixels as shown in Figure 4.11 below.

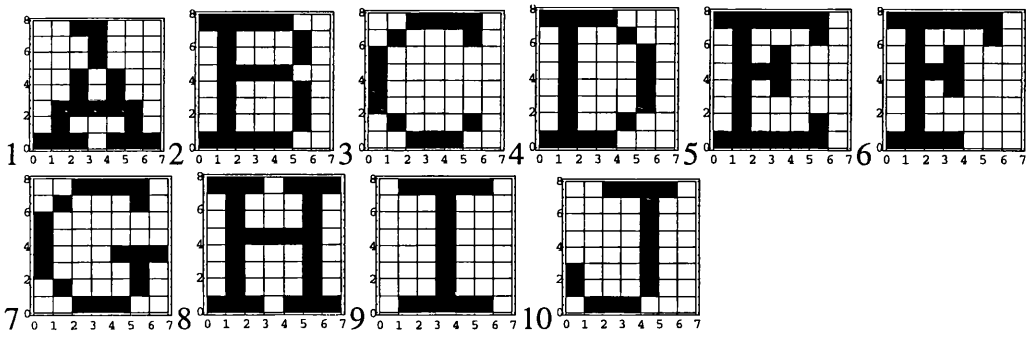


Figure 4.11 Ten characters are used in the data compression.

There are 56 input units in the input layer and 56 output units in the output layer. The hidden units are used to compress data, the output units for restoration. The number of hidden units are varied each training time. We also use the initial random weights each training time.

How compression occurs Since N = 10 distinct patterns are to be identified, we require enough hidden units to transmit 10 distinct vector. The key observation is that $2^3 < 10 < 2^4$, and so four hidden units suffice in the present case (more generally $\log_2 N$ units suffice). Thus, the number of bits used to specify a character is reduced from 56 to 4, a compression ratio of 14:1.

The net will learn when every calculated output value is within a defined tolerance of the required values (0 or 1). We use two tolerances- 0.2 and 0.8 in this example. If the activation is less than or equal to 0.2, then this unit is 'off'; and if the activation is

greater than or equal to 0.8, then this unit is 'on'. In the same manner, if the activation is less than or equal to 0.1, then this unit is 'off', and if the activation is greater than or equal to 0.9, then this unit is 'on'. The accuracy of these results was evaluated in terms of 100% correctness of the reconstructed characters in the training set.

A **Mathematica Module**

Program 1 We use the backpropagation net to compress data. We set the tolerance and the learning rate to 0.2. We randomise the initial weights **w**, **v** between -0.5 and 0.5, and use the Nguyen-Widrow Initialization to adapt the initial weights **v**. The target vectors are in binary form.

Result

The number of hidden units	*Total error*	*Total epochs*
10	2.5	263
11	5.5	216
12	4.5	235
13	0.5	199
14	1.5	191
15	4.5	173
16	4.5	186
17	6.5	172
18	6.0	196
19	6.0	175
20	2.0	160

Total time used = 910 seconds. Notice that the best number of hidden units is 13 because it obtains the smallest total error 0.5, and its number of epochs is not excessive compared with other cases. The compression ratio is 56:13, or about 4:1.

Program 2 We use the backpropagation net to compress data. We set the tolerance and the learning rate to 0.1. We randomise the initial weights **w**, **v** between -0.5 and 0.5, and use the Nguyen-Widrow Initialization to adapt the initial weights **v**. The target vectors are in binary form. Note that the difference condition between Program 1 and Program 2 is the value of the tolerance.

Result

The number of hidden units	*Total error*	*Total epoch*
10	0	589
11	0	660
12	0	565
13	0.5	432
14	0	526
15	0.5	387
16	0.5	463
17	0	383
18	1	380
19	0	336
20	1.5	352

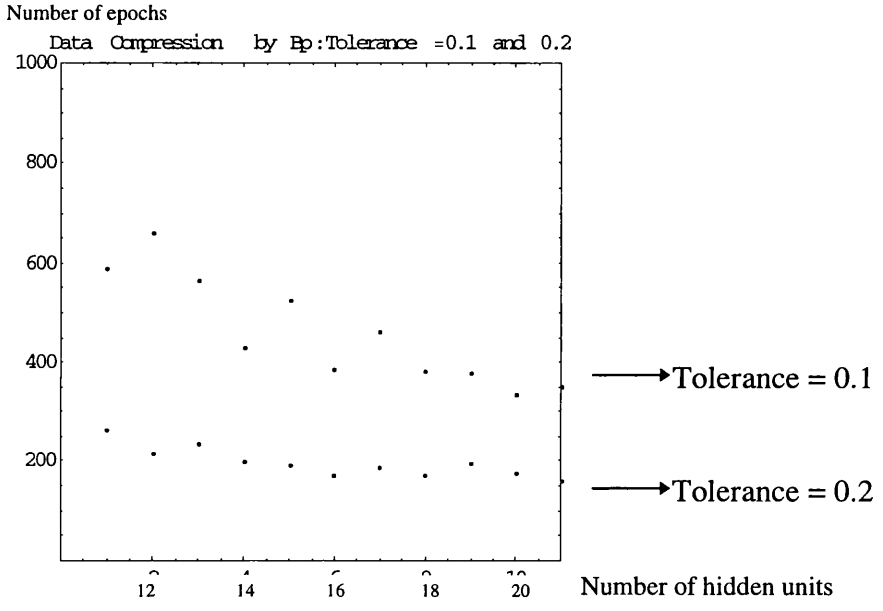


Figure 4.12 Number of epochs required as a function of number of hidden units for two tolerance- 0.1 and 0.2.

Conclusion This graph shows that when the tolerance value increases from 0.1 to 0.2, the number of epochs reduces to less than 300 epochs. The net learns faster but the accuracy is reduced because of setting the value of tolerance.

Part 2 Statistical Shape Models

Chapter 5 SVD, PCA, and Active Shape Models

Introduction

The purpose of machine vision is to recover the image structure and try to analyze what it means. Real Images are complicated and difficult to analyze because of noise. We need to find a model which gives proper examples. One method using Statistical Shape Models can solve the determined patterns of variability in shape and gray-level appearance. This method is very good because it can classify noisy and incomplete images, and also label the recovered structures. We approach this method for recognizing and locating known rigid objects. It may be used for example in medical image interpretation and face recognition so that we can know automatically what images represent. When we adjust the shape parameters, we can build the synthetic shapes which are similar to real face shapes. We will deal with variables because each person has his (or her) special characteristics.

We can use Statistical Shape Models to locate the structure in a target image. Cootes [15] uses his Active Shape Model (ASM) to match a Statistical Shape Model to a data set of shapes. He can synthesise and analyse a variety of new shapes which are similar to shapes in an original shape set. After we have these generating shapes, we can use them for other processing, for example, classification. The method of building new shapes involves *Singular Value Decomposition* (SVD) in Section 5.1, and *Principal Component Analysis* (PCA), in Section 5.2, for which we will explain the basic ideas below. Then we will show how to label the original images and how to align every shape in a common co-ordinate frame using *Procrustes Analysis* in Section 5.3. Next we will explain how to generate new shapes. Finally, we use this Statistical Shape Model to build new shapes of Hand and Face in Section 5.4.

5.1 Singular Value Decomposition (SVD)

It is well-known that a symmetric matrix **A** can be diagonalised by some orthogonal matrix **P**. That is, $\mathbf{PAP}^T$ is a diagonal matrix **D**, or $\mathbf{A} = \mathbf{P}^T\mathbf{DP}$. The *Singular Value Decomposition*, or SVD, goes further: an arbitrary m x n matrix can be diagonalised if we allow *two* orthogonal matrices.

Theorem 5.1 (SVD) Let **A** be an m x n matrix of rank r. Then there exist orthogonal matrices **U** and **V** such that

$$\mathbf{A} = \mathbf{U}^T\mathbf{DV}, \qquad (5.1)$$

where the m x n matrix **D** is zero off its main diagonal,

$$\mathbf{D} = \begin{bmatrix} s_1 & 0 & \dots & 0 & 0 & 0 \\ \cdot & \cdot & & & & \cdot \\ \cdot & & s_r & & & \cdot \\ \cdot & & & 0 & & \\ 0 & \dots & & & 0 & 0 \end{bmatrix}, \qquad (m \text{ x } n)$$

and $s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_i \geq \ldots s_r > 0$.

Note The s_i are called the *singular values* of **A**.

Remark 5.2 The following idea is used in the Mathematica implementation we utilise. The formula (5.1) still holds if we replace all but the first r rows of **U** and **V** by zeros (**U** and **V** with these zero rows deleted are called *row-orthogonal*).

Proof Let **U** have rows $\mathbf{U}_1, \ldots, \mathbf{U}_m$, and **V** have rows $\mathbf{V}_1, \ldots, \mathbf{V}_n$. Then,

$$\mathbf{A} = \mathbf{U}^T\mathbf{DV} = [\mathbf{U}_1^T \ldots \mathbf{U}_m^T] \begin{bmatrix} s_1 & 0 & \dots & 0 & 0 & 0 \\ \cdot & \cdot & & & & \cdot \\ \cdot & & s_r & & & \cdot \\ \cdot & & & 0 & & \\ 0 & \dots & & & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1 \\ \cdot \\ \cdot \\ \mathbf{V}_n \end{bmatrix}$$

$$= [s_1\mathbf{U}_1^T \ldots s_r\mathbf{U}_r^T \; 0 \ldots 0] \begin{bmatrix} \mathbf{V}_1 \\ \cdot \\ \cdot \\ \mathbf{V}_n \end{bmatrix}$$

$$= \sum_i s_i\mathbf{U}_i^T\mathbf{V}_i \quad (1 \leq i \leq r).$$

Remark 5.3 We shall shortly need information about matrices of the form $\mathbf{AA}^T$. For this, the SVD gives

$$\mathbf{AA}^T = \mathbf{U}^T\mathbf{DVV}^T\mathbf{D}^T\mathbf{U},$$
$$= \mathbf{U}^T(\mathbf{DD}^T)\mathbf{U}, \quad \text{since } \mathbf{VV}^T = \mathbf{I}.$$

Hence,
$$(\mathbf{AA}^T)\mathbf{U}^T = \mathbf{U}^T(\mathbf{DD}^T), \tag{5.2}$$

where $\mathbf{DD}^T$ is the diagonal matrix diag $(s_1^2, s_1^2, \ldots, s_i^2, \ldots, s_r^2, 0, \ldots, 0)$. But formula (5.2) says that the matrix $\mathbf{AA}^T$ has eigenvalues $\lambda_i = s_i^2 > 0$ $(1 \le i \le r)$ and $\lambda_i = 0$ for $i > r$, with a corresponding *orthonormal set* of eigenvectors given by the rows of $\mathbf{U}$.

5.2 Principal Component Analysis (PCA)

Suppose we have data consisting of samples $\mathbf{X}_1, \ldots, \mathbf{X}_s$ of a random vector. Assuming a *mean* of zero, Principal Component Analysis, or PCA, proceeds briefly as follows. The *first principal component* of the data is a linear combination which exhibits as much as possible of the total variance. A unit vector in the direction of this component is called the *first principal axis* (Figure 5.1). We subtract from each data vector its component along this axis. Then the first principal component of the modified data is called the *second* principal component of the original data, and so on. The result is conveniently obtained by the SVD, as we now describe. If the data has mean $\bar{\mathbf{X}} = (1/s)\sum_k \mathbf{X}_k$ $(1 \le k \le s)$, its *covariance matrix* is defined to be

$$\mathbf{S} = 1/(s\text{-}1)\sum_k (\mathbf{X}_k - \bar{\mathbf{X}})(\mathbf{X}_k - \bar{\mathbf{X}})^T \qquad (1 \le k \le s). \tag{5.3}$$

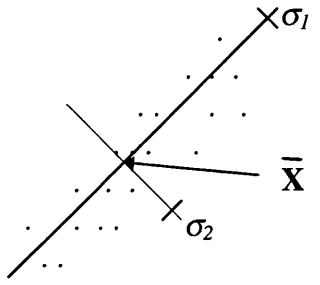


Figure 5.1 By applying PCA to the data, we can find the principal axes of a cloud of n plane points. Here σ_1^2 is the variance of the first principal component, which lies along the first principal axis, the most important direction represented in the data. The component in the subsidiary second direction has variance σ_2^2.

Remark 5.4 $\mathbf{S} = 1/(s\text{-}1)\ \mathbf{A}\mathbf{A}^T$, where $\mathbf{A}$ has k^{th} column $\mathbf{X}_k - \bar{\mathbf{X}}$ $(1 \leq k \leq s)$.

Proof Write $\mathbf{X}_k - \bar{\mathbf{X}} = \mathbf{Z}_k$.

Then, $a_{pk} = (\mathbf{Z}_k)_p$, the p^{th} element of $\mathbf{Z}_k$, and so by definition of matrix multiplication,

$$(\mathbf{A}\mathbf{A}^T)_{ij} = \sum_k a_{ik}\, a_{jk} = \sum_k (\mathbf{Z}_k)_i\, (\mathbf{Z}_k)_j \qquad (5.4)$$

$$= \sum_k (\mathbf{Z}_k \mathbf{Z}_k^T)_{ij}, \text{ as required.}$$

Remark 5.5 From formula (5.4) the $(i, j)^{th}$ element of $\mathbf{S}$ is

$$1/(s\text{-}1)\sum_k (\mathbf{Z}_k)_i\, (\mathbf{Z}_k)_j,$$

which is the (estimated) correlation between the i^{th} and j^{th} variables; the i^{th} diagonal element, case $j = i$, is

$$1/(s\text{-}1)\sum_k (\mathbf{Z}_k)_i^2,$$

namely the variance of the i^{th} variable. Hence $\mathbf{S}$ is also called a *correlation matrix.*

Remark 5.6 The principal axes of the data may be taken to be the orthonormal set of eigenvectors $\mathbf{U}_1, \ldots, \mathbf{U}_m$ for $\mathbf{S}$, obtained by the SVD for $\mathbf{A}$. The reason is that in choosing the $\mathbf{U}_i$ as new coordinate axes we make the following changes:

$$\mathbf{X}_k - \bar{\mathbf{X}} = \mathbf{Z}_k \rightarrow \mathbf{U}\mathbf{Z}_k.$$

Hence, $\mathbf{A} \rightarrow \mathbf{U}\mathbf{A}, \quad (\mathbf{A} = [\mathbf{Z}_1 \ \ldots \ \mathbf{Z}_s])$

and $\mathbf{A}\mathbf{A}^T \rightarrow \mathbf{U}\mathbf{A}(\mathbf{U}\mathbf{A})^T = \mathbf{U}(\mathbf{A}\mathbf{A}^T)\mathbf{U}^T$

$$= \text{diag}\,(s_1^2, \ldots, s_r^2, 0, \ldots, 0)$$

(by Remark 5.3), the correlation matrix of the new variables $y_1, \ldots, y_m$. Thus, the new (zero mean) variables are listed in decreasing order of variance, and are uncorrelated. Also, as will be illustrated in the examples of the next section, we have reduced the dimensionality of the data to r, or with little loss, to some $t < r$.

Note 1 Inserting the factor s-1 to convert $\mathbf{A}\mathbf{A}^T$ to $\mathbf{S}$ simply scales the eigenvalues/variances by $1/(s\text{-}1)$.

(*Brief reason*: $\mathbf{B}\mathbf{V} = \lambda\mathbf{V} \Leftrightarrow (\alpha\mathbf{B})\mathbf{V} = (\alpha\lambda)\mathbf{V} \quad (\alpha \neq 0)$. Now put $\mathbf{B} = \mathbf{A}\mathbf{A}^T$ and $\alpha = 1/(s\text{-}1)$.)

Note 2 If there are significantly fewer variables than samples we may wish to replace $\mathbf{AA}^T$ by the smaller matrix $\mathbf{A}^T\mathbf{A}$. This is valid essentially because if $\mathbf{x}$ is an eigenvector of $\mathbf{AA}^T$ with eigenvalue $\lambda \neq 0$, then $\mathbf{x}$ is an eigenvector of $\mathbf{A}^T\mathbf{A}$ with the same eigenvalue. $[(\mathbf{AA}^T)\mathbf{x} = \lambda\,\mathbf{x} \Rightarrow (\mathbf{A}^T\mathbf{A})\,\mathbf{A}^T\mathbf{x} = \lambda\,\mathbf{A}^T\mathbf{x}]$

5.3 Active Shape Models

Suppose we have s images which are considered as variations of one shape (e.g. a hand or face). We aim to represent each image by a shape vector $\mathbf{X}_j$ $(1 \leq j \leq s)$ of say n points, correspondingly placed in each image. The $\mathbf{X}_j$ are referred to as *training examples* for the shape.

5.3.1 Landmarking the images

To annotate a shape in this way we must consider how to select good positions. Every angle is a good landmark which we can recognize very easily, for example a **T**-junction or a **V**-junction in Figure 5.2. Where we cannot see any junctions or points of high curvature, we will divide the outline by equally spaced points. Each landmark point shows a special part of the object or its boundary, and it must be consistent from one shape to the next - each point is annotated in a similar way.

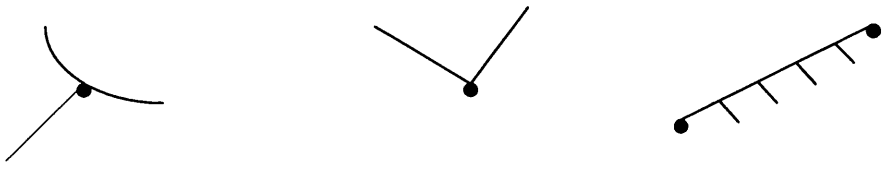


Figure 5.2 A T-Juction, a V-Junction, and some filled-in points.

In principle we can create a data set in any dimension d, and then a shape vector of n points will have dimension nd. But for simplicity we use plane points $z = (x,y)$, $d = 2$, and write a shape vector as $\mathbf{X} = (z_i)$ or in expanded form

$$\mathbf{X} = (z_1, z_2, \ldots, z_n)^T. \qquad (5.5)$$

The next step is to standardise our shape vectors for ease of comparison, using suitable Euclidean Transformations of the plane.

5.3.2 Aligning one shape to another

We begin with the idea of aligning a shape $\mathbf{X}_j$ to a shape $\mathbf{X}$. This means that we perform on every point of $\mathbf{X}_j$ the same rotation θ_j, uniform scaling by s_j, and translation $\mathbf{t}_j$, converting $\mathbf{X}_j$ to say $T_j(\mathbf{X}_j)$, and chosen to minimise the sum of squared differences

$$|\mathbf{X} - T_j(\mathbf{X}_j)|^2.$$

The full standardisation is carried out with respect to some shape $\mathbf{X}_0$, by the iterative procedure below (see Section 5.4 for examples).

5.3.3 Aligning all shapes together (Standardisation)

We translate each shape $\mathbf{X}$ so that the centre of gravity of its points becomes the origin by calculating:

$$z = (1/n) \sum_k z_k \qquad \text{then} \qquad z_j \rightarrow z_j - z \qquad (1 \leq j,k \leq n).$$

Choose $\mathbf{X}_0$ of unit length, say $\mathbf{X}_0 = \mathbf{X}_1 / |\mathbf{X}_1|$ (used as first mean estimate).

Repeat

1. Align each shape to current estimate of mean
2. Recalculate the mean $\overline{\mathbf{X}} = 1/s \sum \mathbf{X}_j$
3. Align the mean with $\mathbf{X}_0$ and scale it to unit length

Until New mean close enough to old.

5.3.4 Generating new shapes

We apply PCA to our training data, to identify for each vector $\mathbf{b}$ (of suitable length) a shape $M(\mathbf{b})$, the model defined by $\mathbf{b}$. The objective is to generate from the data a class of models which are both acceptable as examples, *and* of sufficient variety.

Motivation for the method In Section 5.2 we showed how to compute from the covariance matrix of the data a sequence of principal axes, namely the orthonormal vectors $\{\mathbf{U}_i\}$, so that we can transform to new uncorrelated variables $y_1, ..., y_m$ where

$$\mathbf{X} = \overline{\mathbf{X}} + [\mathbf{U}_i^T \ldots \mathbf{U}_m^T] \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ y_m \end{bmatrix}. \qquad (5.6)$$

We recall that the variables y_i have zero mean and variances of decreasing sizes, given by

$$V(y_1) = \lambda_1, V(y_2) = \lambda_2, ..., V(y_r) = \lambda_r,$$

$$V(y_i) = \lambda_i = 0, \text{ for } i > r.$$

Thus the modes and variables are listed from the most important down to the least.

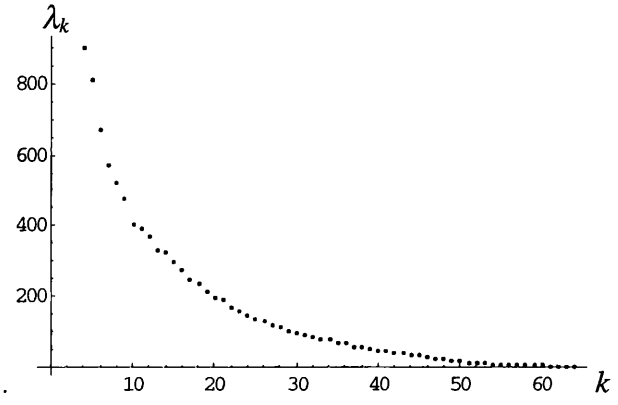


Figure 5.3 This figure shows eigenvalues λ_k decreasing with k.

Now, most of a normally distributed population lies within 3 standard deviations of the mean, in fact a normal random variable y with zero mean and variance σ^2 satisfies

$$P(-3\sigma \le y \le 3\sigma) = 0.997,$$

Hence, assuming the y_k to be normally distributed, we would have

$$P(-3\sqrt{\lambda_k} \le y_k \le 3\sqrt{\lambda_k}) = 0.997.$$

Guided by this, we vary shapes as mentioned by varying a vector **b** of parameters $b_1, ..., b_t$ to obtain shapes M(**b**) of the form

$$\mathbf{X} = \overline{\mathbf{X}} + [\mathbf{U}_1^T \ldots \mathbf{U}_t^T] \begin{bmatrix} b_1 \\ \cdot \\ \cdot \\ b_t \end{bmatrix} \qquad (-3\sqrt{\lambda_k} \le b_k \le 3\sqrt{\lambda_k}) \qquad (5.7)$$

where we use the t most important *modes of variation* $\mathbf{U}_1, ..., \mathbf{U}_t$, for suitable choice of $t \leq r$. In particular, a given proportion f_v of the total variance is supplied ("explained") by these t modes provided

$$\lambda_1 + ... + \lambda_t \geq f_v \sum_i \lambda_i \quad (i = 1,..., r). \tag{5.8}$$

Thus we choose $f_v = 0.98$ for example, and take the least t satisfying (5.8). When we know the number t of modes to retain, we can estimate any training shapes to within a given accuracy. To check the accuracy we increase the number of modes and test how the new images can represent the training set, and we choose the first model that passes our desired standard.

5.4 Using Active Shape Models to generate new shapes

In our project, we are interested in Hand and Face shapes. We will use this Statistical Shape Model to build new generated shapes as follows. The first section introduces 'hand shapes'.

(I) Hand Shape Models

Program 5.1 We will generate new hand shapes from twenty original hand shapes (Figure 5.4) by using forty-six landmark positions. Therefore a data set of images $\mathbf{X}_j$ are 92-tuple shape vectors, where $j = 1, ..., 20$. These hands are drawn manually on grid paper. The true landmarks are the points at the junctions (or angles) and between each two true landmarks we will divide into equal distance for each point as noted previously.

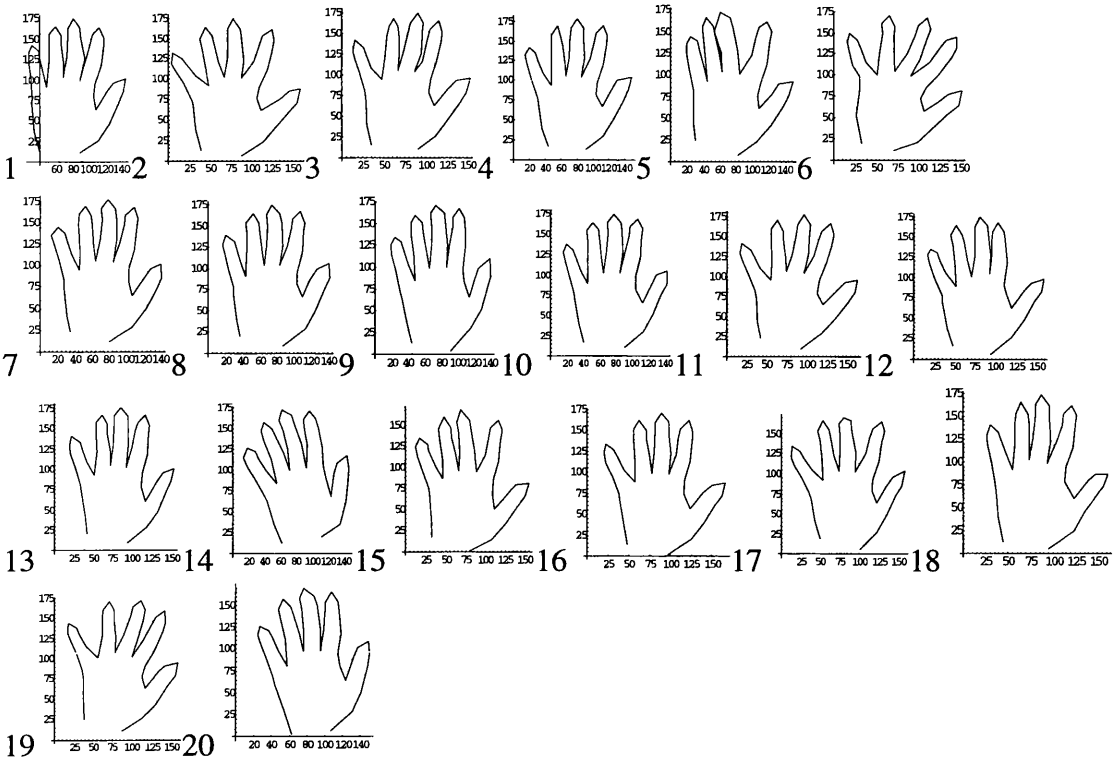


Figure 5.4 The twenty hands used as training data.

This Program will align the hands, calculate a covariance matrix **S**, and then use SVD and PCA to generate new hands as described in Section 5.3.

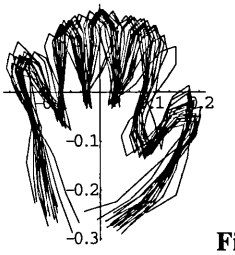


Figure 5.5 After we have translated, and scaled 20 original hands, we have these hands.

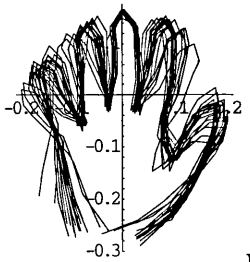


Figure 5.6 After we have fixed the orientation, we have these hands.

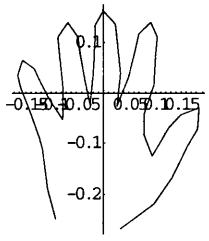


Figure 5.7 This is a mean shape from 20 original hands.

Figure 5.8 below shows the 5^{th} - 7^{th} shapes after alignment.

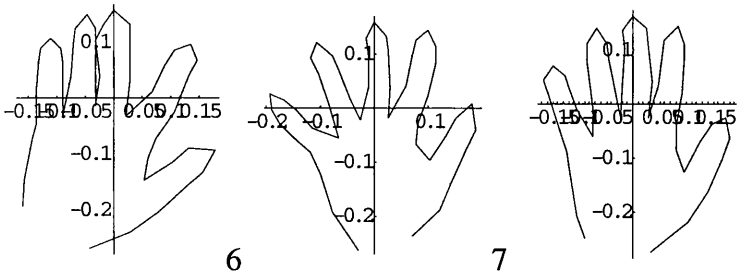


Figure 5.8 These are alignment examples from the original shapes 5^{th}, 6^{th}, and 7^{th}.

After we have aligned every hand, we can generate new shapes. This program obtains 1,120 synthetic images from twenty original hand shapes. Below, we show examples of varying the parameter b_k for each mode $\mathbf{U}_k$ in turn within the range $\pm\sqrt{\lambda_k}$. There are eight nonzero eigenvalues, of which we use the greatest six.

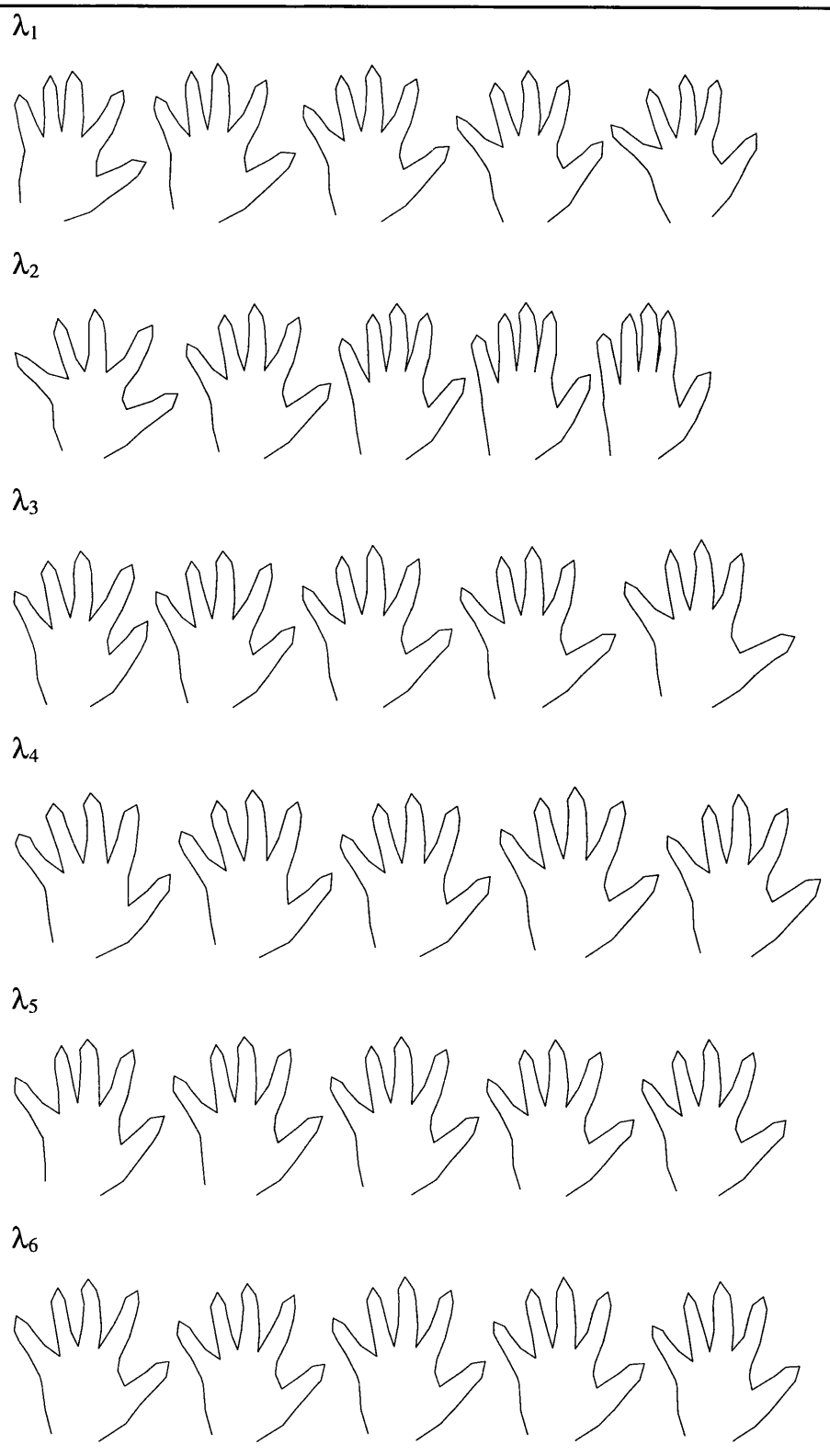


Figure 5.9 We use each λ_k to generate shapes in a few different manners. These figures show some examples of them.

(II) Face Shape Models

The images are taken in the same position by using a common camera. We use a photocopying machine to magnify and print on grid paper before labeling the coordinate system of points manually.

For face shapes, we have two Mathematica Implementations. The difference between these implementation is the number of original face shapes (training examples)- Program 5.2 uses 9 original face shapes, and Program 5.3 uses 30 original face shapes. The purpose is to know how generated shapes improve when the number of original shapes increases.

Program 5.2 This program uses 9 original face images (Figure 5.11) with 93 landmark points for generating faces. Hence each training example represents a 186-tuple vector. Every image has the same order of the coordinate system. That means the first position of the first image is the same as the second image, etc.

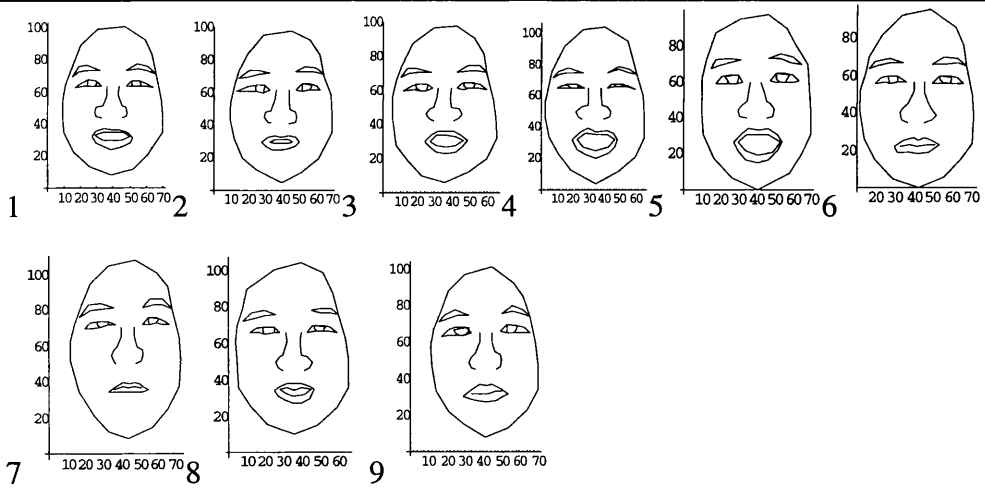


Figure 5.10 The nine original faces for Program 5.2.

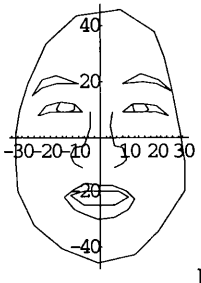


Figure 5.11 The first original shape is aligned.

We use seven eigenvalues λ_k, i.e. $t = 7$, and the parameters b_k are varied within limits $\pm\sqrt{\lambda_k}$ in this program (usually $\pm 3\sqrt{\lambda_k}$). One original face can produce 7 x 8 = 56 generated shapes so that nine original faces build 504 images. These generated shapes are shown in Chapter 6.

Program 5.3 (Face Project) Previously we used 9 original face shapes, and now we increase the number of face images to 30 (Figure 5.12). Each image has 92 landmarks. We use an implementation which is very similar to Program 5.2. After aligning face shapes, we obtain those as shown in Figure 5.12.

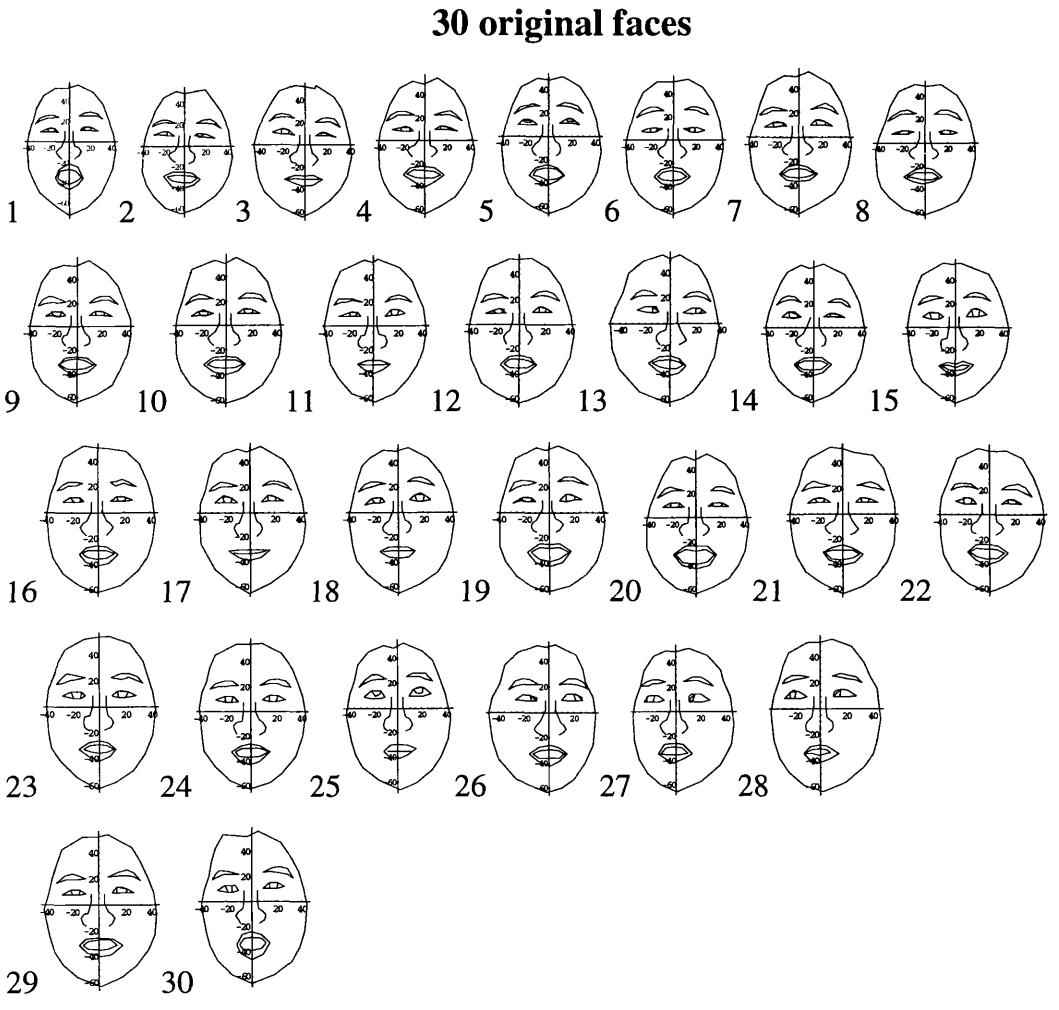


Figure 5.12. The thirty (original) aligned faces for Program 5.3.

We choose to use 98% of the variance (f_v = 0.98), and Equation (5.8) then says we should use only the first three modes (i.e. $t = 3$). When we bring out the range of facial

expressions generated by varying a selection of values $-\sqrt{\lambda_k} \leq b_k \leq \sqrt{\lambda_k}$ and $k = 1, 2, 3$. We choosed to let one face shape build 29 synthetic face shapes. There are 30 x 29 x 3 = 2,610 synthetic shapes. Figure 5.13 shows some of them.

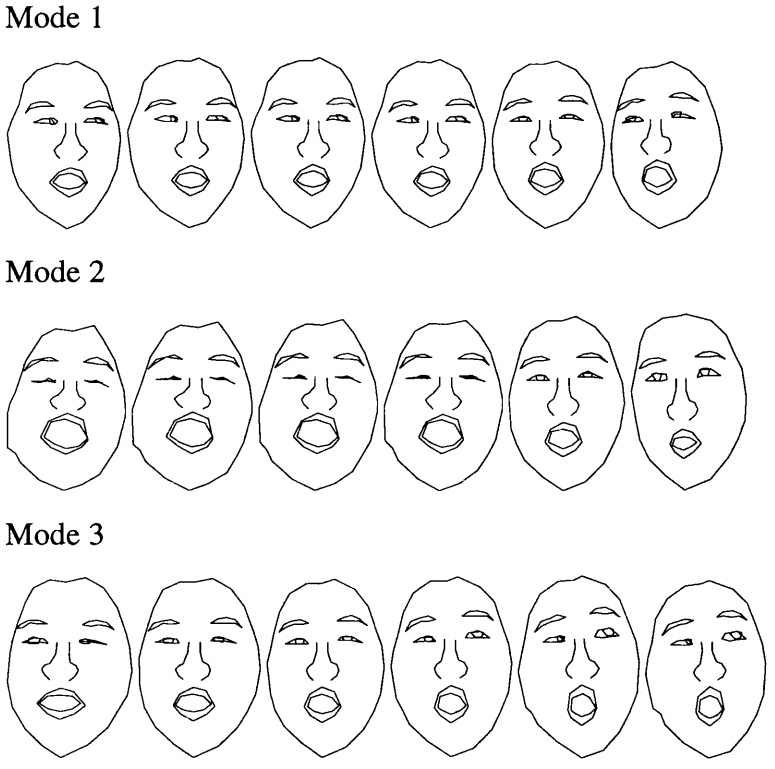


Figure 5.13 We show the first six varied shapes of each mode.

Conclusion If we consider images of the face, we shall see that each person has own features, for example eyes and mouth expression. The reason is because of a wide degree of variation which changes in expression, speaking. Our generated images change their expression continuously, as is born out by the fact that in a sequence of screen frames, they give a convincing animation

Part 3 Illustrative Projects

Chapter 6 LVQ, SOM, and human observation

6.1 Introduction

We use the synthetic shapes (both Hand and Face shapes from Chapter 5) to classify by two methods of artificial neural networks, Kohonen Self-Organizing Maps (SOM) and Learning Vector Quantizations (LVQ). Our aim is to codify the extent to which the various groupings correspond to human visual observation. We also make an attempt at a classification scheme based on what human vision might pick out. Comparing Kohonen and LVQ, we aim to discover which method gives the best classification.

Note that the Cases 1-5 will use the training examples from thirty generated hands (Figure 6.1), derived from the 5^{th}-7^{th} original shapes (Figure 5.4). We select these particular shapes because each original shape has its own special characteristics.

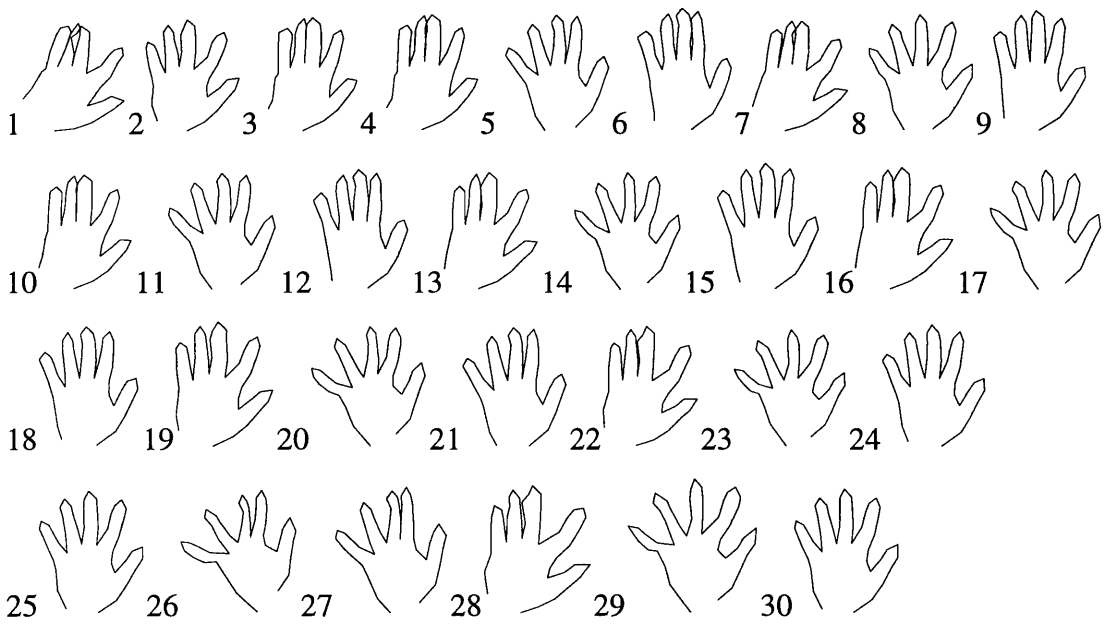


Figure 6.1. Thirty generated shapes are classified by using Cases 1-5. They express different attitudes and also keep the original characteristic of each shape.

6.2 Hand Shapes

In Case 1, we use LVQ to classify the 30 training examples. In Cases 2-5, we use the Kohonen SOM method. The difference amongst Cases 2 to 5 is the topological structure - No Topological Structure, a Linear Array, a Diamond Structure, and a Rectangular Neighbourhood. The aim is to know which structure gives the best result - match similar shapes to the same cluster unit, and the fewer groups, the better. Finally, we compare the result of LVQ with the result of SOM.

Table 6.1 Conditions.

The SOM Cases Cases 2-5 have these conditions in common:

1. The learning rate - initial learning rate $\alpha = 0.6$.

   - the learning rate is reduced by $\alpha = 0.6 - 0.003\,t$.

2. The initial weights are randomised between 0.1 and 0.9.

Case	Learning rate	Initial weights	Stopping Condition	Topology nbds.
1 (LVQ)	0.1 - t / 500	Hands 1-3	$\alpha < 0.02$	None
2 (SOM)	0.6 - 0.003*t	0 .1 to 0.9	$\alpha < 0.01$	$R = 0$
3 (SOM)	"	"	"	Linear $R = 2 - 1 - 0$
4 (SOM)	"	"	$\alpha < 0$	Diamond $R = 1 - 0$
5 (SOM)	"	"	$\alpha < 0$	Rectangular $R = 1 - 0$

Case 1 An LVQ Case.

We choose to have three clusters, using as reference vector the 5^{th} - 7^{th} original shapes of Figure 5.4 mentioned in Section 6.1 above.

Table 6.2 Reference vectors. Notation is given below.

3f-4f = The third finger closes the fourth finger.

s = All fingers are separated.

n = Normal Hand in the sense that the hand is posted in natural style.

C# = Cluster unit #.

Reference vectors	Shape 5	Shape 6	Shape 7
Description	3f-4f	s	n
Represent	C1	C2	C3

By using the LVQ method, the result is obtained as shown in Table 6.3.

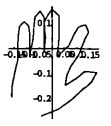
Table 6.3 LVQ with Hand Shapes.

No. of Cluster unit	*Shapes*
1	1,2,3,4,7,10,13,16,19,22,28
2	5,8,11,14,17,20,23,26,27,29
3	6,9,12,15,18,21,24,25,30

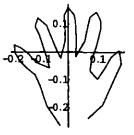
The cluster units of these shapes are shown below.

Figure 6.2 The LVQ classification of the selected 30 hands.

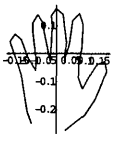
The reference vector for Cluster unit 1.



The reference vector for Cluster unit 2.

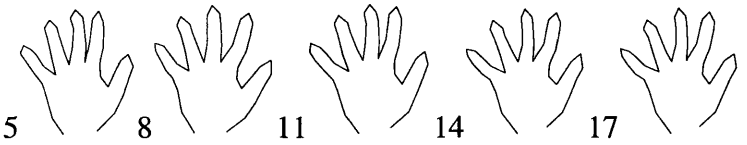


The reference vector for Cluster unit 3.

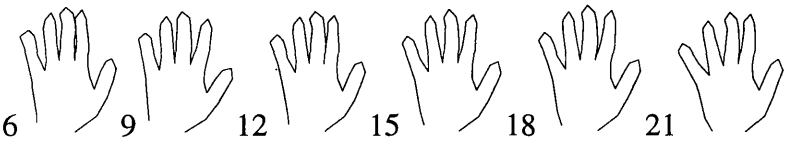


Cluster unit 1

Cluster unit 2



Cluster unit 3



Discussion

Cluster unit 1 We can characterise this cluster unit by saying the third and fourth fingers close together.

Cluster unit 2 Fingers are spread out. Some patterns' second finger is closest to the third. However the fourth and the fifth finger are still spread out.

Cluster unit 3 Fingers are spread out but noticeably less so than in Cluster unit 2. We might describe them as the normal hand, i.e. as placed on a table naturally. We can mention one or two exceptions (not too much).

The result from this LVQ method is shown in Table 6.7.

SOM CASES

Case 2 SOM with No Topological Structure.

We obtain the result as shown in Table 6.4.

Table 6.4 SOM (no topological structure) with 30 Hand Shapes.

No. of cluster unit	*Shapes*
1	1
2	2,3,4,7,9,10,13,16,19,22,28
3	5,6,8,11,12,14,15,17,18,20,21,23,24,25,26,27,29,30

The shapes are classified as shown below.

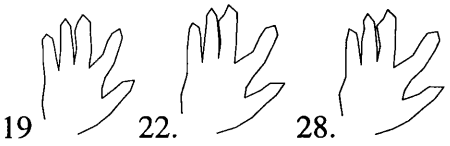
Figure 6.3 The Kohonen classification of the 30 selected hands ($R = 0$).

Cluster unit 1

1.

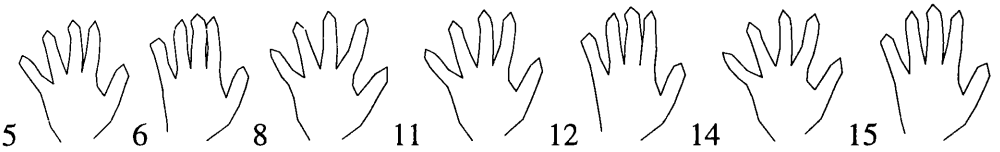
Cluster unit 2

2 3 4 7 9 10 13 16



19 22. 28.

Cluster unit 3



5 6 8 11 12 14 15

17 18 20 21 23 24

25 26 27 29 30.

Discussion

Cluster unit 1 The 1st shape is not a realistic finger so that it separated from others. The striking thing about this group is that it consists of a single *inhuman* hand. Thus the Kohonen method has worked excellently in consigning it to a class on its own. The other cases may stretch things a bit, but they do not possess the impossibility of the group 1 hand.

Cluster unit 2 These shapes' third finger is closest to the fourth, and the second finger is spread out clearly. The shape 2's fingers are a bit spread out. However, if we compare the closest distance, then we found that the third finger is closest to the fourth.

Cluster unit 3 We can observe that some patterns' second finger is closest to the third finger and not the same as patterns in Cluster unit 2. In more detail, the 6^{th}, 12^{th} are normal hands and their second finger is closest to the third. The 20^{th} pattern's fingers are spread out and the second finger is closest to the third finger. But a rough overall characterisation is that the fingers are spread out.

Case 3 SOM with Linear Array.

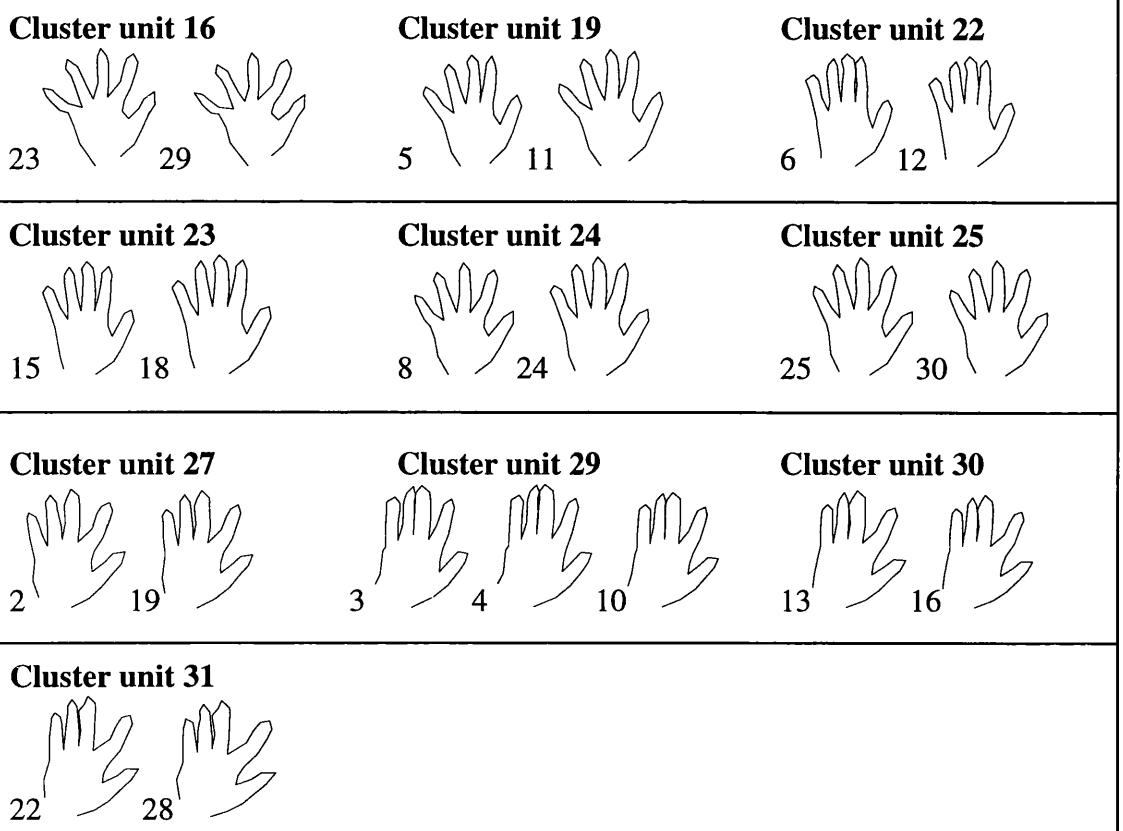
We start with $R = 2$, reducing to $R = 1$ when $90 < t < 170$ and $R = 0$ when $t \geq 170$. The result is shown in Table 6.5.

Table 6.5 SOM (linear array) with 30 Hand Shapes.

No. of cluster unit	*Shapes*
6	26
11	27
14	14
16	23,29
17	17
18	20
19	5,11
20	21
22	6,12
23	15,18
24	8,24
25	25,30
26	9
27	2,19
29	3,4,10
30	13,16
31	22,28
32	7
33	1

Some classified shapes from Table 6.5 are shown below.

Figure 6.4 Some classified hands by using SOM (linear array).



Discussion

The similar hand shapes classified by using SOM (Linear Array) are put to the same cluster unit. Hence, this experiment is *better* than No Topological Structure. However, each cluster unit has mostly two shapes except Cluster unit 29 has three shapes. The good result should be grouped more than two shapes in the same cluster.

Case 4 SOM with Diamond neighbourhoods.

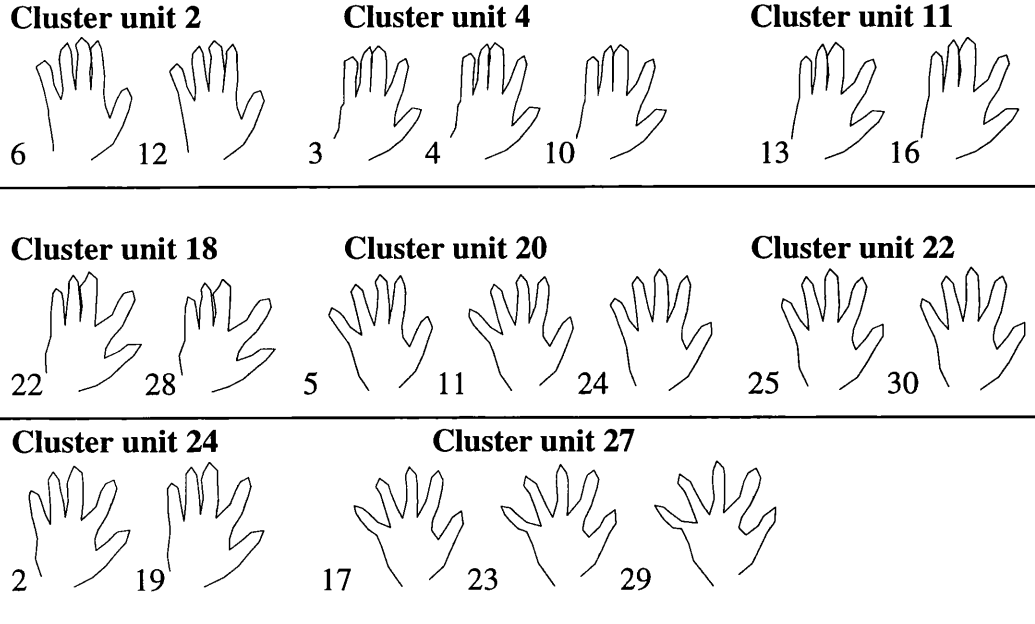
We start with $R = 1$, reducing to $R = 0$ when $t \geq 120$. The result is shown in Table 6.6.

Table 6.6 SOM (diamond neighbourhoods) with 30 Hand Shapes.

No. of cluster unit	*Shapes*
2	6,12
4	3,4,10
6	1
7	15
8	18
9	9
11	13,16
12	7
14	21
18	22,28
19	27
20	5,11
22	24,25,30
24	2,19
27	17,23,29
28	8
30	14
31	26
32	20

Some classified shapes from Table 6.6 are shown in Figure 6.5 below.

Figure 6.5 Some classified hands by using SOM (Diamond neighbourhoods).



Discussion

The similar hand shapes classified by using SOM (Diamond neighbourhoods) are put to the same cluster unit. Moreover, there are three cluster units, i.e. 4, 22, and 27, which each cluster unit has three similar shapes. Hence, this experiment is *better* than Linear Array.

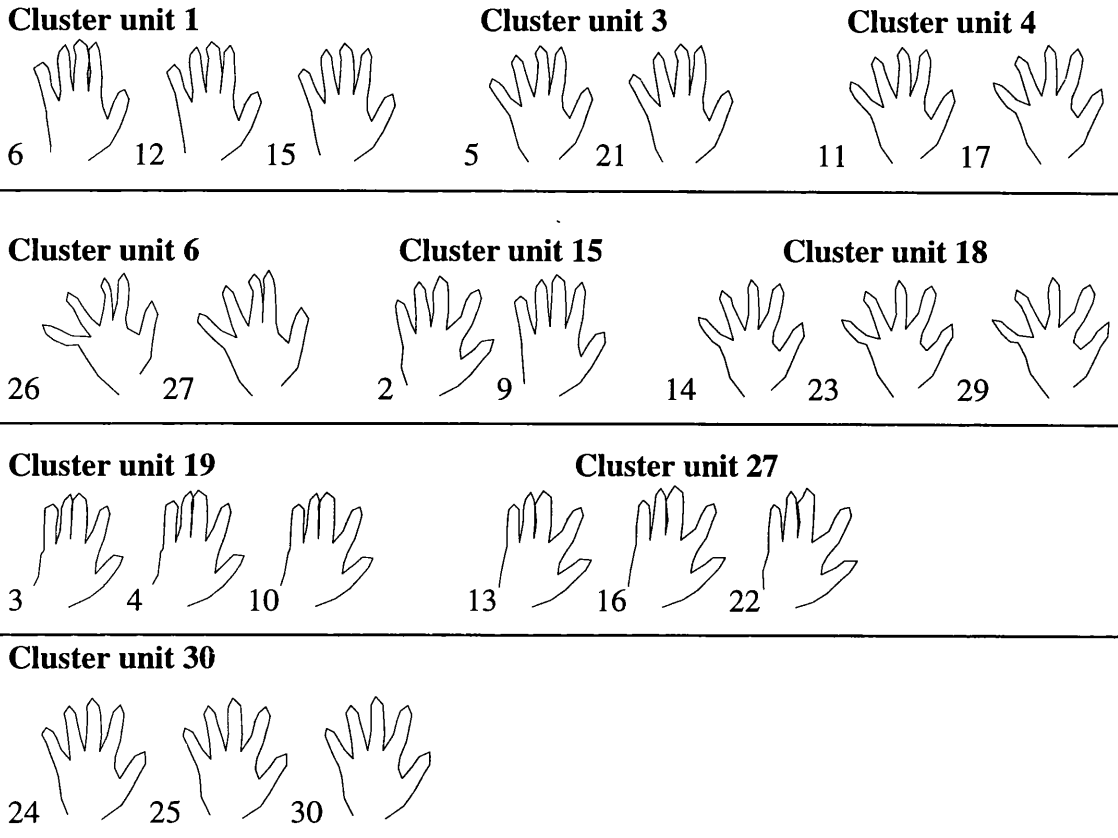
Case 5 SOM with Rectangular neighbourhoods.

We start with $R = 1$, reducing to $R = 0$ when $t \geq 120$. The result is shown in Table 6.7.

Table 6.7 SOM (rectangular neighbourhoods) with 30 Hand Shapes.

No. of cluster unit	*Shapes*
1	6,12,15
3	5,21
4	11,17
5	20
6	26,27
8	18
15	2,9
18	14,23,29
19	3,4,10
22	19
24	8
27	13,16,22
30	24,25,30
31	1
32	7
33	28

Figure 6.6 Some classified hands by using SOM (Rectangular neighbourhoods).



Discussion

By comparison, the *Diamond neighbourhoods* have 19 cluster units, but the *Rectangular neighbourhoods* has 16 cluster units. In addition, the similar hand shapes from Rectangular neighbourhoods are still put to the same cluster. Moreover, there are five cluster units which each cluster has three similar shapes. To conclude, the *best* structure of SOM in these experiments is the Rectangular neighbourhoods. Therefore, we select the SOM with Rectangular neighbourhoods to compare with the result LVQ (Table 6.8 below).

Discussion and comparison between LVQ and SOM (Rectangular nbds.)

CONCLUSION It seems that Kohonen self-organizing maps are better than Learning Vector Quantization because a Kohonen net can classify in more detail. The shapes from the same cluster unit are closest to each other.

We observe that there are dominant characteristics (description) with which we can classify hand shapes by ourselves. Table 6.8 below is the database underlying Table 6.9 and the final clearcut conclusions of Table 6.10.

Classification Scheme

C# = Cluster unit #.
s = The fingers are spread out.
n = Normal hand in the sense that the hand is posed in natural style.
2f-3f = The second finger is closest to the third finger.
3f-4f = The third finger is closest to the fourth finger.

Table 6.8 Summarize the results of a LVQ method and a SOM method [Rectangular neighbourhoods] with dominant hand description by human.

Hand shapes	1	2	3	4	5	6	7	8	9	10
Description	3f-4f	3f-4f	3f-4f	3f-4f	2f-3f	n	3f-4f	s	n	3f-4f
LVQ	C1	C1	C1	C1	C2	C3	C1	C2	C3	C1
SOM	C31	C15	C19	C19	C3	C1	C32	C24	C15	C19
Hand shapes	11	12	13	14	15	16	17	18	19	20
Description	s	n	3f-4f	s	n	3f-4f	s	n	3f-4f	s
LVQ	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2
SOM	C4	C1	C27	C18	C15	C27	C4	C8	C22	C5
Hand shapes	21	22	23	24	25	26	27	28	29	30
Description	2f-3f	3f-4f	s	s	s	2f-3f	2f-3f	3f-4f	s	s
LVQ	C3	C1	C2	C3	C3	C2	C2	C1	C2	C3
SOM	C3	C27	C18	C30	C30	C6	C6	C33	C18	C30

For example, Hand shapes 1-4 (Figure 6.2) are in Cluster unit 1 by LVQ, and each shape's third finger is closest to its fourth finger. By using Table 6.8, we make Table 6.9 below in order to clarify 4 distinct human classifications.

Table 6.9 Hand Classification by Human.

	3f-4f	s	n	2f-3f
Shapes	1,2,3,4,7,10,13, 16,19,22,28	8,11,14,17,20 23 ,24,25,29,30	6,9,12,15,18	5,21,26,27

we compare LVQ with SOM in order to find the best method. The good classification results depend on the following decision.

(I) (Most important) The number of different shapes should be small number.

(II) (Important) The percentage of error should be very small.

Note The percentage of error is calculated by using the following.

Percentage of error = (No. of different shapes/Total shape numbers) * 100

Table 6.10 The results of LVQ and SOM Comparison. The number of shapes different from the prevailing shape in their cluster. For example, Shape 5 '2f-3f' differs from Shape 8, 11, 14, etc. 's' in the same Cluster unit 2 (LVQ).

Method	No. of different shapes	Percentage of error
LVQ	7	23.33%
SOM	2	6.67%

Discussion between LVQ and SOM

(I) Comparison

LVQ method: an excellent result is that Cluster unit 1 (C1) can put similar shapes '3f-4f' to the same cluster unit correctly. However, there are 7 different shapes which are the most important impact. The good result should have small different shape number as mentioned. This number effects the high percentage of error to 23.33%.

SOM method: there are 2 different shapes. This number effects the percentage of error to 6.67%. Although there are small shape numbers in one cluster, these shapes close clearly together.

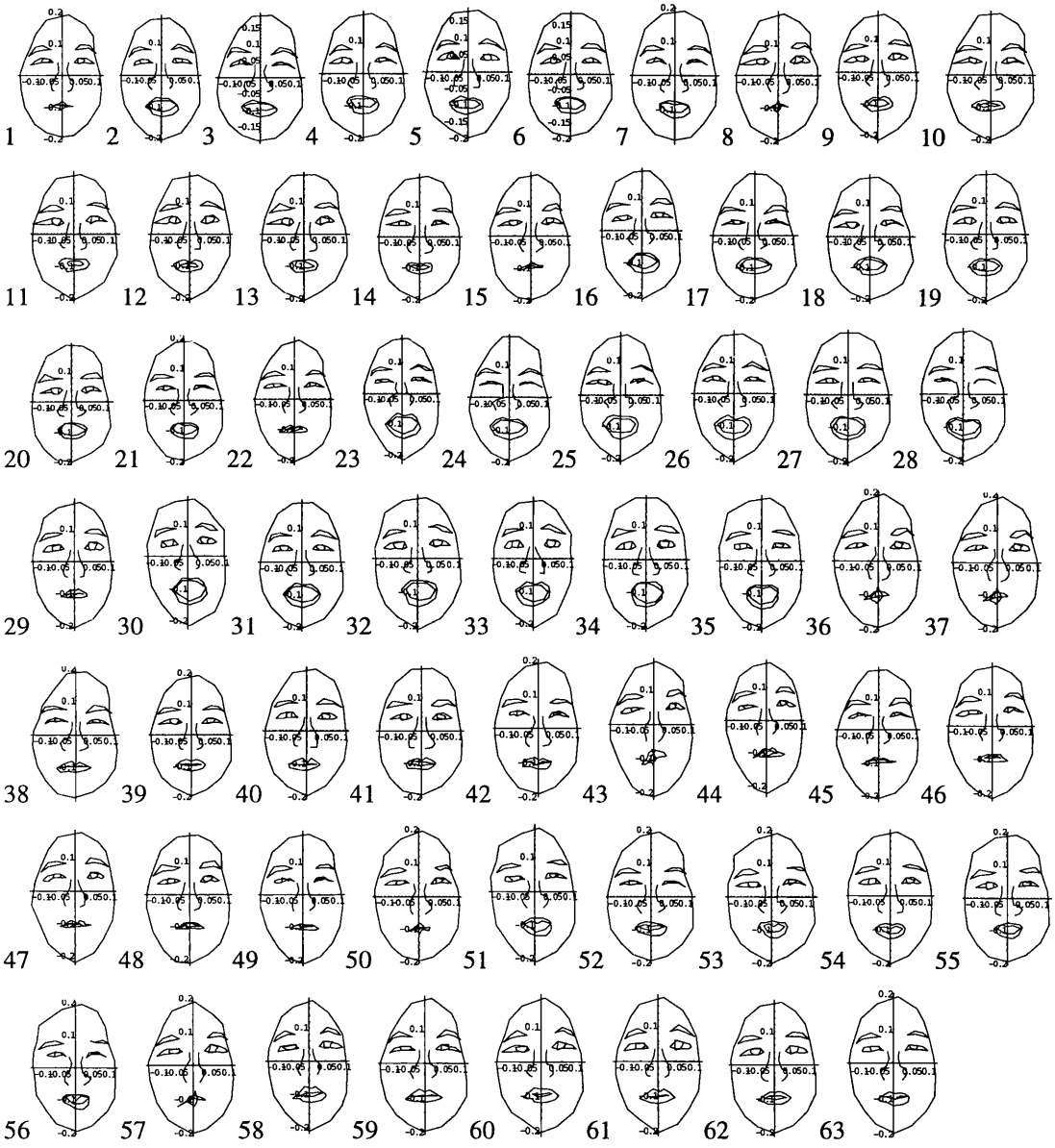
(II) Conclusion

This module again shows that a Kohonen net (SOM method) can classify more details than a LVQ net. In addition, the percentage of error is less than the LVQ method. Hence, SOM is more powerful than LVQ in mimicking human vision.

6.3 Face Shapes

We select some 63 from 504 synthesised images (Figure 6.7) to classify by Learning Vector Quantization (LVQ), and Kohonen Self-Organizing Maps (SOM).

Figure 6.7 The 63 faces for classification.



Selecting only sixty-three generating faces to classify by LVQ and Kohonen methods.

This face - generating module obtains some destroyed mouths implying we might have to pay more attention to annotation. In addition, it might give a better result

when we increase the number of original faces. Now 63 selected generating faces are classified by using a LVQ net in Case 6.

Table 6.11 Conditions for Cases 6-7.

Case	Learning rate	Initial weights	Stopping Condition	Topology nbds.
6 (LVQ)	0.1 - t / 500	-0.3 to 0.3	$\alpha < 0.01$	None
7 (SOM)	0.6 - 0.003 * t	0 .1 to 0.9	$\alpha < 0$	Rectangular R = 1- 0

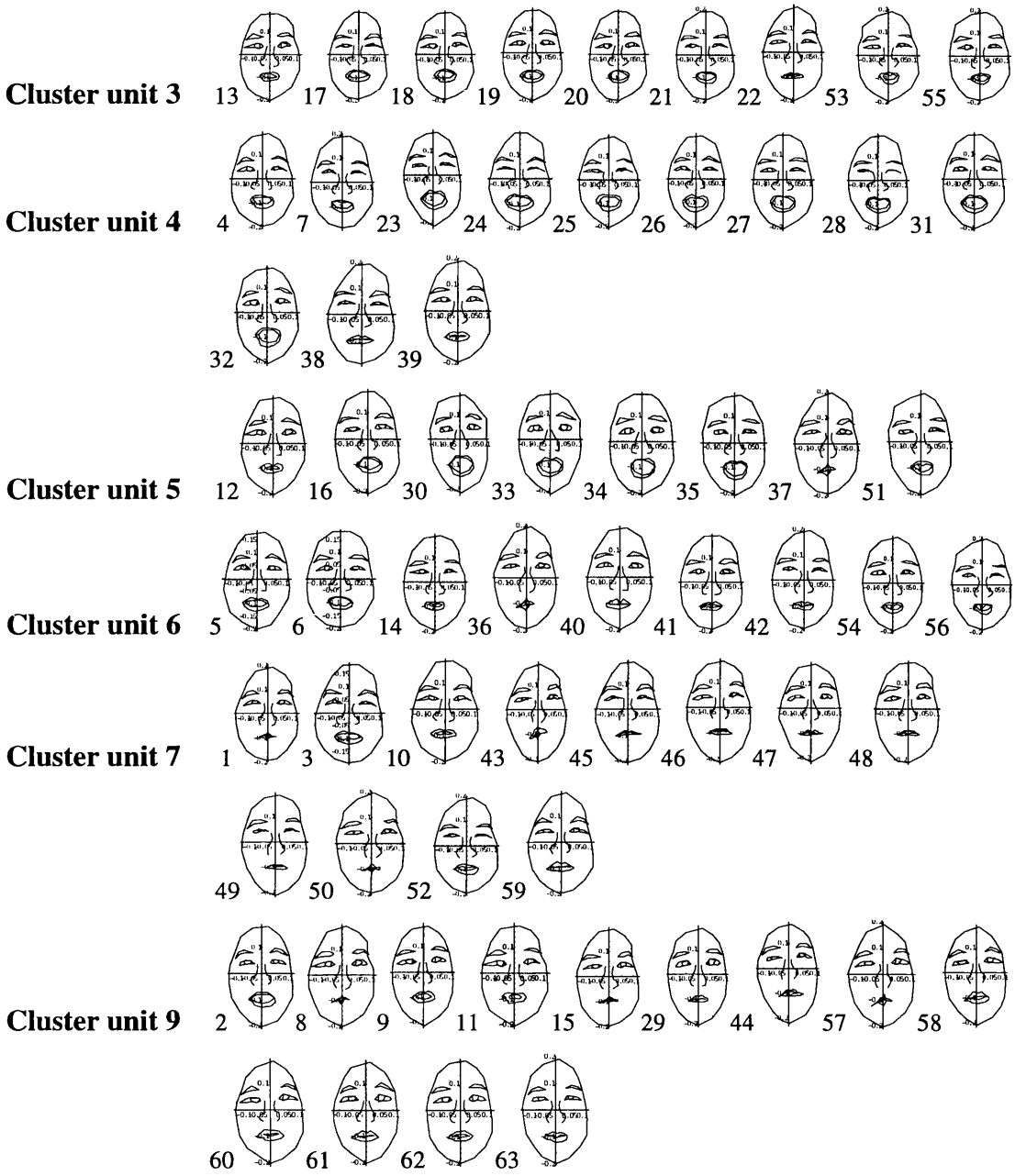
Case 6 LVQ.

There are 6 cluster units. The result is shown in Table 6.12.

Table 6.12 LVQ with 63 Face Shapes.

No. of Cluster unit	*Shapes*
3	13,17,18,19,20,21,22,53,55
4	4,7,23,24,25,26,27,28,31,32,38,39
5	12,16,30,33,34,35,37,51
6	5,6,14,36,40,41,42,54,56
7	1,3,10,43,45,46,47,48,49,50,52,59
9	2,8,9,11,15,29,44,57,58,60,61,62,63

Figure 6.8 The LVQ classification of the selected 63 Faces.



Discussion

Cluster unit 3 Each mouth is opened wide in common except the 22nd shape. The eyes are opened moderately.

Cluster unit 4 Most patterns' eyes are slightly opened except the 31st shape and the 32nd shape. Their mouth is opened very wide except the 38th shape and the 39th shape.

Cluster unit 5 All members' eyes are opened wider than the other groups. Their mount is also opened wide except the 12th shape. Notice that the 37th shape's mouth is destroyed.

Cluster unit 6 The mouth is opened widely except the 40th-42th shapes. One mouth is destroyed. The eyes are opened wide but less than members in Group 5.

Cluster unit 7 The mouth is closed some mouths are destroyed. The mouth from the 3rd shape is opened moderately.

Cluster unit 9 The mouth is closed or opened very slightly except the second pattern. Their eyes are opened wide. In more detail, the mouths from the 8th and the 57th shapes are destroyed.

Conclusion

The result from this LVQ application is not satisfactory because there are some members, from the same group, which are still different from most of them. This program is not able to characterise the especially evidence clearly.

Case 7 SOM with Rectangular neighbourhoods.

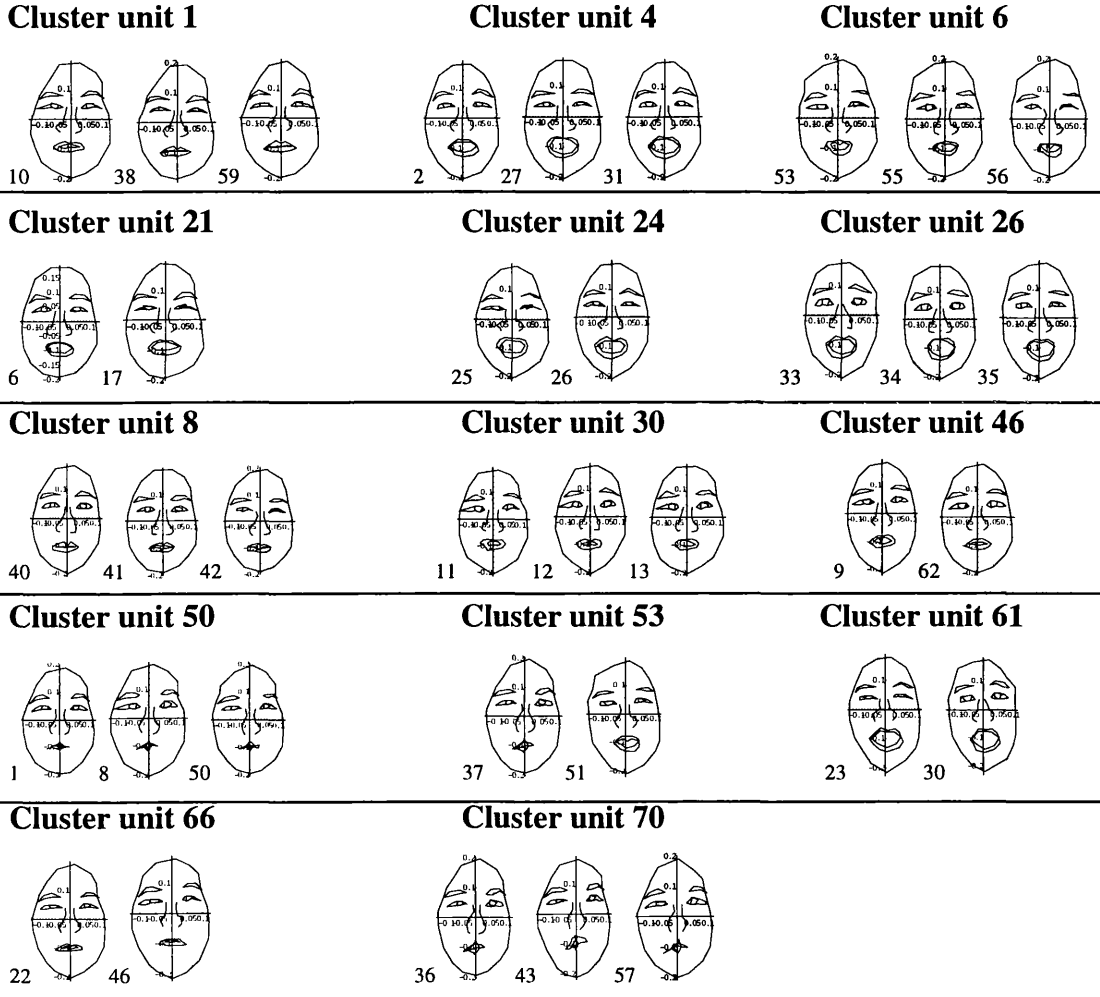
We start with $R = 1$, reducing the radius R to 0 when $t \geq 0$. The result is shown in Table 6.13.

Table 6.13 SOM (rectangular neighbourhoods) with 63 Face Shapes.

No. of Cluster unit	*Shapes*
1	10,38,59
2	52
3	7
4	4,5
6	2,27,31
7	19
8	39
10	53,55,56
12	3
19	54
20	14
21	6,17
23	24
24	25,26
25	32
26	33,34,35
28	40,41,42
29	15
30	11,12,13
31	20
33	28
41	18
42	21
44	63
45	61
46	9,62
47	29
48	47
50	1,8,50
52	16
53	37,51
58	48
61	23,30
63	44
64	58
65	60
66	22,46
67	49
68	45
70	36,43,57

We show a selection to illustrate this because there are many groups and each group has at most three members. All patterns are shown in Figure 6.9. For example, Cluster 26 has Shapes 33rd - 35th so that:

Figure 6.9 Some classified faces by using SOM (Rectangular neighbourhoods).



Discussion

Most of these shapes are put to the right place. Each cluster has the similarly expressed face shapes. For example, three shapes in Cluster unit 1 have the slightly opened eyes and closed mouths. In addition, their eyebrows and face structure close each other. However, there are five different shapes (27, 17, 42, 37, 23) put in the wrong cluster unit. For example, Shape 37 has the destroyed mouth rather than the opened mouth.

Discussion and Comparison between LVQ and SOM

Table 6.14 below is the database underlying Table 6.15 and the final clearcut conclusions of Table 6.16.

Classification scheme

C# = Cluster unit #.
mcs = The mouth is between closed and opened slightly.
mw = The mouth is opened.
ms = The mouth is opened slightly.
mm = The mouth is opened moderately.
md = The mouth is destroyed.
mso = The mouth has the "o" shape.
ew = The eyes are opened wide.
es = The eyes are opened slightly.
em = The eyes are opened moderately.
esm = The eyes are opened between slightly and moderately.

Table 6.14 Database for Comparing a LVQ method with a KOHONEN method [Rectangular Neighbourhoods].

Patterns	1	2	3	4	5	6	7	8	9	10	11
Description	em,md	em,mw	es,mm	es,mm	ew,mw	ew,mw	es,mm	em,md	ew,mm	em,mcs	ew,mso
LVQ	C7	C9	C7	C4	C6	C6	C4	C9	C9	C7	C9
KOHONEN	C50	C6	C12	C4	C4	C21	C3	C50	C46	C1	C30

Patterns	12	13	14	15	16	17	18	19	20	21	22
Description	ew,mso	ew,mso	ew,mso	em,md	em,mw	es,mm	esm,mw	ew,mw	esm,mw	es,mw	es,mcs
LVQ	C5	C3	C6	C9	C5	C3	C3	C3	C3	C3	C3
KOHONEN	C30	C30	C20	C29	C52	C21	C41	C7	C31	C42	C66

Patterns	23	24	25	26	27	28	29	30	31	32	33
Description	es,mw	es,mw	es,mw	em,mw	es,mw	es,mw	ew,mcs	em,mw	em,mw	ew,mw	em,mw
LVQ	C4	C4	C4	C4	C4	C4	C9	C5	C4	C4	C5
KOHONEN	C61	C23	C24	C24	C6	C33	C47	C61	C6	C25	C26

Patterns	34	35	36	37	38	39	40	41	42	43	44
Description	ew,mw	em,mw	em,md	em,md	em,mcs	em,mcs	ew,mcs	ew,mcs	em,mcs	em,md	em,mcs
LVQ	C5	C5	C6	C5	C4	C4	C6	C6	C6	C7	C9
KOHONEN	C26	C26	C70	C53	C1	C8	C28	C28	C28	C70	C63

Patterns	45	46	47	48	49	50	51	52	53	54	55
Description	em,mcs	em,mcs	em,mcs	ew,mcs	em,mcs	em,md	esm,mw	es,mm	esm,mw	em,mw	esm,mw
LVQ	C7	C7	C7	C7	C7	C7	C5	C7	C3	C6	C3
KOHONEN	C68	C66	C48	C58	C67	C50	C53	C2	C10	C19	C10

Patterns	56	57	58	59	60	61	62	63
Description	ecm,mw	ew,md	em,mm	em,mcs	ew,mm	ew,mm	ew,mcs	em,mcs
LVQ	C6	C9	C9	C7	C9	C9	C9	C9
KOHONEN	C10	C70	C64	C1	C65	C45	C46	C44

Table 6.15 Face Classifications by Human.

	Group 1	Group 2	Group 3	Group 4	Group 5
Description	mcs,em	mcs,ew	mm,em	mw,ew	mcs,em
Shapes	22,44,46, 47,48,49 10,38,45,59 38,42,63	29,40,41,61	9,58,60,62	30,32,33,34,35	18,20,51,53 55,56
	Group 6	Group 7	Group 8	Group 9	Group 10
Description	mm,es	mw,em	mw,es	mso,em	md,em
Shapes	4,3,7,17,52	2,5,6,16, 19,31,54	21,23,24,25 26,27,28	11,12,13,14	1,8,15,36,37 43,50,57

Table 6.16 The results of LVQ and SOM Comparison. The number of shapes different from the prevailing shape in their cluster.

Method	No. of different shapes	Percentage of error
LVQ	35	55.56%
SOM	5	7.94%

(I) Comparison

LVQ method: There are 35 different shapes so that the percentage of error is high.

Kohonen method: There are 5 different shapes which give the percentage of error less than the percentage of error of SOM method.

(II) Conclusion

This Kohonen application is excellent because it is able to characterise each pattern clearly. Each member in the same group is very close to each other and they also have similar expression.

Final Conclusion

(compare a Kohonen application with a LVQ application)

A Kohonen net can give the result more precisely than a LVQ net. To summarize, the Kohonen method is a better human than the LVQ method.

6.4 Main Project (117 Face shapes)

In Chapter 5, we built 2,610 generating faces. We now select 117 generated face shapes (Figure 6.10 below), selected from the first 13 original shapes and each shape is varied by the first 3 modes, to classify them by using Learning Vector Quantization (LVQ) in Case 6, and Kohonen Self-Organizing Maps in Case 7.

Figure 6.10 The 117 faces to classify.

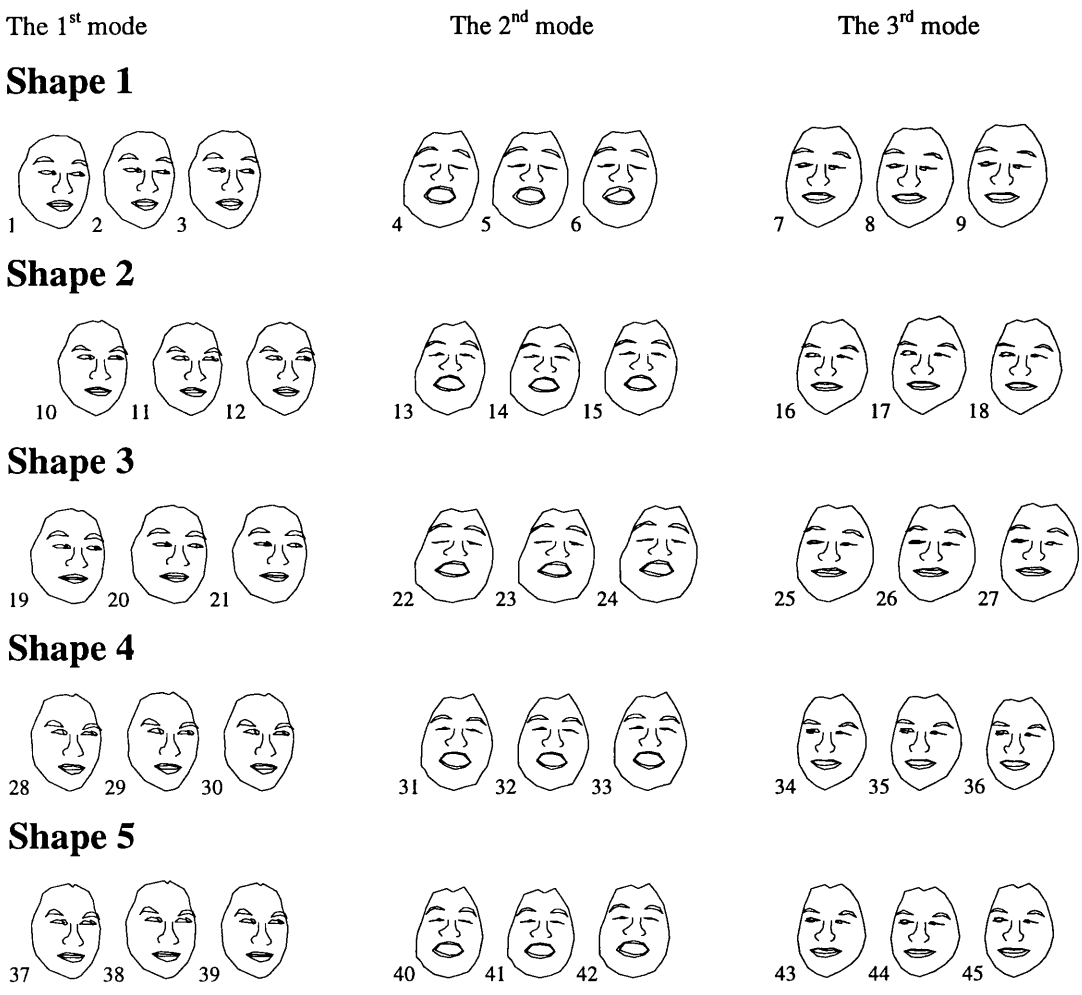
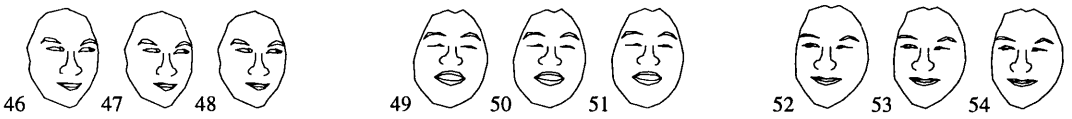
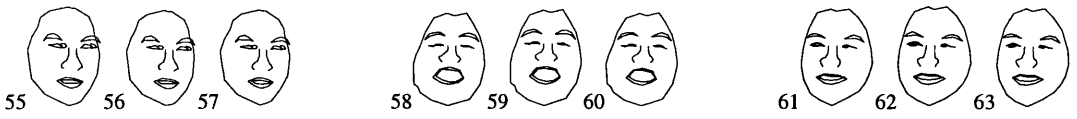


Figure 6.10 (Continued)

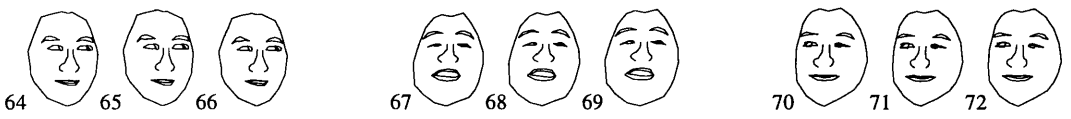
Shape 6



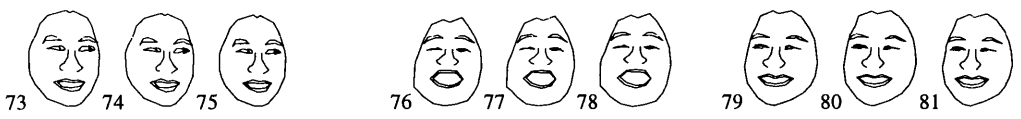
Shape 7



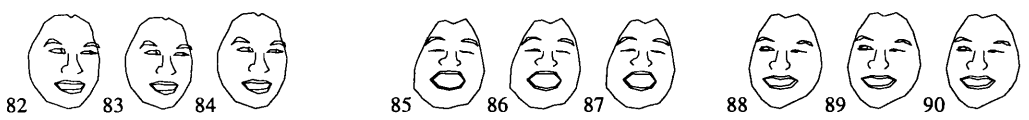
Shape 8



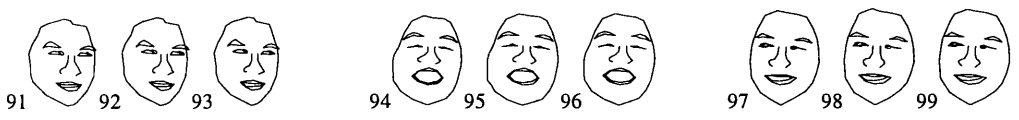
Shape 9



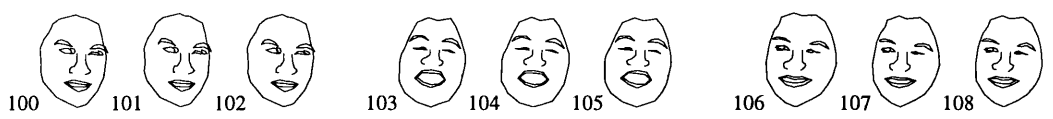
Shape 10



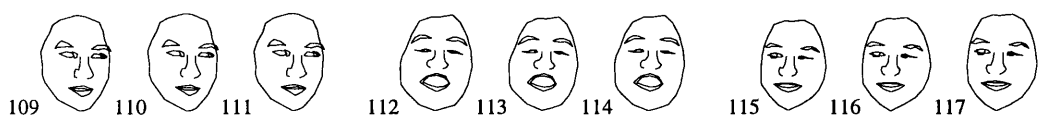
Shape 11



Shape 12



Shape 13



Generating face images (from 30 original faces).

In Figure 6.10, the faces from the same shape and the diagonal element are very close to each other. For example, eyes of one face look in the same direction. If the eyes are

closed, then the eyes of the other are closed, or vice versa. If one mouth is opened wide, then the mouth of the other, from the same shape, is open wide as well.

Table 6.17 Conditions for Cases 8-9.

Case	Learning rate	Initial weights	Stopping Condition	Topology nbhds.
8 (LVQ)	0.1 - t / 500	-0.3 to 0.3	$\alpha < 0.01$	None
9 (SOM)	0.7 - 0.00233*t	0 .1 to 0.9	$\alpha < 0.001$	Rectangular R = 1- 0

Case 8 LVQ with 117 FACE shapes.

There are three cluster units. The result is shown in Table 6.18.

Table 6.18 LVQ with 117 Face Shapes.

No. of Cluster unit	*Shapes*
1	1,2,3,10,11,12,19,20,21,28,29,30,37,38,39,46,47,48,55,56,57, 64,65,66,73,74,75,82,83,84,91,92,93,100,101,102,109,110,111
2	4,5,6,13,14,15,22,23,24,31,32,33,40,41,42,49,50,51,58,59,60, 67,68,69,76,77,78,85,86,87,94,95,96,103,104,105,112,113,114
3	7,8,9,16,17,18,25,26,27,34,35,36,43,44,45,52,53,54,61,62,63, 70,71,72,79,80,81,88,89,90,97,98,99,106,107,108,115,116,117

Figure 6.11 The LVQ Classification of the 117 Faces.

Cluster unit 1

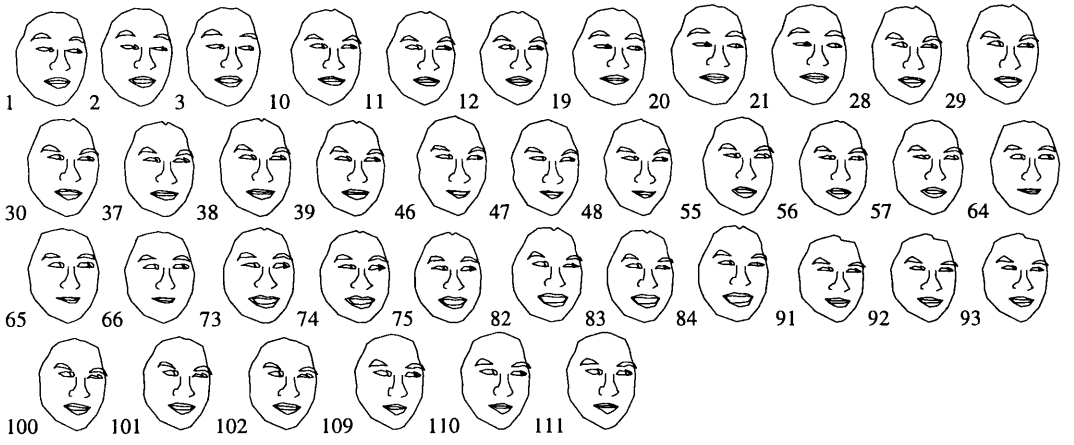
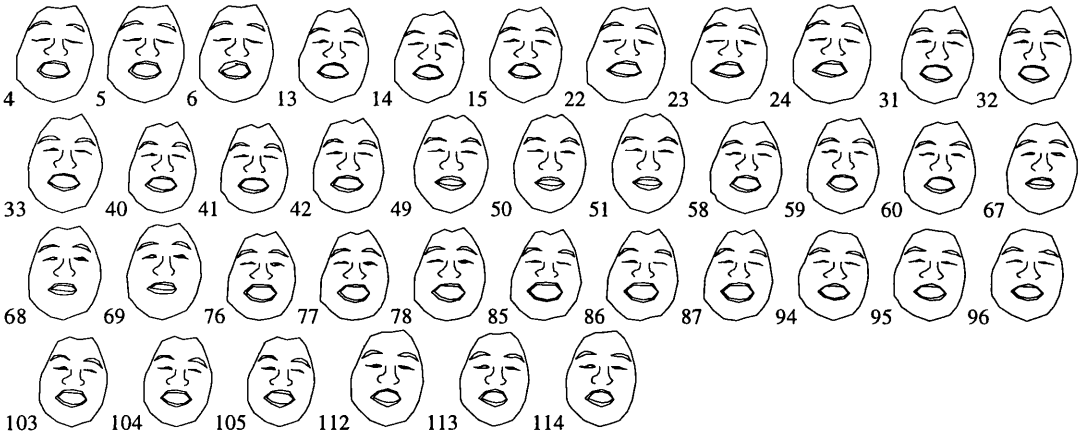
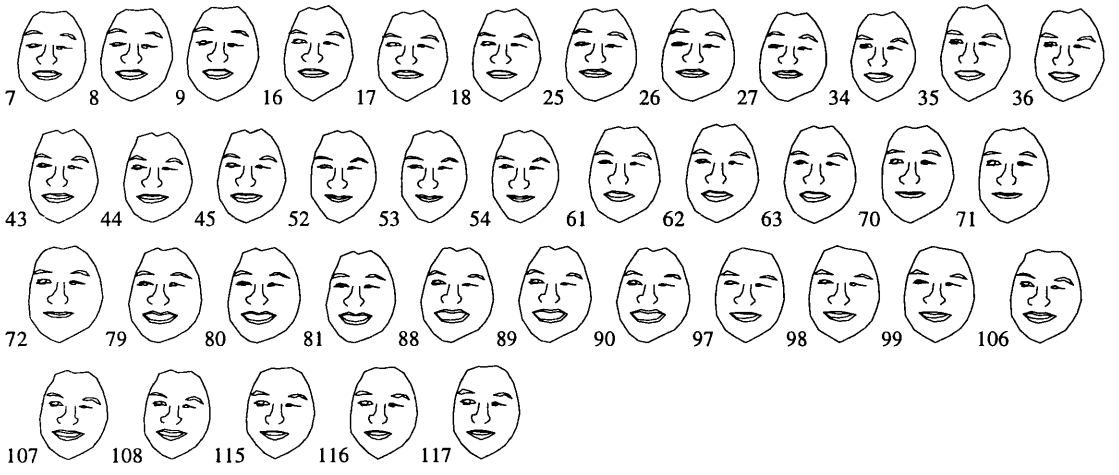


Figure 6.11 (Continued)

Cluster unit 2



Cluster unit 3



These are classifications by using the LVQ method.

Discussion

Cluster unit 1 The eyes look right hand side. The mouth is opened moderately. Some patterns' mouth is opened slightly but some mouths of them are opened widely.

Cluster unit 2 The eyes are almost closed. The mouth is opened very widely. In more detail, some patterns' eyes look down, but some eyes of them look up.

Cluster unit 3 Each pattern has a similar pattern. We show a selection to illustrate this. The person's mouth is opened moderately and the right eye is winking. The eyes are looked left hand side.

Conclusion

This classifying program, LVQ, gives good results because the members of each group have a very similar shape in facial expression. Although, for example, Shapes 1-3 are a bit different from Shapes 5-7. All members look similar.

Case 9 Kohonen SOM with Rectangular neighbourhoods.

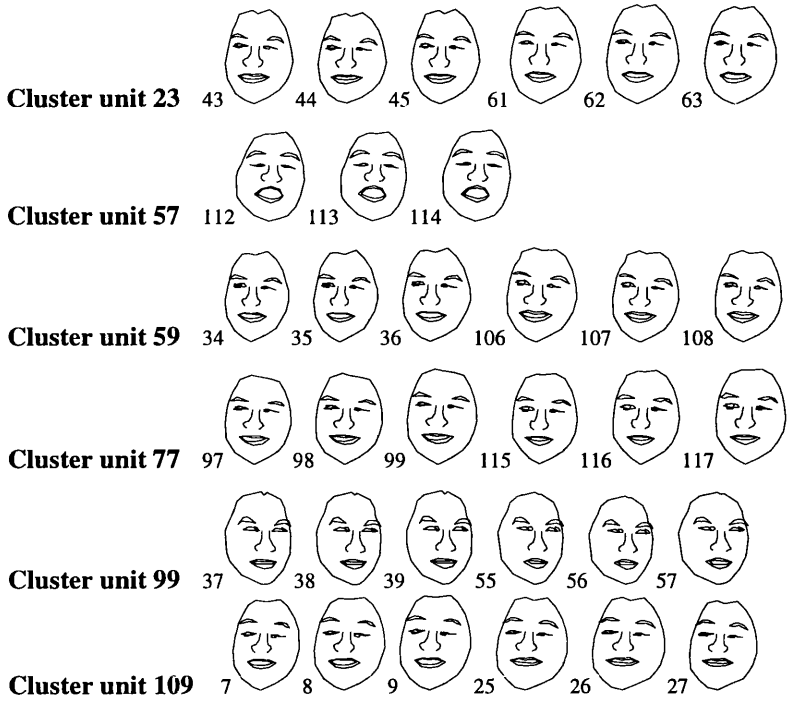
The reason why we reduce the learning rate α by $\alpha(t) = 0.7 - 0.00233*t$ is we wish to reduce it very slowly over many epochs - there are 300 epochs. We start with $R = 1$, reducing to $R = 0$ when $t \geq 180$. The result is shown in Table 6.19.

Table 6.19 SOM (rectangular neighbourhoods) with 117 Face Shapes.

No. of cluster unit	*Shapes*
1	67,68,69
3	103,104,105
5	31,32,33
7	76,77,78
9	13,14,15
19	58,59,60
21	40,41,42
23	43,44,45,61,62,63
25	4,5,6
27	22,23,24
37	85,86,87
39	94,95,96
41	88,89,90
43	79
44	80
45	81
55	49,50,51
57	112,113,114
59	34,35,36,106,107,108
61	91,92,93
63	82,83,84
73	70,71,72
75	52,53,54
77	97,98,99,115,116,117
79	109,110,111
81	73,74,75
91	16,17,18
93	64,65,66
95	46,47,48
97	10,11,12
99	37,38,39,55,56,57
109	7,8,9,25,26,27
111	19,20,21
113	1,2,3
115	28,29,30
117	100,101,102

Some examples are shown below.

Figure 6.12 Some cluster examples of the Kohonen Classification of 117 Faces.



The members of cluster units 23, 57, 59, 77, 99, and 109.

Discussion

For example, Cluster unit 57 has shapes 112, 113, and 114 so that it has been shown in Figure 6.11. The mouth of each pattern is opened widely. The eyes are opened moderately and are focused in a downward direction. More examples, Cluster unit 77 has Shapes 97, 98, 99, 115, 116, and 117. The right eye of each shape is winking and looking left hand side. The mouth is opened moderately. Cluster unit 99 has Shapes 34, 35, 36, 106, 107, and 108 as shown in the same figure below. The mouth of each pattern is opened moderately but it is wider than each shape in Cluster unit 77. In addition, each pattern in Cluster unit 99 is not the same as each pattern in Cluster unit 77 because the top of face is more rough jagged. It does work in many places except Shapes 79, 80, and 81 because they are not in the same group, but these shapes are very similar to each other (see Figure 6.13).

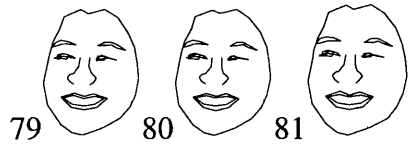


Figure 6.13 Shapes 79, 80, and 81 close together, but they are classified, by SOM, to different Cluster units.

Conclusion

This Kohonen method classified each pattern in depth details. The members of a cluster are very close to each other. For example, members in Cluster unit 99, have

their mouths opened wider than members in Cluster 77, and the top of face is more rough and jagged than those of members in Cluster 77.

Discussion and Comparison between LVQ and SOM

Table 6.20 below is the database underlying Table 6.21 and the final clearcut conclusions of Table 6.22.

Classification Scheme

es = Eyes straight
er = Eyes right.
el = Eyes left.
eu = Eyes up.
ed = Eyes down.

eo = Eyes is opened.
ec = Eyes is closed.
wr = Right eye is winking.

ms = Mouth is opened slightly.
mw = Mouth is opened wide.
mm = Mouth is opened moderately.
mc = Mouth is closed.

Table 6.20 Database for comparing LVQ with Kohonen (rectangular neighbourhoods).

Patterns	1	2	3	4	5	6	7	8	9	10	11
Desciption	er,ms	er,ms	er,ms	ed,mw	ed,mw	ed,mw	el,wr,mm	el,wr,mm	el,wr,mm	er,ms	er,ms
LVQ	G1	G1	G1	G2	G2	G2	G3	G3	G3	G1	G1
KOHONEN	G113	G113	G113	G25	G25	G25	G109	G109	G109	G97	G97

Patterns	12	13	14	15	16	17	18	19	20	21	22
Desciption	er,ms	ec,mw	ec,mw	ec,mw	el,wr,mm	el,wr,mm	el,wr,mm	er,ms	er,ms	er,ms	ed,mw
LVQ	G1	G2	G2	G2	G3	G3	G3	G1	G1	G1	G2
KOHONEN	G97	G9	G9	G9	G91	G91	G91	G111	G111	G111	G27

Patterns	23	24	25	26	27	28	29	30	31	32	33
Desciption	ed,mw	ed,mw	el,wr,mm	el,wr,mm	el,wr,mm	er,ms	er,ms	er,ms	ec,mw	ec,mw	ec,mw
LVQ	G2	G2	G3	G3	G3	G1	G1	G1	G2	G2	G2
KOHONEN	G27	G27	G109	G109	G109	G115	G115	G115	G5	G5	G5

Patterns	34	35	36	37	38	39	40	41	42	43	44
Desciption	el,wr,mm	el,wr,mm	el,wr,mm	er,ms	er,ms	er,ms	ec,mw	ec,mw	ec,mw	el,wr,mm	el,wr,mm
LVQ	G3	G3	G3	G1	G1	G1	G2	G2	G2	G3	G3
KOHONEN	G59	G59	G59	G99	G99	G99	G21	G21	G21	G23	G23

Patterns	45	46	47	48	49	50	51	52	53	54	55
Desciption	el,wr,mm	er,ms	er,ms	er,ms	ec,mw	ec,mw	ec,mw	el,wr,mc	el,wr,mc	el,wr,mc	er,ms
LVQ	G3	G1	G1	G1	G2	G2	G2	G3	G3	G3	G1
KOHONEN	G23	G95	G95	G95	G55	G55	G55	G75	G75	G75	G99

Patterns	56	57	58	59	60	61	62	63	64	65	66
Desciption	er,ms	er,ms	ec,mw	ec,mw	ec,mw	el,wr,mm	el,wr,mm	el,wr,mm	er,ms	er,ms	er,ms
LVQ	G1	G1	G2	G2	G2	G3	G3	G3	G1	G1	G1
KOHONEN	G99	G99	G19	G19	G19	G23	G23	G23	G93	G93	G93

Patterns	67	68	69	70	71	72	73	74	75	76	77
Desciption	eu,mm	eu,mm	eu,mm	el,wr,mc	el,wr,mc	el,wr,mc	er,mw	er,mw	er,mw	ec,mw	ec,mw
LVQ	G2	G2	G2	G3	G3	G3	G1	G1	G1	G2	G2
KOHONEN	G1	G1	G1	G73	G73	G73	G81	G81	G81	G7	G7

Patterns	78	79	80	81	82	82	84	85	86	87	88
Desciption	ec,mw	el,wr,mw	el,wr,mw	el,wr,mw	er,mw	er,mw	er,mw	ec,mw	ec,mw	ec,mw	el,wr,mw
LVQ	G2	G3	G3	G3	G1	G1	G1	G2	G2	G2	G3
KOHONEN	G7	G43	G44	G45	G63	G63	G63	G37	G37	G37	G41

Patterns	89	90	91	92	93	94	95	96	97	98	99
Desciption	el,wr,mw	el,wr,mw	er,ms	er,ms	er,ms	ec,mw	ec,mw	ec,mw	el,wr,mm	el,wr,mm	el,wr,mm
LVQ	G3	G3	G1	G1	G1	G2	G2	G2	G3	G3	G3
KOHONEN	G41	G41	G61	G61	G61	G39	G39	G39	G77	G77	G77

Patterns	100	101	102	103	104	105	106	107	108	109	110
Desciption	er,ms	er,ms	er,ms	ec,mw	ec,mw	ec,mw	el,wr,mm	el,wr,mm	el,wr,mm	er,ms	er,ms
LVQ	G1	G1	G1	G2	G2	G2	G3	G3	G3	G1	G1
KOHONEN	G117	G117	G117	G3	G3	G3	G59	G59	G59	G79	G79

Patterns	111	112	113	114	115	116	117
Desciption	er,ms	es,mw	es,mw	es,mw	el,wr,mm	el,wr,mm	el,wr,mm
LVQ	G1	G2	G2	G2	G3	G3	G3
KOHONEN	G79	G57	G57	G57	G77	G77	G77

Discussion between LVQ and SOM

(I) Comparison

Table 6.21 below shows a classification obtained by human observation. Then Table 6.22 compares the agreement of LVQ and SOM with this.

Table 6.21 Shape Classification by Human.

	Group 1 el,wr,mw	Group 2 el,wr,mc	Group 3 el,wr,mm	Group 4 eu,mm	Group 5 es,mw
Shape	79,80,81, 88,89,90	52,53,54, 70,71,72	7,8,9,16,17, 18,25,26,27, 34,35,36,43, 44,45,61,62, 63,97,98,99, 106,107,108, 115,116,117	67,68,69	112,113,114
	Group 6 ed,mw	Group 7 ec,mw	Group 8 er,ms	Group 9 er,mw	
Shape	4,5,6,22, 23,24	13,14,15,31, 32,33,40,41, 42,49,50,51, 58,59,60,76, 77,78,85,86, 87,94,95,96, 103,104,105	1,2,3,10,11,12, 19,20,21,28,29, 30,37,38,39,46, 47,48,55,56,57, 64,65,66,91,92, 93,100,101,102, 109,110,111	72,73,74, 82,83,84	

Table 6.22 Comparison between LVQ and SOM.

Method	No. of different shapes	Percentage of error
LVQ	30	25.64%
SOM	0	0.00%

(II) Conclusion

Table 6.22 comfirms strongly that for our (limited) data a Kohonen is better than a LVQ in that it classifies shapes more in line with human observation.

Broad conclusion for hands and faces

If the generating patterns are very similar, 30 faces program, then an LVQ method is able to characterise human intuition well for our data, but a Kohonen method can characterise in depth. Otherwise, if the generating patterns are very different, then a Kohonen method is able to characterise much better than the LVQ method.

Appendix

Mathematica

Chapter 2

Example 2.5 (page 23)

```
spc[x_,t_]:=Module[{w,b,α,θ,i,j,c,done,y,yin},w={0,0,0};θ=0.2;α=1;
                  Print["********************"];Print["   Net        Out      w"];
                  Print["********************"];done=False;
                  For[j=1,done==False,j++,Print["Epoch ",j];done=True;
                    Do[yin=w.x[[i]];If[yin>θ,y=1,If[yin<-θ,y=-1,y=0]];If[y!=t[[i]],
                      w=w+α t[[i]] x[[i]];done=False];Print["  ",yin,"  ",y,"   ",w],{i,4}]]];
x={{1,1,1},{1,0,1},{0,1,1},{0,0,1}};t={1,-1,-1,-1};
spc=[x,t];
```

Example 2.6 (page 26)

Mathematica Module

```
spc[x_,t_]:=Module[{w,b,α,θ,i,j,c,done,y,yin},w=Table[0,{i,Length[x]}];θ=0.1;α=1;
                  Print["********************"];Print["   Net        Out      w"];
                  Print["********************"];done=False;
                  For[j=1,done==False,j++,Print["Epoch ",j];done=True;
                    Do[yin=w.x[[i]];If[yin>θ,y=1,If[yin<-θ,y=-1,y=0]];If[y!=t[[i]],
                          w=w+α t[[i]] x[[i]];done=False];
                          Print["  ",yin,"  ",y,"   ",w],{i,Length[x]}]]];
x={{1,1,1,1},{1,1,0,1},{1,0,1,1},{0,1,1,1}};t={1,-1,-1,-1};spc=[x,t];
```

Example 2.7 (page 26)

```
spc[x_,t_]:=Module[{w,b,α, θ,i,j,done,yin,y},w=Table[0,{i,Length[x[[1]]]}];
                  θ=0.1;α=1;done=False;
                  For[j=1,done==False,j++, done=True;
                    Do[yin=w.x[[i]];yin>θ,y=1,If[yin<- θ,y=-1,y=0]];
                      If[y!=t[[i]],w=w+α t[[i]] x[[i]];done=False],
                    {i,Length[x]}]];Return[w]] ;

test[w_,ch_]:=Module[{i,θ},θ=0.1;Do[Print["Character ",i];
                   If[w[[i]].ch> θ,Print["Yes"],If[w[[i]].ch<-θ;Print["No"],
                     Print["Unsure"]]],{i,Length[w]}]];

perc[x_,t_]:=Module[{w,i,u},w=Table[0,{7},{64}];
                  Do[u=Transpose[t][[i]];w[[i]]=spc[x,u],{i,7}];Do[test[w,ch[[i]]],{i,Length[ch]}];
x={{1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,1,-1,
    1,1,1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,1,1,-1,-1,-1,
    1,-1,-1,-1,1},{-1,-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,1,-1,
    1,1,1,1,1,-1,1,1,-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,1,-1,1,1,
    1,1,-1,-1,-1,-1,-1,-1,-1,-1,1,1,1},{1,1,-1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,1,1,-1,-1,1,1,
    1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,-1,1,1,1,1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,1,1,1,1,-1,1,1,1,1,-1,
    1,1,-1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,1,-1,
    1,1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,1,-1,-1,-1,-1,-1,1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,-1,-1,
    1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,-1,
    1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,1},{1,1,
    1,-1,-1,-1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,-1,
    1,1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,-1,-1,1,1,1,
```

```
    1},{-1,-1,-1,1,1,1,-1,-1,1,-1,1,1,-1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,1,1,1,1,
    1,-1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,1,1,1,1,-1,
    1,-1,-1,-1,1,1,1,-1,-1,1}};
t={{1,-1,-1,-1,-1,-1,-1},{-1,1,-1,-1,-1,-1,-1},{-1,-1,
    1,-1,-1,-1,-1},{-1,-1,-1,1,-1,-1,-1},{-1,-1,-1,-1,
    1,-1,-1},{-1,-1,-1,-1,-1,1,-1},{-1,-1,-1,-1,-1,-1,1}};
ch = {{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1,1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1,
   -1, -1, -1, -1, -1, 1, 1, -1, 1, 1, 1, -1, 1, 1, -1, 1, 1, 1, -1, 1, -1, -1, -1,1, -1, -1, -1, 1},
   {1, -1, -1, -1, -1, -1, 1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1,1, -1, 1, 1, 1, 1, -1, 1, -1, -1, -1, -1, -1,
    1, 1, -1, 1, 1, 1,1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1},
   {1, 1, -1, -1, -1, -1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1,1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1,
    1, 1, 1, 1, 1,1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, -1, -1, -1, -1,1, 1},
   {1, 1, -1, -1, -1, 1, 1, 1, -1, 1, 1, 1, -1, 1, 1, -1, 1, 1, 1,1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, 1, -1, 1,
   -1, 1, 1, 1,1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, 1, -1, 1, 1, 1, -1, -1, -1,1, 1, 1},
   {-1, -1, -1, -1, -1, -1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1,1, -1, 1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1, 1,
    1, -1, 1, -1, 1, 1, 1,1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,1, -1, -1, -1, -1, -1, -1, -1, -1, 1},
   {1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1,1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1,
    1, 1, -1, 1, 1, -1, 1,1, 1, -1, 1, 1, -1, 1, 1, 1, -1, 1, 1, 1, -1, -1, -1, 1, 1, 1},
    {1, -1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, 1, 1, 1, -1, 1, -1, 1, 1, 1,1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1,
   -1, 1, -1, 1, 1, 1,1, -1, 1, 1, -1, 1, 1, 1, -1, 1, 1, 1, -1, 1, 1, -1, 1, 1, 1, -1, -1,1}};
Print["This Char is tested"]
Do[ListDensityPlot[Reverse[Partition[ch[[i]],7]]],{i,Length[ch]}];
perc[x,t]
```

Example 2.8 (page 28)

```
spc[x_,t_]:=Module[{w,b,α,θ,i,j,done,yin,y},w=Table[0,{i,Length[x[[1]]]}];θ=0.1;α=1;done=False;
                  For[j=1,done==False,j++, done=True;
                    Do[yin=w.x[[i]];If[yin>θ,y=1,If[yin<-θ,y=-1,y=0]];
                     If[y!=t[[Mod[i-1,7]+1]],w=w+α t[[Mod[i-1,7]+1]] x[[i]];
                       done=False],{i,Length[x]}](*End Do*)];(*End For*)Return[w]];
                     perc[x_,t_]:=Module[{w,i,u},w=Table[0,{7},{64}];
                           Do[u=Transpose[t][[i]];w[[i]]=spc[x,u];(*Training*)
                                 Print["w[[" ,i,"]]=  ",w[[i]]],{i,7}];(*End Do*)Return[w]];
test[w_,ch_]:=Module[{i,θ,found,chs,y,t},θ=0.1;y=Table[0,{7}];
                  Do[If[w[[i]].ch>θ,y[[i]]=1,If[w[[i]].ch<-θ,y[[i]]=-1,Print["Unsure"]]],
                  {i,Length[w]}];(*End Do*)Print["Tested Input ",a];Print["w.ch = ",w.ch];
                  Print["Therefore y= ",y];found=False;t={-1,-1,-1,-1,-1,-1,1};chs="ABCDEJK";
                    For[i=1,i≤7,i++,t=RotateRight[t];If[y==t,Print["The Character is ",
                     StringTake[chs,{i}]];found=True]](*End For*)
                       If[found==False,Print["No Character"]];Print["---------------------------------"]];
```

```
x={{1,1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,1,1,1,-
1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,1,-1,-1,-1,1,-1,-1,-1,1},{-1,-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,-
1,1,1,1,1,-1,1,-1,-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1},{1,1,-1,-
1,-1,-1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-
1,1,1,1,1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,1,1,1,1,-1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-
1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,1,1,-
1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1},{1,1,1,-1,-1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,-1,1,1,1,1,-1,-1,-1,1,1,1,1},{-1,-1,-1,1,1,1,-1,-1,1,-1,1,1,1,-1,1,1,1,-
```

1,1,-1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,-1,1,1,-1,-
1,1}(*Input1*),
{1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,-1,-1,-1,-
1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-
1,-1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1,1},{1,1,-1,-1,-1,1,1,1,1,-
1,1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,-1,1,1,1,1,-
1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-
1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-
1,1},{1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,-1,1,1,1,-
1,1,1,-1,1,1,1,-1,1,1,1,-1,-1,-1,1,1,1},{-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,-1,1,1,-1,1,1,1,-1,1,-1,1,1,1,1,-1,-
1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,1,1,-1,1,1,1,-1,1,1}(*Input2*),
{1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,-1,-1,-1,1,-
1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,-1,-1,1},{-1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,-
1,-1,-1,-1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1},{1,1,-1,-1,-1,-
1,1,-1,1,-1,1,1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,-
1,1,1,1,-1,1,1,1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,-1,-1,-1,1,1,1},{1,1,1,1,-1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-1,-1,-1,1,1,1},{-1,-1,-1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-1,-1,1,1,1,-1,-
1,1}};
t={{1,-1,-1,-1,-1,-1,-1},{-1,1,-1,-1,-1,-1,-1},{-1,-1,1,-1,-1,-1,-1},{-1,-1,-1,1,-1,-1,-1},{-1,-1,-1,-1,1,-1,-
1},{-1,-1,-1,-1,-1,1,-1},{-1,-1,-1,-1,-1,-1,1}};
ch={{1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,1,-1,-1,-1,-1,-1,1,1,-
1,1,-1,1,-1,1,1,-1,1,1,1,-1,1,-1,1,-1,1,-1,1},{-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,-
1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,1,-1,1,1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1},{1,1,-1,-
1,-1,-1,-1,-1,-1,1,1,1,-1,1,-1,-1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-
1,1,1,-1,1,-1,1,1,1,-1,-1,-1,-1,1,1},{-1,-1,-1,-1,-1,1,1,1,-1,1,-1,1,-1,1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,-1,1,-
1,1,1,1,1,1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,1},{-1,-1,-1,1,-1,-1,-1,-1,1,-
1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,-1,-1,1,1,1,-1,1,-1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-
1,-1,-1,-1,-1,-1,-1,1},{1,1,1,-1,-1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-
1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-
1,1}(*Tested Input 1*),
{1,1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,-1,1,-1,-1,-1,1,1,-1,1,-
1,1,1,1,1,1,1,-1,1,1,1,-1,1,-1,1},{-1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,-1,-
1,1,-1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,1,-1,1,1,1,-1,-1,-1,-1,-1,1,1},{1,1,-1,-1,-1,1,1,1,-

```
1,1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,-1,1,-1,1,-1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,1,-1,-1,-1,1,1,1}, {1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,-
1,1,1,1,-1,-1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-1,1,1,1,1,-1,1,1,-1,-1,-1,-1,1,1,1}, {1,-1,-1,-1,-1,-1,-
1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-1,-1,-1,-1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,1}, {1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,-1,-1,1,1,1,1}, {1,1,1,1,1,1,1,-1,1,1,1,-1,1,1,-1,1,1,-
1,1,1,1,-1,1,-1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-
1,1,1,1,1,1,1,1,1,1,1}(*Tested Input 2*),
{1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,1,-1,1,1,1,-1,1,-1,1,1,-1,1,1,1,1,1,-
1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,1,-1,-1,1}, {-1,-1,-1,-1,-1,-1,1,-1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,-1,-1,-1,-
1,1,-1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1,1}, {1,1,-1,-1,-1,1,-
1,1,-1,1,1,1,-1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,-
1,1,1,1,1,-1,-1,1,1,1}, {-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,-1,-1,1,1,1,1,-
1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,1}, {-1,-1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,-1,1,1,-1,1,-1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,-1,1,-1,-1,1,-1,-1,-
1,1}, {1,1,1,1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-
1,1,1,-1,1,1,1,1,1,1,1,-1,-1,-1,1,1,1,1}, {1,-1,-1,1,1,-1,1,1,-1,1,1,1,-1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,1,1,1,-1,-
1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,-1,1}};
Print["These Chars are tested."]
Do[ListDensityPlot[Reverse[Partition[ch[[a]],7]]],{a,Length[ch]}]
FrameBox[StyleBox["Finally Weight Changes","Section"]]//DisplayForm
w=perc[x,t];Print[""];Do[test[w,ch[[a]]],{a,Length[ch]}];
```

Chapter 3

Example 3.1 (page 33)

```
Maxnet[ε_]:Module[{a, k, m, j, ac},a ={0.25, 0.45, 0.65, 0. 85};done = False; ac = a;
            For[m=1, done = = False, m++, done = True; Print["Epoch = ", m];
              Do[x = a[[j]] - ε Sum[a[[k]],{k, Length[a]}]-a[[j]]);
                  If[x>0,ac[[j]] = x, ac[[j]] = 0], {j, Length[a]}]; (*End Do*)
            Print[ac]; If[Count[ac,0] < Length[ac] - 1, done = = False,
            done = True];a = ac](*End For*)];
Maxnet[0.1];
```

Example 3.2 (page 35)

```
Lvq[x_, m_, n_] := Module[{i, j, norm, a, d, w, α, done, T, C}, w = {{1, 0, 0, 0}, {0, 0, 1, 0}};
    α = 0.1; C = {1, 2}; T = {1, 1, 2}; done = False;
      For[i = 1, done == False, i++, Print["Epoch ", i];
        Do[xnew = Table[x[[a]], {Length[w]}]; d = xnew - w;
                              norm = Table[d[[j]].d[[j]], {j, Length[w]}];
                              j = Position[norm, Min[norm]][[1, 1]];
                              If[C[[j]] == T[[a]], w[[j]] = w[[j]] + α (x[[a]] - w[[j]] ),
                              (*Else*) w[[j]] = w[[j]] - α (x[[a]] - w[[j]] )];
                              Print["j= ", j, " ", w[[j]]], {a, Length[x]}];
                              (*End Do*) Print["α ", α= α - i/100];
                               done = (α < 0.01)] (*End For*) ];
x={{1, 1, 0, 0},{1, 0, 0, 1},{0, 1, 1, 0}};Lvq[x,3,4];
```

Example 3.3 (page 36)

```
Lvq[x_,m_,n_]:=Module[{i,j,norm,xnew,a,d,w,α,done,T,C},w={{0,0},{1,1},{1,0},{0,1}};
T={1,1,1,1,1,1,1,1,3,3,3,1,1,1,1,1,1,1,1,3,3,3,1,1,1,1,1,1,1,3,3,1,1,1,1,1,1,1,3,
    3,1,1,1,1,1,1,1,1,3,3,3,1,1,1,1,1,1,1,1,2,2,2,1,1,1,1,1,1,1,1,2,2,2,4,4,4,4,4,4,2,2,
    2,2,4,4,4,4,4,4,4,2,2,2,2,2};done=False; α =0.1;C={1,2,3,4};
              For[i=1,done==False,i++,Print["Epoch ",i];
              Do[xnew=Table[x[[a]],{Length[w]}];d=xnew-w;
              norm=Table[d[[j]].d[[j]],{j,Length[w]}];
              j=Position[norm,Min[norm]][[1,1]];Print["Closest node ",j];
                   If[C[[j]]==T[[a]],w[[j]]=w[[j]]+α (x[[a]]-w[[j]]),(*ELSE*)
                       w[[j]]=w[[j]]- α (x[[a]]-w[[j]])],{a,Length[x]}];(*END DO*)
                       Print["w= ",N[w,2]];Print["α = ",α = α -(i/625)];
                       done=(α<0.02)](*END FOR*)];
x={{0.1,0.1},{0.2,0.1},{0.3,0.1},{0.4,0.1},{0.5,0.1},{0.6,0.1},{0.7,0.1},{0.8,0.1},
{0.9,0.1},{0.1,0.2},{0.2,0.2},{0.3,0.2},{0.4,0.2},{0.5,0.2},{0.6,0.2},{0.7,0.2},
{0.8,0.2},{0.9,0.2},{0.1,0.3},{0.2,0.3},{0.3,0.3},{0.4,0.3},{0.5,0.3},{0.6,0.3},
{0.7,0.3},{0.8,0.3},{0.9,0.3},{0.1,0.4},{0.2,0.4},{0.3,0.4},{0.4,0.4},{0.5,0.4},
{0.6,0.4},{0.7,0.4},{0.8,0.4},{0.9,0.4},{0.1,0.5},{0.2,0.5},{0.3,0.5},{0.4,0.5},
{0.5,0.5},{0.6,0.5},{0.7,0.5},{0.8,0.5},{0.9,0.5},{0.1,0.6},{0.2,0.6},{0.3,0.6},
{0.4,0.6},{0.5,0.6},{0.6,0.6},{0.7,0.6},{0.8,0.6},{0.9,0.6},{0.1,0.7},{0.2,0.7},
{0.3,0.7},{0.4,0.7},{0.5,0.7},{0.6,0.7},{0.7,0.7},{0.8,0.7},{0.9,0.7},{0.1,0.8},
{0.2,0.8},{0.3,0.8},{0.4,0.8},{0.5,0.8},{0.6,0.8},{0.7,0.8},{0.8,0.8},{0.9,0.8},
{0.1,0.9},{0.2,0.9},{0.3,0.9},{0.4,0.9},{0.5,0.9},{0.6,0.9},{0.7,0.9},{0.8,0.9},{0.9,0.9}};
Lvq[x,81,2];
```

Example 3.4 (page 40)

```
Som[x_, m_, n_] : = Module[{a, j, i, d, w, α, R, done, t, xnew, norm},
        w = {{0.1, 0.8, 0.5, 0.9},{0.2, 0.5, 0.1, 0.9}}; α = 0.6;
        Print["Initial weights = ", w]; R = 0; done = False;
          For[t=1, done==False, t++, Print["Epoch ", t];
            Do[xnew = Table[x[[a]], {m}]; d= xnew- w; norm = Table[d[[j]].d[[j]],{j, m}];
                Print["norm= ", N[norm,2]]; J = Position[norm, Min[norm]][[1,1]];
                Print["Closest to node ", j]; w[[j]] = [[j]] + α (x[[a]]-w[[j]]);
                Print["w = ",          N[w,2]],{a, Length[x]}];(*End Do*)
            Print[" _____"];
            Print[" α = ", N[α = 0.96 α, 2]];done = (α < 0.01)](*End For*)];


x = {{1, 0, 0, 0},{0, 1, 0, 1},{1, 0, 1, 0},{0, 0, 1, 1}};
Som[x, 2, 4];
```

Case 1 (page 41)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α, R, done, t, xnew, norm},
        w=Table[N[Random[Real,{0.1,0.9}],1],{25},{63}]; α=0.6;
        Print["Initial weight matrix w= ",w// MatrixForm];R=0;done=False;
        For[t=1,done= =False, t++, Print["Epoch ",t];
          Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
              Print["norm= ",N[norm,2]];j=Position[norm, Min[norm]][[1,1]];
              Print["Closest to node ",j];w[[j]]=w[[j]]+α (x[[a]]-w[[j]]),{a, Length[x]}];(*End Do*)
          Print["------------------------------------"];Print["α = ",N[α=0.96  α,2]];done=( α<0.01)]
        (*End For*)];
```

```
x={{1,1,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,0,
    0,0,0,0,1,1,0,1,1,1,0,1,1,0,1,1,1,0,1,0,0,0,1,0,0,0},{0,0,0,0,0,0,1,1,0,
    1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,0,0,0,0,0,1,1,0,1,1,1,1,0,1,0,1,
    1,1,1,0,1,0,1,1,1,1,0,0,0,0,0,0,0,1},{1,1,0,0,0,0,0,1,0,1,1,1,1,0,0,1,1,
    1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,
    1,1,0,1,1,0,0,0,0,1},{0,0,0,0,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1,0,1,1,
    1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,0,1,0,0,0,0,0,
    1,1},{0,0,0,0,0,0,0,1,0,1,1,1,1,0,1,0,1,1,1,1,1,1,0,1,0,1,1,1,1,0,0,0,1,
    1,1,1,0,1,0,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,0,0,0,0,0,0,0,0},{1,1,1,0,0,
    0,0,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,
    1,1,0,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,0,0,1,1},{0,0,0,1,1,0,0,1,0,1,1,0,1,
    1,1,0,1,0,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,1,0,1,1,1,1,0,1,1,0,1,1,
    1,0,1,1,1,0,1,0,0,0,1,1,0,0}(*input1*),
   {1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,0,1,
    1,1,0,1,1,0,0,0,0,0,1,1,0,1,1,1,0,1,1,0,1,1,1,0,1},{0,0,0,0,0,0,1,0,1,1,1,1,1
   ,0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,0,0,0,0,1,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,1
   ,1,1,1,1,0,0,0,0,0,0,0,1},{1,1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,1,1,1,0,0,1,1,1
   ,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,0,1,1,1,0,0,0,1,1},
{0,0,0,0,0,1,1,0,1,1,1,1,0,1,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,
 1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,0,1,0,0,0,0,0,1,1},{0,0,0,0,0,0,0,0,1,1,1,1,
 1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,0,0,0,0,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,
 1,1,1,1,1,1,0,0,0,0,0,0,0},{1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,
 1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,0,0,1,1},
{0,1,1,1,1,0,1,0,1,1,1,0,1,1,0,1,1,0,1,1,1,0,1,0,1,1,1,1,0,0,1,1,1,1,1,0,1,0,
 1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1}(*Input2*),
{1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,0,1,1,1,0,1,1,0,0,
 0,0,0,1,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,0,1,1,1,0,0},{0,0,0,0,0,0,1,1,0,1,1,1,
 1,0,1,0,1,1,1,1,0,1,0,0,0,0,0,1,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,
 0,1,1,1,1,0,0,0,0,0,0,0,1},{1,1,0,0,0,1,0,1,0,1,1,1,0,0,0,1,1,1,1,1,0,0,1,1
,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,0,1,1,1,0,0,0,1,1},
{0,0,0,0,0,1,1,1,0,1,1,1,0,1,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,0,1,1,1,1,0,1,0,
1,1,1,1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,0,0,0,0,1,1},{0,0,0,0,0,0,0,1,0,1,1,1
,1,0,1,0,1,1,0,1,1,1,0,0,0,0,1,1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0
,1,1,1,1,0,0,0,0,0,0,0,0},{1,1,1,1,0,0,0,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1
,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,0,0,1,1},
{0,0,0,1,1,0,0,1,0,1,1,1,0,1,1,0,1,1,0,1,1,1,0,1,0,1,1,1,1,0,0,1,1,1,1,1,0,
 1,0,1,1,1,1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,0,0,1,1,0,0}(*input3*)};Som[x,25,63];
```

Case 4 (page 43)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α,done, t, xnew, norm, δ,μ},
              w=Table[N[Random[Real,{0.1,0.9}],1],{25},{63}];
              α=0.6; μ = Sum[x[[i]], {i, Length[x]}] / Length[x]; Print["μ = ", μ];
              δ = 1/ Length[x] * Sum[(x[[a]] - μ) . (x[[a]] - μ), {a, Length[x]}]; Print["δ = ", δ];
              Print["Initial weight matrix w= ",w// MatrixForm];done=False;
                 For[t=1,done = = False, t++, Print["Epoch ",t];
                    Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
```

```
                Print["norm= ",N[norm,2]];j=Position[norm, Min[norm]][[1,1]];
                Print["Closest to node ",j]; w[[j]]=w[[j]]+α (x[[a]]-w[[j]]),
            {a, Length[x]}];(*End Do*) Print["------------------------------------"];
        Print["α = ",N[α= α Exp[ - (1/ δ),2]];done=( α< 0.01)](*End For*)];

x={{1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,-1,-1,-1,-1,1,1,-
1,1,1,1,-1,1,1,-1,1,1,1,-1,1,-1,-1,-1,1,-1,-1,-1},{-1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1},{1,1,-1,-
1,-1,-1,-1,1,-1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,1,-
1,1,1,1,1,-1,1,1,-1,-1,-1,-1,1},{-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,1,-
1,1,1,1,1,-1,1,-1,1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,-1,1,1,1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,1,-1,-1,-
1,-1,-1,-1,-1,-1},{1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-
1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-1,-1,-1,1,1,1},{-1,-1,-1,1,1,-1,-1,1,-1,1,1,-1,1,1,1,-1,1,-1,1,1,1,1,-
1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,-1,1,1,-1,-1}(*input1*),
{1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,-1,-1,-1,-
1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,-1,1},{-1,-1,-1,-1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,-
1,-1,-1,-1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,1},{1,1,-1,-1,-1,1,1,1,-
1,1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,1,-
1,-1,-1,1,1},{-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,-
1,-1,1,1,1,1,1,-1,-1,1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1},{-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-
1,1,1,1,1,1,1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-
1},{1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,-1,1,1,1,-
1,1,1,-1,1,1,1,-1,1,1,1,-1,-1,-1,1,1},{-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,-1,1,1,-1,1,1,1,-1,1,-1,1,1,1,1,-1,-
1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1}(*Input2*),
{1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,-1,-1,-1,1,-
1,1,1,1,1,1,-1,-1,1,1,1,1,1,-1,-1,-1,1,1,1,-1,-1},{-1,-1,-1,-1,-1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,-1,-
1,-1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,1},{1,1,-1,-1,-1,1,-
1,1,-1,1,1,1,-1,-1,-1,1,1,1,1,1,-1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,-1,1,-1,1,1,1,-
1,1,1,1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,1,1,1,-1,1,1,1,-1,1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,-1,1,1,1,-1,1,-1,-1,-1,-1,-1,1,1,1},{-1,-1,-1,-1,-1,-1,-1,1,-1,1,1,1,1,-1,1,-
1,1,1,-1,1,1,1,-1,-1,-1,-1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,-1,-1,-1,-1,-1,-1,-
1},{1,1,1,1,-1,-1,-1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,1,1,-
1,1,1,-1,1,1,1,-1,1,1,1,-1,-1,-1,1,1,1},{-1,-1,-1,1,1,-1,-1,1,-1,1,1,1,-1,1,1,-1,1,1,-1,1,1,1,-1,1,-1,1,1,1,1,-
1,-1,1,1,1,1,1,-1,1,-1,1,1,1,1,-1,1,1,-1,1,1,1,-1,1,1,1,-1,1,-1,-1,-1,1,1,-1,-1}(*Input3*)};
Som[x,25,63];
```

Case 6 (page 44)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α, done, t, xnew, norm, Nb, s},
    w=Table[N[Random[],1],{25},{63}];Nb={-1,0,1}; α =0.6;
        Print["Initial weight matrix w= ",w // MatrixForm];done=False;
            For[t=1,done= =False, t++, Print["Epoch ",t];
                Do[ xnew=Table[x[[a]],{m}];d = xnew-w; norm=Table[d[[j]].d[[j]],{j, m}];
                    Print["norm= ",N[norm,2]];j=Position[norm, Min[norm]][[1,1]];
                    Print["Closest to node ",j];
                    Do[s = j + Nb[[i]];If[1 ≤ s ≤ m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])]
                ,{a, Length[x]}](*End Do*)
        Print["------------------------------------"];Print[" α = ",N[ α =0.96 α,2]];
done=( α <0.01)](*End For*)];
Som[x,25,63];
```

Case 7 (page 45)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α, done, t, xnew, norm, Nb, s},
    w=Table[N[Random[Real,{0.1, 0.9}],1],{25},{63}];Nb={-1,0,1}; α =0.6;
    Print["Initial weight matrix w= ",w // MatrixForm];done=False;
    For[t=1,done= =False, t++, Print["Epoch ",t]; If[t > 60, Nb = {0};
```

```
        Print["Nb = ", Nb]]; Do[ xnew=Table[x[[a]],{m}];d = xnew-w;
        norm=Table[d[[j]].d[[j]],{j, m}];Print["norm= ",N[norm,2]];
        j=Position[norm, Min[norm]][[1,1]];Print["Closest to node ",j];
          Do[s = j + Nb[[i]];If[1 ≤ s ≤ m,w[[s]]=w[[s]]+ α (x[[a]]-w[[s]])]
          ,{i, Length[Nb]}] ,{a, Length[x]}](*End Do*)
              Print[" ----------------------------------------------------"];
              Print["α = ", N[α = 0.6 - 0.0059*t, 2]]; done = ( α < 0.01)](*End For*)];
```

Case 8 (page 46)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α, done, t, xnew, norm, Nb, s, μ,δ},
    w=Table[N[Random[],1],{25},{63}];
    μ= Sum[x[[i]],{i, Length[x]}] / Length[x];Print[" μ= ", μ];Nb={-1,0,1}; α =0.6;
    δ = 0;Print["Initial weight matrix w= ",w // MatrixForm];done=False;
      For[t=1,done= =False, t++, Print["Epoch ",t];
          Do[ δ = δ+ (x[[a]] - μ ) . (x[[a]] - μ / Length[x]; xnew=Table[x[[a]],{m}];
          d = xnew-w; norm=Table[d[[j]].d[[j]],{j, m}];Print["norm= ",N[norm,2]];
          j=Position[norm, Min[norm]][[1,1]];Print["Closest to node ",j];
            Do[s = j + Nb[[i]];If[1 ≤ s ≤ m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])]
            ,{i, Length[Nb]}],{a, Length[x]}](*End Do*)
              Print["-------------------------------------"];
              Print[" α = ",N[ α = α* Exp[ - 1/ δ,2]];done=( α <0.01)](*End For*)];
```

Case 9 (page 46)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α, done, t, xnew, norm, Nb, s, μ,δ},
    w=Table[N[Random[Real,{ 0.1, 0.9}],1],{25},{63}];
    δ = (x[[a]] - μ ) . (x[[a]] - μ) / Length[x];
    μ= Sum[x[[i]],{i, Length[x]}] / Length[x];Print[" μ= ", μ];Nb={-1,0,1}; α =0.6;
    δ = 0;Print["Initial weight matrix w= ",w // MatrixForm];done=False;
      For[t=1,done= =False, t++, Print["Epoch ",t]; If[ t > 70, Nb = {0};
          Print["Nb = ", Nb];Do[ δ = δ+ (x[[a]] - μ ) . (x[[a]] - μ) / Length[x];
          xnew=Table[x[[a]],{m}];d = xnew-w; norm=Table[d[[j]].d[[j]],{j, m}];
          Print["norm= ",N[norm,2]];j=Position[norm, Min[norm]][[1,1]];
          Print["Closest to node ",j];
            Do[s = j + Nb[[i]];If[1 ≤ s ≤ m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],
            {i, Length[Nb]}],{a, Length[x]}](*End Do*)
              Print["-------------------------------------"];
      Print[" α = ",N[ α = α* Exp[ - 1/ δ],2]];done=( α <0.01)](*End For*)];
```

Case 10 (page 47)

```
Som[x_,m_,n_]:=Module[{a,j,i,d,w,α,done,t,xnew,norm,Nb,s},
          w=Table[N[Random[],1],{25},{63}];Nb={-1,0,1,63,-63};α=0.6;
          Print["Initial weight matrix w= ",w//MatrixForm];done=False;
            For[t=1,done==False,t++, Print["Epoch ",t];
              Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
               Print["norm= ",N[norm,2]];j=Position[norm,Min[norm]][[1,1]];
               Print["Closest to node ",j];
                  Do[s=j+Nb[[i]];If[1≤s≤m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],
                  {i,Length[Nb]}],{a,Length[x]}](*End Do*)
      Print["-------------------------------------"];
      Print["α= ",N[α=0.96  α,2]];done=(α<0.01)](*End For*)];
```

Case 11 (page 48)

```
Som[x_, m_, n_]:=Module[{a, j, i, d, w, α,done, t, xnew, norm, δ,μ},
          w=Table[N[Random[Real,{0.1,0.9}],1],{25},{63}]; α=0.6;
```

```
            μ = Sum[x[[i]], {i, Length[x]}] / Length[x]; Print["μ = ", μ];
            δ = 1/ Length[x] * Sum[(x[[a]] - μ) . (x[[a]] - μ), {a, Length[x]}]; Print["δ = ", δ];
            Nb = {-63, -1, 0, 1, 63}; Print["Initial weight matrix w= ",w//MatrixForm];
            done=False;
            For[t=1,done = = False, t++, Print["Epoch ",t]; If[70 < t < 100, Nb = {-1, 0, 1};
            Print["Nb = ", Nb]; If[100 ≤ t ≤ 186, Nb= {0}; Print["Nb = ", Nb]];
              Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
                Print["norm= ",N[norm,2]];j=Position[norm, Min[norm]][[1,1]];
                Print["Closest to node ",j];w[[j]]=w[[j]]+α (x[[a]]-w[[j]]);
            ,{a, Length[x]}];(*End Do*)
        Print["---------------------------------------"];
        Print["α = ",N[α= α Exp[ - (1/ δ),2]];done=( α< 0.01)](*End For*)];


Case 12 (page 49)
Som[x_, m_, n_]:=Module[{a,j,i,d,w,α,done,t,xnew,norm,δ,μ},
                    w=Table[N[Random[Real,{0.1,0.9}],1],{25},{63}]; α=0.6;
                    Nb = {-63, -1, 0, 1, 63};
                    Print["Initial weight matrix w= ",w// MatrixForm];done=False;
                    For[t=1,done = = False, t++, Print["Epoch ",t];
                      If[70 < t < 100, Nb = {-1, 0, 1};Print["Nb = ", Nb];
                      If[100 ≤ t ≤ 186, Nb= {0}; Print["Nb = ", Nb]];
                      Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
                        Print["norm= ",N[norm,2]];j=Position[norm, Min[norm]][[1,1]];
                        Print["Closest to node ",j];w[[j]]=w[[j]]+α (x[[a]]-w[[j]])
                      ,{a, Length[x]}];(*End Do*)
                  Print["---------------------------------------"];
                  Print["α = ",N[α= α Exp[ - (441/ 19912),2]];done=( α< 0.01)](*End For*)];


Case 13 (page 50)
Som[x_,m_,n_]:=Module[{a,j,i,d,w,α,done,t,xnew,norm,Nb,s},
        w=Table[N[Random[],1],{25},{63}];Nb={-1,0,1,63,-63};α=0.6;
        Print["Initial weight matrix w= ",w//MatrixForm];done=False;
          For[t=1,done==False,t++, Print["Epoch ",t];If[60<t<80, Nb= {-1,0,1}];
            If[t ≥ 80, Nb ={0}]; Print[" Nb= ", Nb];
            Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
            Print["norm= ",N[norm,2]];j=Position[norm,Min[norm]][[1,1]];
            Print["Closest to node ",j];
            Do[s=j+Nb[[i]];If[1≤s≤m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])]
            ,{i,Length[Nb]}],{a,Length[x]}](*End Do*)
            Print["---------------------------------------"];
Print["α= ",N[α=0.96  α,2]];done=(α<0.01)](*End For*)];


Case 14 (page 51)
Som[x_,m_,n_]:=Module[{a,j,i,d,w,α,done,t,xnew,norm,Nb,s},
        w=Table[N[Random[],1],{25},{63}];Nb={-1,0,1,63,-63, 62, -62, 64, -64};α=0.6;
        Print["Initial weight matrix w= ",w//MatrixForm];done=False;
    For[t=1,done==False,t++, Print["Epoch ",t]; If[t>= 80, Nb={0}];Print["Nb = ",Nb];
          Do[ xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
            Print["norm= ",N[norm,2]];
                j=Position[norm,Min[norm]][[1,1]];Print["Closest to node ",j];
                  Do[s=j+Nb[[i]];If[1≤s≤m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],
                    {i,Length[Nb]}],{a,Length[x]}](*End Do*)
                    Print["---------------------------------------"];
          Print["α= ",N[α= 0.6 - 0.00012 t ,2]];done=(α<0.01)](*End For*)];
```

Example 3.4 (page 54)

```
Tsp[city_,m_,n_]:=Module[{a,j,i,d,w,α,done,t,xnew,norm,Nb,s,wei,ci},
        w={{0.72,1},{0.92,0.66},{0.4,0.39},{0.78,0.55},{0.47,0.74},{0.74,0.49},
        {0.82,0.59},{0.7,0.34},{0.37,0.27},{0.74,0.13}};α=0.5;
        Print["Initial weight matrix weight mattix w= ",w//MatrixForm];done=False;
        For[t=1,done==False,t++,Print["Epoch ",t];
          Do[xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
          Print["norm= ",N[norm,2]];j=Position[norm,Min[norm]][[1,1]];
          Print["Closest to node ",j];If[t<=100,Nb={-1,0,1};Do[s=j+Nb[[i]];
          If[1<=s<=m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]]);Print["w[",s,"]= ",w[[s]]]],{i,Length[Nb]}];
          α=0.5-0.001*t ,(*Else*) Nb={0};w[[j]]=w[[j]]+α (x[[a]]-w[[j]]);
          Print["w[",j,"]= ",N[w[[j]],2]];α=0.4-0.002*(t-100)](*End if*)
          ,{a,Length[x]}](*End Do*);done=(α<0.02);Print["α= ",N[α,2]];
    Print["w= ",N[w,2]//MatrixForm];Print["--------------------------"]];(*End For*)
    wei= ListPlot[w,Prolog->AbsolutePointSize[3],Frame->True,
    PlotRange->{{0,1},{0,1}},AspectRatio->1];
    ci=ListPlot[x,Prolog->AbsolutePointSize[5],PlotRange->{{0,1},{0,1}},
   Frame->True,AspectRatio->1]];
```

Chapter 4

Case 1(page 63)

```
Bp61[x_] := Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,σin,delv,e,j,etotal},
    v = {{0.1970, 0.3099, -0.3378}, {0.3191, 0.1904, 0.2771}, {-0.1448, -0.0347, 0.2859}
        ,{0.3594,-0.4861, -0.3329}};done = False; etotal = Table[0, {3000}];e = Table[0, {4}];
    w = {0.4919, -0.2913, -0.3979, 0.3581, -0.1401}; t = {0, 1, 1, 0}; α = 0.2;

(*FeedForward*)
    For[i = 1, done == False, i++,
     Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}];z = Table[1/(1 + Exp[-zin]];
      dfzin = Table[z[[p]]*(1 - z[[p]]), {p, Length[zin]}];z = Append[z, 1]; yin = w.z;
      y = 1/(1 + Exp[-yin];dfyin = y*(1 - y);

(*Backpropagation error*)
       σk = (t[[n]] - y)*dfyin;delw = α*σk*z;σin = Table[σk*w[[j]], {j, 4}];σj = σin*dfzin;
       delv = α*Table[σj[[j]]*x[[n]], {j, Length[σj]}];

(*Update weights & biases*)
       w = w + delw;v = v + delv; e[[n]] = (y - t[[n]])^2;Print["e[[", n, "]]= ", e[[n]]], {n, Length[x]}];
      (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];Print["total error", i, "= ", etotal[[i]]];
  done = (etotal[[i]] < 0.05)]; (*End For*)
  ListPlot[etotal,PlotJoined -> True,PlotRange ->{{1, 3000},{0, 1.2}},Frame->True,AspectRatio-> 1,
          PlotLabel -> StyleForm[Backpropagation, "section"]]];
x={{1,1,1},{1,0,1},{0,1,1},{0,0,1}};Bp61[x];
```

Case 2 (page 64)

```
Bp61[x_] := Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,σin,delv,e,j,etotal},
    v = {{0.1970, 0.3099, -0.3378}, {0.3191, 0.1904, 0.2771},{-0.1448, -0.0347, 0.2859},
        {0.3594, -0.4861, -0.3329}}; done = False; etotal = Table[0, {388}];e = Table[0, {4}];
    w = {0.4919, -0.2913, -0.3979, 0.3581,-0.1401};t = {-1, 1, 1, -1};α = 0.2;

(*Feed Forward*)
    For[i = 1, done == False, i++,
     Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}];z = Table[2/(1 + Exp[-zin]  - 1];
      dfzin = Table[0.5*(1 + z[[p]])*(1 - z[[p]]),{p,Length[zin]}];z =Append[z, 1]; yin = w.z;
```

```
    y = 2/(1 + Exp[-yin]) - 1;dfyin = 0.5*(1 + y)*(1 - y);

(*Backpropagation error*)
    σk = (t[[n]] - y)*dfyin;delw = α*σk*z;σin = Table[σk*w[[j]], {j, 4}];σj = σin*dfzin;
     delv = α*Table[σj[[j]]*x[[n]], {j, Length[σj]}];

(*Update weights & biases*)
    w = w + delw;v = v + delv; e[[n]] = (y - t[[n]])^2;Print["e[[", n, "]]= ", e[[n]]], {n, Length[x]}];
    (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];
    Print["total error", i, "= ", etotal[[i]]];
    done = (etotal[[i]] < 0.05)]; (*End For*)
    ListPlot[etotal,PlotJoined->True,PlotRange->{{1,388},{0, 5}},Frame -> True, AspectRatio->1,
        PlotLabel -> StyleForm[Backpropagation, "section"]]];
x={{1,1,1},{1,-1,1},{-1,1,1},{-1,-1,1}};Bp61[x];
```

Case 3 (page 64)

```
Bp61[x_] :=Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,σin,delv,e,j,etotal},
   v = {{0.1970, 0.3099, -0.3378}, {0.3191, 0.1904, 0.2771},{-0.1448, -0.0347, 0.2859},
        {0.3594, -0.4861,-0.3329}}; done = False; etotal = Table[0, {265}];e = Table[0, {4}];
   w = {0.4919, -0.2913, -0.3979, 0.3581, -0.1401}; t = {-0.8, 0.8, 0.8, -0.8}; α = 0.2;

(*Feed Forward*)
   For[i = 1, done == False, i++,
    Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}]; z = Table[2 /(1 + Exp[-zin]   - 1];
     dfzin = Table[0.5*(1 + z[[p]])*(1 - z[[p]]), {p,Length[zin]}]; z = Append[z, 1]; yin = w.z;
     y = 2/(1 + Exp[-yin]   - 1; dfyin = 0.5*(1 + y)*(1 - y);

(*Backpropagation error*)
    σk = (t[[n]] - y)*dfyin; delw = α*σk*z; σin = Table[σk*w[[j]], {j, 4}];σj = σin*dfzin;
    delv = α*Table[σj[[j]]*x[[n]], {j, Length[σj]}];

(*Update weights & biases*)
    w = w + delw; v = v + delv; e[[n]] = (y - t[[n]])^2;Print["e[[", n, "]]= ", e[[n]]], {n, Length[x]}];
    (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];Print["total error", i, "= ", etotal[[i]]];
    done = (etotal[[i]] < 0.05)]; (*End For*)
    ListPlot[etotal, PlotJoined -> True, PlotRange->{{1,265},{0,5}},Frame->True,AspectRatio -> 1,
    PlotLabel -> StyleForm[Backpropagation, "section"]]];
 x = {{1, 1, 1}, {1, -1, 1}, {-1, 1, 1}, {-1, -1, 1}};
 Bp61[x];
```

Case 4 (page 65)

```
Bp61[x_] :=Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,σin,delv,e,j,etotal,normv},
    v = {{0.1970, 0.3099,-0.3378}, {0.3191, 0.1904, 0.2771}, {-0.1448, -0.0347, 0.2859},
         {0.3594, -0.4861,-0.3329}}; done = False; etotal = Table[0, {77}];e = Table[0, {4}];
    normv = Table[√v[[p]].v[[p]], {p, Length[v]}];v = 1.4*v /normv; Print["Nguyen v is ", v];
    w = {0.4919, -0.2913, -0.3979, 0.3581, -0.1401};t = {0, 1, 1, 0}; α = 0.2;

(*Feed Forward*)
   For[i = 1, done == False,i++,
    Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}];z = Table[Tanh[zin[[p]]], {p, Length[zin]}];
     dfzin = Table[1 - z[[p]]^2, {p, Length[zin]}]; z = Append[z, 1]; yin = w.z; y = Tanh[yin];
     dfyin = 1 - y^2;

(*Backpropagation error*)
    σk = (t[[n]] - y)*dfyin; delw = α*σk*z; σin = Table[σk*w[[j]], {j, 4}];σj = σin*dfzin;
    delv = α*Table[σj[[j]]*x[[n]], {j, Length[σj]}];
```

```
(*Update weights &biases*)
      w = w + delw; v = v + delv; e[[n]] = (y - t[[n]])^2;Print["e[[", n, "]]= ", e[[n]]], {n, Length[x]}];
     (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];Print["total error", i, "= ", etotal[[i]]];
  done =(etotal[[i]] < 0.05)]; (*End For*)
  ListPlot[etotal, PlotJoined -> True, PlotRange -> {{0, 77},{0, 5}},Frame -> True, AspectRatio -> 1,
          PlotLabel -> StyleForm[Backpropagation, "section"]]];
 x = {{1, 1, 1}, {1, 0, 1}, {0, 1, 1}, {0, 0, 1}}; Bp61[x];
```

Case 5 (page 66)

```
Bp61[x_] :=Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,σin,delv,e,j,etotal,normv},
   v = {{0.1970, 0.3099, -0.3378}, {0.3191, 0.1904, 0.2771}, {-0.1448, -0.0347, 0.2859},
        {0.3594, -0.4861,-0.3329}}; done = False; etotal = Table[0, {388}];e = Table[0, {4}];
    normv = Table[√v[[p]].v[[p]], {p, Length[v]}]; v = 1.4*v/ normv; Print["Nguyen v is ", v];
    w = {0.4919, -0.2913,-0.3979, 0.3581,-0.1401}; t = {-1, 1, 1, -1}; α = 0.2;

(*Feed Forward*)
    For[i = 1, done == False, i++,
     Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}];z = Table[2 /(1 + Exp[-zin])  - 1];
        dfzin = Table[0.5*(1 + z[[p]])*(1 - z[[p]]), {p,Length[zin]}]; z = Append[z, 1]; yin = w.z;
        y = 2/(1 + Exp[-yin]) - 1; dfyin = 0.5*(1 + y)*(1 - y);

(*Backpropagation error*)
      σk = (t[[n]] - y)*dfyin;delw = α*σk*z;σin = Table[σk*w[[j]], {j, 4}];σj = σin*dfzin;
      delv = α*Table[σj[[jj]]*x[[n]], {j, Length[σj]}];

(*Update weights & biases*)
    w = w + delw;v = v + delv; e[[n]] = (y - t[[n]])^2; Print["e[[", n, "]]= ", e[[n]]], {n, Length[x]}];
     (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];Print["total error", i, "= ", etotal[[i]]];
     done = (etotal[[i]] < 0.05)]; (*End For*)
     ListPlot[etotal, PlotJoined ->True,PlotRange->{{1, 388},{0, 5}},Frame->True,AspectRatio -> 1,
     PlotLabel -> StyleForm[Backpropagation, "section"]]];
x={{1,1,1},{1,-1,1},{-1,1,1},{-1,-1,1}};Bp61[x];
```

Case 6 (page 67)

```
Bp61[x_] :=Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,σin,delv,e,j,etotal, normv},
        v = {{0.1970, 0.3099,-0.3378}, {0.3191, 0.1904, 0.2771}, {-0.1448, -0.0347, 0.2859},
        {0.3594, -0.4861, -0.3329}}; done = False; etotal = Table[0, {52}];e = Table[0, {4}];
        normv = Table[√v[[p]].v[[p]], {p, Length[v]}];v = 1.4*v /normv; Print["Nguyen v is ", v];
        w = {0.4919, -0.2913, -0.3979, 0.3581,-0.1401};t = {-1, 1, 1,-1}; α = 0.2;

(*FeedForward*)
    For[i = 1, done == False, i++,
     Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}];z = Table[Tanh[zin[[p]]], {p, Length[zin]}];
        dfzin = Table[1 - z[[p]]^2, {p, Length[zin]}];z = Append[z, 1]; yin = w.z; y = Tanh[yin];
        dfyin = 1 - y^2;

(*Backpropagation error*)
        σk = (t[[n]] - y)*dfyin; delw = α*σk*z;σin=Table[σk*w[[j]], {j, 4}];σj=σin*dfzin;
        delv = α*Table[σj[[jj]]*x[[n]], {j, Length[σj]}];

(*Update weights &biases*) w = w + delw;
        v = v + delv; e[[n]] = (y - t[[n]])^2; Print["e[[", n, "]]= ", e[[n]]], {n, Length[x]}];
     (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];Print["total error", i, "= ", etotal[[i]]];
  done = (etotal[[i]] < 0.05)]; (*End For*)
  ListPlot[etotal, PlotJoined -> True, PlotRange -> {{0, 52}, {0, 7}},Frame->True,AspectRatio -> 1,
          PlotLabel -> StyleForm[Backpropagation, "section"]]];
 x = {{1, 1, 1}, {1, -1, 1}, {-1, 1, 1}, {-1, -1, 1}};
 Bp61[x];
```

Case 7 (page 68)

```
(* 2 layers NN with Bp (1 hidden layer) : v from input, x, to hidden, z,
    w (from hidden, z,  to output, y)  *)
 Bp61[x_] := Module[{w,y,σj,σk,v,α,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,in,delv,e,j,etotal, normv},
    v = {{0.1970, 0.3099, -0.3378}, {0.3191, 0.1904, 0.2771}, {-0.1448, -0.0347, 0.2859}
        ,{0.3594, -0.4861,-0.3329}}; done = False; etotal = Table[0, {25}];e = Table[0, {4}];
    normv = Table[√v[[p]].v[[p]], {p, Length[v]}]; v = 1.4*v /nomv; Print["Nguyen v is ", v];
    w = {0.4919, -0.2913, -0.3979, 0.3581, -0.1401};t = {-0.8, 0.8, 0.8, -0.8}; α= 0.2;

(*Feed Forward*)
    For[i = 1, done == False, i++,
     Do[zin = Table[x[[n]].v[[j]], {j, Length[v]}];z = Table[Tanh[zin[[p]]], {p, Length[zin]}];
        dfzin = Table[1 - z[[p]]^2, {p, Length[zin]}]; z = Append[z, 1]; yin = w.z; y = Tanh[yin];
        dfyin = 1 - y^2;

 (*Backpropagation error*)
        σk = (t[[n]] - y)*dfyin;delw = α* σk*z;σin = Table[σk*w[[j]], {j, 4}];σj = σin*dfzin;
        delv = α*Table[σj[[j]]*x[[n]], {j, Length[σj]}];

(*Update weights & biases*)
        w = w + delw; v = v + delv; e[[n]] = (y - t[[n]])^2;Print["e[[", n, "]]= ", e[[n]]],
        {n, Length[x]}]; (*End Do*) etotal[[i]] = Sum[e[[p]], {p, Length[x]}];
     Print["total error", i, "= ", etotal[[i]]]; done = (etotal[[i]] < 0.05)];  (*End For*)
    ListPlot[etotal, PlotJoined -> True, PlotRange -> {{0, 25}, {0, 5}},Frame->True,AspectRatio->1,
          PlotLabel -> StyleForm[Backpropagation, "section"]]];
 x = {{1, 1, 1}, {1, -1, 1}, {-1, 1, 1}, {-1, -1, 1}};
 Bp61[x];
```

Program 1 (page 71)

```
Compres[x_, h_] := Module[{w,y,σj,σk,v,σ,w1,i,z,p,n,done,zin,yin,dfyin,dfzin,delw, tol1,σin,
                    delv,e,j,etotal,k,a,b,normv,etotalold},b=Table[0,{Length[h]}];etotalold = 0;
        Do[Print["h[[", a, "]]= ", h[[a]]]; v = Table[N[Random[Real, {-0.5, 0.5}], 2], {h[[a]]}, {57}];
           w = Table[N[Random[Real, {-0.5, 0.5}], 2], {56}, {h[[a]] + 1}];
           normv = Table[√v[[p]].v[[p]], {p, Length[v]}]; v = 0.7*h[[a]]^(1/56)*v/normv;
           done = False; e = Table[0, {Length[x]}]; α = 0.2;

    t = {{1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
        1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0}, {0, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
        1, 0, 0, 0, 0, 0, 1, 1}, {1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
        1, 1}, {0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1}, {0, 0, 0,
        0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
        1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1}, {0, 0, 0, 0, 0, 0, 1, 1, 0,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
        1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 1, 1, 1}, {1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1}, {
        0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
        1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
        1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0}, {1, 0, 0, 0, 0, 0,
```

```
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
        1, 1, 0, 0, 0, 0, 0, 1}, {1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
        1, 1}};

(*Feed Forward*)
    For[i = 1, done == False, i++, b[[a]] = i;
     Do[zin = Table[x[[n]]. v[[j]], {j, Length[v]}];z = Table[1/(1 + Exp[-zin]];
       dfzin = Table[z[[p]]*(1 - z[[p]]), {p, Length[zin]}];z = Append[z, 1];
       yin = Table[w[[k]].z, {k, Length[w]}];y = 1/(1 + Exp[-yin];
       Do[If[y[[p]] > 0.8, y[[p]] = 1, If[y[[p]] < 0.2, y[[p]] = 0]], {p, Length[y]}];
         dfyin = y*(1 - y);

(*Backpropagation error*)
          σk = (t[[n]] - y)*dfyin; delw = Table[α*(σk[[k]]* z), {k, Length[y]}];w1 = Transpose[w];
          σin = Table[σk.w1[[j]], {j, Length[w1] - 1}]; σj = σin*dfzin;
          delv = α *Table[σj[[j]]* x[[n]], {j,Length[σj]}];

(*Update weigths & biases*)
          w = w + delw; v = v + delv;e[[n]] = 0.5*(y - t[[n]]).(y - t[[n]]),{n,Length[x]}] (*End Do*);
       etotal = Sum[e[[p]], {p, Length[x]}]; Print["total error", i, "= ", etotal];
      done = (etotal == etotalold); etotalold = etotal](*End For*);Print["timeused= ", TimeUsed[]];
    Print["_____"], {a, Length[h]}];Print["total epoch= ", b];
    tol1 = ListPlot[b, PlotRange -> {{0, Length[h]}, {0, 1300}},AxesLabel -> {hidden unit, epoch},
Frame -> True, AspectRatio -> 1,PlotLabel -> "Data Compression by Bp:Tolerance=0.2"]];
```

Program 2 (page 71)

```
Compres[x_, h_] := Module[{w,y,σj,σk,v,α,w1,i,z,p,n,done,zin,yin,dfyin,dfzin,delw,tol1,σin,delv,e,j,
                          etotal,k,a,b,normv,etotalold},b = Table[0, {Length[h]}]; etotalold = 0;
       Do[Print["h[[", a, "]]= ",h[[a]]];v=Table[N[Random[Real,{-0.5,0.5}],2],{h[[a]]}, {57}];
            w = Table[N[Random[Real, {-0.5, 0.5}], 2], {56}, {h[[a]] + 1}];
            normv =Table[√v[[p]].v[[p]],{p,Length[v]}];v=0.7*h[[a]]^(1/56)*v/normv;done= False;
            e = Table[0, {Length[x]}]; α = 0.2;
    t = {{1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
        1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0}, {0, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
        1, 0, 0, 0, 0, 0, 1, 1}, {1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
        1, 1}, {0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1}, {0, 0, 0,
        0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
        1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1}, {0, 0, 0, 0, 0, 0, 1, 1, 0,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
        1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 1, 1, 1}, {1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1}, {
        0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
        1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
        1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0}, {1, 0, 0, 0, 0, 0,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
```

```
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
        1, 1, 0, 0, 0, 0, 0, 1}, {1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
        1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
        1, 1}};

(*Feed Forward*)
     For[i = 1, done == False, i++, b[[a]] = i;
      Do[zin = Table[x[[n]]. v[[j]], {j, Length[v]}];
       z = Table[1/(1 + Exp[-zin]]; dfzin = Table[z[[p]]*(1 - z[[p]]), {p, Length[zin]}];
       z = Append[z, 1]; yin = Table[w[[k]].z, {k, Length[w]}]; y = 1/(1 + Exp[-yin];
       Do[If[y[[p]] > 0.9, y[[p]] = 1,If[y[[p]] < 0.1, y[[p]] = 0]], {p, Length[y]}];dfyin = y*(1 - y);

(*Backpropagation error*)
       σk = (t[[n]] - y)*dfyin; delw = Table[α *(σk[[k]]* z), {k, Length[y]}];w1 = Transpose[w];
       σin = Table[σk.w1[[j]], {j, Length[w1] - 1}];σj =σin*dfzin;
       delv = α*Table[σj[[j]]* x[[n]], {j,Length[σj]}];

(*Update weigths & biases*)
    w = w + delw; v = v + delv;e[[n]] = 0.5*(y - t[[n]].(y - t[[n]]), {n,Length[x]}] (*End Do*) ;
    etotal = Sum[e[[p]], {p, Length[x]}]; Print["total error", i, "= ", etotal];
    done = (etotal == etotalold); etotalold = etotal] (*End For*) ; Print["timeused= ", TimeUsed[]];
    Print["_____"], {a, Length[h]}];Print["total epoch= ", b];
    toll = ListPlot[b, PlotRange -> {{0, Length[h]}, {0, 1300}}, AxesLabel -> {hidden unit, epoch},
        Frame -> True, AspectRatio -> 1, PlotLabel -> "Data Compression by Bp:Tolerance=0.2"]];
```

Chapter 5

Program 5.1 (page 80)

```
asm[rdb_] := Module[{hd,hdm,h,un,i,j,sca,mall,S,ss,totv,u,md,v,tv,b,bar,di,ang},
        hd = ReadList["hand.txt"]; hd = Flatten[hd, 1];
        Do[ListPlot[hd[[i]], PlotJoined -> True, AspectRatio -> Automatic], {i, 20}];
        (*Find its own gravity for each example*)
          hdm = Table[0, {Length[hd]}]; h = Table[0, {Length[hd]}];
          Do[hdm[[i]] = (1/Length[hd[[1]]] Sum[hd[[i, j]],{j,Length[hd[[1]]]}], {i, Length[hd]}];
        (*Move 1st hand to a new origin, i.e., hdm[[1]]*)
             Do[hd[[1, j]] = hd[[1, j]] - hdm[[1]], {j, Length[hd[[1]]]}];
                h[[1]] = ListPlot[hd[[1]], PlotJoined -> True, AspectRatio -> Automatic];
        (*Scale hd[[1]]*)
                hd[[1]] = Flatten[hd[[1]]];un =√Sum[hd[[1, j]]^2,{j,Length[hd[[1]]]}];
                hd[[1]] = 1/un hd[[1]];
        (*Check*)
                sca = Sum[hd[[1, j]]^2,{j=1,Length[hd[[1]]]}];
        (*Move 2nd hand's old origin to hdm[[1]]*)
                   For[i = 2, i <= Length[hd], i++, di = hdm[[1]] - hdm[[i]];
                    Do[hd[[i, j]] = hd[[i, j]] + di, {j, Length[hd[[2]]]}];
                          hdm[[i]] = 1/Length[hd[[2]]] Sum[hd[[i, j]],{j,Length[hd[[i]]]}];
                             Do[hd[[i, j]] = hd[[i, j]] - hdm[[i]], {j, Length[hd[[2]]]}]; ListPlot[hd[[i]],
                             PlotJoined -> True, AspectRatio -> Automatic]
        (* We will plot 1st hand - 20  th hand in the same axes later*) ] (*End For*) ;
                    Do[hd[[i]] = Flatten[hd[[i]]];
                    un = √Sum[hd[[i,j]]^2,{j,Length[hd[[2]]]}];hd[[i]] = (1/un) hd[[i]];
        (*Check*)
                     sca = Sum[hd[[i, j]]^2,{j,hd[[2]]}],{i, 2, Length[hd]}]; (*End Do*)
        (*This is for converting to figure*)
```

```
            Do[hd[[i]] = Partition[hd[[i]], {2}]; h[[i]] = ListPlot[hd[[i]], PlotJoined -> True,
            AspectRatio -> Automatic], {i, Length[hd]}]; (*End Do*) Show[h];
      (*Orientation*)
            ang = {0, 0, 10, 2,-13, 24, 4, 0,-2, 2, 8, 1, 4,-15, -6, 2,-3, 0, 20, -7};
            ang =ang*π/180; m = Table[0, {Length[ang]}];
            For[i = 1, i <= Length[hd], i++,
              Do[m[[i]] = {{Cos[ang[[i]]], -Sin[ang[[i]]]}, {Sin[ang[[i]]],Cos[ang[[i]]]}};
                hd[[i, j]] = m[[i]] . hd[[i, j]], {j, Length[hd[[2]]]}]; (*End Do*)
                h[[i]] = ListPlot[hd[[i]], PlotJoined -> True, AspectRatio -> Automatic]]; (*End
            For*) Show[h];
            mall = 1/Length[hd] Sum[hd[[i]],{i,Length[hd]}];
  (*Now we have mean shape of all shapes rotated scaled, and translated*)
  Print["This hand is the mean shape of all aligned shapes"];
  ListPlot[mall, PlotJoined -> True, AspectRatio -> Automatic];
  (*Find a covariance matrix S *)
  mall = Flatten[mall]; hd = Flatten[hd]; hd = Partition[hd, {92}];
  Print["dim[hd] now= ", Dimensions[hd]]; S = Table[0, {Length[mall]}];
  ss = Table[0, {Length[mall]}, {Length[mall]}];
  For[i = 1, i <= Length[hd], i++, bar = hd[[i]] - mall;
      Do[S[[j]] = bar[[j]] bar, {j, Length[bar]}]; ss = S + ss];(*End For*)
      {u, md, v} = SingularValues[N[ss]]; Print["md= ", N[md, 3]];b = Table[0, {92}, {19}];
      totv = Sum[md[[i]],{i,Length[md]}]; Print["Sum[λi,{i,t}]>= ", 0.98 totv];
      Print["t=8,Sum[λi,{i,8}]= ", Sum[md[[i]]],{i,8}]];
        Do[Print["i= ", i, "..Ratio λi/λt= ", md[[i]]/totv*100], {i, 8}]; (*End Do*)
  (*Find approximated x*)
  For[i = 1, i <= Length[hd], i++,
   Print["This is no. ", i, " shape"];b = v . (hd[[i]] - mall); tv = Transpose[v];
   Do[Print["This is λ no. ", k]; Do[b[[k]] = b[[k]] - rdb[[k, j]]; hdnew = mall + (tv . b);
     ListPlot[Partition[hdnew, 2], AspectRatio -> Automatic, PlotJoined -> True, Axes -> False];
     b = v . (hd[[i]] - mall), {j, 7}], {k, 8}] (*End Do*) ] (*End For*) ] (*End Module*)
rdb={{-0.3,-0.2,-0.1,0,0.1,0.2,0.3},{-0.2,-0.15,-0.1,0,0.1,0.15,0.2},
   {-0.11,-0.08,-0.04,0,0.04,0.08,0.11},{-0.06,-0.04,-0.02,0,0.02,0.04,0.06},
   {-0.05,-0.03,-0.015,0,0.015,0.03,0.05},{-0.049,-0.03,-0.02,0,0.02,0.03,0.049},
   {-0.04,-0.025,-0.015,0,0.015,0.025,0.04},{-0.035,0.025,0.015,0,0.015,0.025,0.035}};asm[rdb];
```

Program 5.2 (page 84)

```
GenerateF[rdb_] := Module[{hd,hdnew,h,hdm,mall,di,un,sca,i,j,u,v,md,totv,tv,ss,S,b,k},(*Find its own
gravity for each example*)
  hd = ReadList["a:/mt.txt"]; hd = Flatten[hd, 1]; hdm = Table[0, {Length[hd]}];
  Do[(hdm[[i]] = 1/Length[hd[[1]]] Sum[hd[[i, j]],{j,Length[hd[[1]]]}], {i, Length[hd]}];
      h = Table[0, {Length[hd]}];
(*Move 1st hand to a new origin, ie, hdm[[1]]*)
      Do[hd[[1, j]] = hd[[1, j]] - hdm[[1]], {j, Length[hd[[1]]]}];
  CreateFace[x_] := Module[{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v},
      a = Take[hd[[x]], 19]; a = Append[a, a[[1]]];
      b = ListPlot[a, PlotJoined -> True, AspectRatio -> Automatic];
      c = Take[hd[[x]], {20, 25}]; c = Append[c, c[[1]]];
      d = ListPlot[c, PlotJoined -> True, AspectRatio -> Automatic];
      e = Take[hd[[x]], {26, 31}]; e = Append[e, e[[1]]];
      f = ListPlot[e, PlotJoined -> True, AspectRatio -> Automatic];
      g = Take[hd[[x]], {32, 39}]; g = Append[g, g[[1]]];
    (*Right eye*) h = ListPlot[g, PlotJoined -> True, AspectRatio -> Automatic];
    i = Take[hd[[x]], {40, 47}]; i = Append[i, i[[1]]];
    (*Left eye*) j = ListPlot[i, PlotJoined -> True, AspectRatio -> Automatic];
    k = Take[hd[[x]], {48, 54}];
    (*Nose*) l = ListPlot[k, PlotJoined -> True, AspectRatio -> Automatic];
    m = Take[hd[[x]], {55, 61}]; n = ListPlot[m, PlotJoined -> True, AspectRatio -> Automatic];
    (*Mouth*) o = Take[hd[[x]], {62, 73}];
    o = Append[o, o[[1]]]; p = ListPlot[o, PlotJoined -> True, AspectRatio -> Automatic];
```

```
      q = Take[hd[[x]], {74, 81}]; q = Append[q, q[[1]]];
      r = ListPlot[q, PlotJoined -> True, AspectRatio -> Automatic];
       (*Eye balls*) s = Take[hd[[x]], {82, 87}]; s = Append[s, s[[1]]];
      t = ListPlot[s, PlotJoined -> True, AspectRatio -> Automatic];
      u = Take[hd[[x]], {88, 93}]; u = Append[u, u[[1]]];
      v = ListPlot[u, PlotJoined -> True, AspectRatio -> Automatic];
      Show[b, d, f, h, j, l, n, p, r, t, v]]; CreateFace[1];
(*Scale hd[[1]]*)
        hd[[1]] = Flatten[hd[[1]]];
        un = √Sum[hd[[1, j]]^2,{j,Length[hd[[1]]}]; hd[[1]] = 1/un  hd[[1]];
(*Check*)
        sca =Sum[hd[[1, j]]^2,{j,Length[hd[[1]]}];
(*Move 2nd hand' s old origin to hdm[[1]]*)
        For[x = 2, x <= Length[hd], x++, di = hdm[[1]] - hdm[[x]];
          Do[hd[[x, j]] = hd[[x, j]] + di, {j, Length[hd[[2]]]}];hdm[[x]] = 1/Length[hd[[2]]]
          Sum[hd[[x, j]],{j,Lenght[hd[[x]]}];
          Do[hd[[x, j]] = hd[[x, j]] - hdm[[x]], {j, Length[hd[[2]]]}]; CreateFace[x];
(*We will plot 1st hand - 20th hand in the same axes later*) ] (*End For*) ;
        Do[hd[[i]] = Flatten[hd[[i]]];
        un =√Sum[hd[[i, j]]^2),{j,Length[hd[[2]]}]; hd[[i]] = 1/un  hd[[i]];
 (*Check*)
        sca = Sum√hd[[i, j]]^2,{j,Length[hd[[2]]}], {i, 2, Length[hd]}];(*End Do*)
(*This is for converting to figure*)
        Do[hd[[i]] = Partition[hd[[i]], {2}], {i, Length[hd]}]; (*End Do*)
        mall = 1/Length[hd] Sum[hd[[i]],{i,Length[hd]}];
(*Find S*)
        mall = Flatten[mall]; hd = Flatten[hd];
        hd = Partition[hd, {186}]; S = Table[0, {Length[mall]}];
        ss = Table[0, {Length[mall]}, {Length[mall]}];
    For[i = 1, i <= Length[hd], i++, bar = hd[[i]] - mall;
     Do[S[[j]] = bar[[j]] bar, {j, Length[bar]}];
     ss = S + ss]; (*End For*) {u, md, v} = SingularValues[N[ss]];
    Print["md= ", N[md, 3]]; Dimensions[u]; Dimensions[md];
    totv = Sum[md[[i]],{I,Length[md]}]; Print["Sum[λI,{i,t}]>= ", 0.98 totv];
    Do[Print["i= ", i, "..Ratio λi/λt= ", md[[i]] totv*100], {i, 8}]; (*End Do*)
        For[i = 1, i <= Length[hd], i++, Print["This is no. ", i, "  shape"];
          b = v.(hd[[i]] - mall); tv = Transpose[v];
         Do[Print["This is λ no. ", k];
          Do[b[[k]] = b[[k]] - rdb[[k, j]]; hdnew = mall + (tv.b); hdnew = Partition[hdnew, {2}];
           Print["hdnew= ", N[hdnew, 3]]; b = v.(hd[[i]] - mall), {j, 8}], {k, 7}] (*End Do*) ]
        (*End For*) ];
rdb={{-0.115,-0.11,-0.1,0,0.1,0.11,0.113,0.115},{-0.068,-0.06,-0.04,-0.2,0,0.02,0.04,0.06},
    {-0.058,-0.05,-0.03,-0.01,0.1,0.2,0.04,0.58},{-0.052,-0.045,-0.035,-0.02,0,0.02,0.04,0.05},
    {-0.046,-0.03,-0.02,-0.01,0,0.01,0.02,0.04},{-0.044,-0.033,-0.022,-0.011,0,0.011,0.022,0.04},
    {-0.038,-0.029,-0.019,-0.01,0,0.01,0.02,0.035}};GenerateF[rdb];
```

Program 5.3 (page 85)

```
GenerateF[rdb_] := Module[{hd,hdnew,h,hdm,mall,di,un,sci,i,j,u,v,md,totv,tv,ss,S,b,k},
(*Find its own gravity for each example*)
  hd = ReadList["a:/realproj.txt"]; hd = Flatten[hd, 1];hdm = Table[0, {Length[hd]}];
   Do[hdm[[i]] =1/Length[hd[[1]]] Sum[hd[[i, j]],{j,Length[hd[[1]]}],{i, Length[hd]}];
      h = Table[0, {Length[hd]}];
(*Move 1st hand to a new origin, ie, hdm[[1]]*)
   Do[hd[[1, j]] = hd[[1, j]] - hdm[[1]], {j, Length[hd[[1]]]}];
  CreateFace[x_] := Module[{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v},
      a = Take[hd[[x]], 22];a = Append[a, a[[1]]];
      b = ListPlot[a, PlotJoined -> True, AspectRatio -> Automatic];
      (*Eye - brows*) c = Take[hd[[x]], {23, 30}]; c = Append[c, c[[1]]];
      d = ListPlot[c, PlotJoined -> True, AspectRatio -> Automatic];
```

```
        e = Take[hd[[x]], {31, 38}]; e = Append[e, e[[1]]];
        f = ListPlot[e, PlotJoined -> True, AspectRatio -> Automatic];
        (*Eyes*) g = Take[hd[[x]], {39, 44}]; g = Append[g, g[[1]]];
        h = ListPlot[g, PlotJoined -> True, AspectRatio -> Automatic];
        i = Take[hd[[x]], {45, 50}]; i = Append[i, i[[1]]];
        (*Left eye*) j = ListPlot[i, PlotJoined -> True, AspectRatio -> Automatic];
        (*Eye balls*) k = Take[hd[[x]], {51, 56}]; k = Append[k, k[[1]]];
        l = ListPlot[k, PlotJoined -> True, AspectRatio -> Automatic];
        m = Take[hd[[x]], {57, 62}]; m = Append[m, m[[1]]];
        n = ListPlot[m, PlotJoined -> True, AspectRatio -> Automatic];
        (*Nose*) o = Take[hd[[x]], {63, 69}];
        p = ListPlot[o, PlotJoined -> True, AspectRatio -> Automatic]; q = Take[hd[[x]], {70, 76}];
        r = ListPlot[q, PlotJoined -> True, AspectRatio -> Automatic];
        (*Mouth*) s = Take[hd[[x]], {77, 84}]; s = Append[s, s[[1]]];
        t = ListPlot[s, PlotJoined -> True, AspectRatio -> Automatic];
        u = Take[hd[[x]], {85, 92}]; u = Append[u, u[[1]]];
        v = ListPlot[u, PlotJoined -> True, AspectRatio -> Automatic];
        Show[b, d, f, h, j, l, n, p, r, t, v]]; CreateFace[1];
(*Scale hd[[1]]*)
          hd[[1]] = Flatten[hd[[1]]];
          un = √Sum[hd[[1, j]]^2,{j,Length[hd[[1]]}]; hd[[1]] = 1/un  hd[[1]];
(*Check*)
sca = Sum[hd[[1, j]]^2,{j,Length[hd[[1]]}];
(*Move2  nd hand' s old origin to hdm[[1]]*)
    For[x = 2, x <= Length[hd], x++, di = hdm[[1]] - hdm[[x]];
      Do[hd[[x, j]] = hd[[x, j]] + di, {j, Length[hd[[2]]]}];
      hdm[[x]] = 1/Length[hd[[2]]]  Sum[hd[[x, j]],{j,Length[hd[[x]]}];
      Do[hd[[x, j]] = hd[[x, j]] - hdm[[x]], {j,Length[hd[[2]]]}]; CreateFace[x];
(*We will plot 1st face - 30th face in the same axes later*) ] (*End For*) ;
      Do[hd[[i]] = Flatten[hd[[i]]]; un =√Sum[hd[[i, j]]^2,{j,Length[hd[[2]]}]; hd[[i]] = 1/un  hd[[i]];
 (*Check*)
       sca = Sum[hd[[i, j]]^2, {j,Length[hd[[2]]}],{i, 2, Length[hd]}];  (*End Do*)
 (*This is for converting to figure*)
    Do[hd[[i]] = Partition[hd[[i]], {2}], {i, Length[hd]}]; (*End Do*)
    mall = 1/Length[hd] Sum[hd[[i]],{i,Length[hd]}];
(*Find S*)
    mall = Flatten[mall]; hd = Flatten[hd]; hd = Partition[hd, {184}]; S = Table[0, {Length[mall]}];
    ss = Table[0, {Length[mall]}, {Length[mall]}];
    For[i = 1, i <= Length[hd], i++, bar = hd[[i]] - mall;
      Do[S[[j]] = bar[[j]] bar, {j, Length[bar]}];
      ss = S + ss];  (*End For*) {u, md, v} = SingularValues[N[ss]];
    Print["md= ", N[md, 3]]; Dimensions[u]; Dimensions[md];
    totv = Sum[md[[i]],{i,Length[md]}];
    Print["Sum[λi]>= ", 0.98 totv];
    Do[Print["i= ", i, "..Ratio λi/λt= ", md[[i]]/totv*100], {i, 22}];  (*End Do*)
(*Find approximated x *)
    For[i = 1, i <= Length[hd], i++, Print["This is no. ", i, "  shape"]; b = v.(hd[[i]] - mall);
         tv = Transpose[v];
      Do[Print["This is λ no. ", k];
        Do[b[[k]] = b[[k]] - rdb[[k, j]]; hdnew = mall + (tv.b); hdnew = Partition[hdnew, {2}];
          Print["hdnew= ", N[hdnew, 3]]; b = v.(hd[[i]] - mall), {j, 29}], {k, 3}] (*End Do*) ]
     (*End For*) ];
rdb={{-0.223,-0.22,-0.21,-0.2,-0.19,-0.18,-0.17,-0.16,-0.15,-0.14,-0.13,-0.12,-0.11,-
0.1,0,0.1,0.11,0.12,0.13,0.14,0.15,0.16,0.17,0.18,0.19,0.2,0.21,0.22,0.223},
{-0.18,-0.17,-0.165,-0.16,-0.155,-0.15,-0.145,-0.14,-0.135,-0.13,-0.125,-0.12,-0.115,-0.11,-0.105,
-0.1,0,0.1,0.105,0.11,0.115,0.12,0.125,0.13,0.14,0.15,0.16,0.17,0.18},
{-0.141,-0.14,-0.138,-0.135,-0.13,-0.128,-0.125,-0.12,-0.118,-0.115,-0.11,-0.105,-0.1,
-0.05,0,0.05,0.1,0.105,0.11,0.115,0.118,0.12,0.125,0.128,0.13,0.135,0.138,0.14,0.141}};
GenerateF[rdb]; Print["time used = ",TimeUsed[]];
```

Chapter 6

Case 1 (page 88)

```
x=ReadList["a:/alignedh567.txt"];x=Flatten[x];x=Partition[x,92];w=ReadList["a:/w.txt"];
w=Flatten[w];w=Partition[w,92];done=False;α=0.1;c={1,2,3};T={1,2,3,1,2,3,1,2,3,1,2,3,1,2,3};
  For[i=1,done==False,i++,Print["Epoch ",i];
   Do[xnew=Table[x[[a]],{Length[w]}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,Length[w]}];
   j=Position[norm,Min[norm]][[1,1]];Print["Closest to node ",j];If[c[[j]]==T[[a]],
   w[[j]]=w[[j]]+α (x[[a]]-w[[j]]),(*Else*)w[[j]]=w[[j]]-α (x[[a]]-w[[j]])],
   {a,Length[x]}];(*End Do*)
  Print["α= ",α=0.1-(i/500)];done=(α<0.02)];(*End For*)Print[w];
```

Case 3 (page 93)

```
Som[m_,n_]:=Module[{x,a,j,i,d,w,α,jnew,done,t,xnew,norm,Nb,s,μ,σ2},Nb={-2,-1,0,1,2};
        x=ReadList["a:/kohonenhand567.txt"];x=Flatten[x];x=Partition[x,n];
        α=0.6;done=False;j=jnew=Table[0,{Length[x]}];
        w=Table[N[Random[Real,{0.1,0.9}],1],{m},{n}];
        For[t=1,done==False,t++,Print["Epoch ",t];
          Do[xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
            j[[a]]=Position[norm,Min[norm]][[1,1]];
           Do[s=j[[a]]+Nb[[i]];If[1<=s<=m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],{i,Length[Nb]}]
           (*End do*),{a,Length[x]}](*End Do*);Print["j= ",j];
        Print["jnew=j ",(jnew==j)];Print["--------------------------"];Print["α= ",N[α=0.6-0.003*t,2]];
        If[90<t<170,Nb={-1,0,1}];If[t>=170,Nb={0}];Print["Nb= ",Nb];
        done=((jnew==j)&&(α<0.01));jnew=j](*End For*)];Som[33,92];
```

Case 5 (page 96)

```
Som[m_,n_]:=Module[{x,a,j,i,d,w,α,jnew,done,t,xnew,norm,Nb,s,μ,σ2},
                x=ReadList["a:/kohonenhand567.txt"];x=Flatten[x];x=Partition[x,n];
                α=0.6;done=False;j=jnew=Table[0,{Length[x]}];
                w=Table[N[Random[Real,{0.1,0.9}],1],{m},{n}];
                For[t=1,done==False,t++,Print["Epoch ",t];
                  Do[xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
                    j[[a]]=Position[norm,Min[norm]][[1,1]];Nb={-7,-6,-5,-1,0,1,5,6,7};
                    If[j[[a]]==1,Nb={0,1,6,7}];If[j[[a]]==6,Nb={-1,0,5,6}];
                    If[(j[[a]]==7)||(j[[a]]==19)||(j[[a]]==25)||(j[[a]]==13),Nb={-6,-5,0,1,6,7}];
                    If[(j[[a]]==12)||(j[[a]]==18)||(j[[a]]==24),Nb={-7,-6,-1,0,5,6}];
                    If[j[[a]]==27,Nb=Take[Nb,8]];If[j[[a]]==28,Nb=Take[Nb,7]];
                    If[j[[a]]==29,Nb=Take[Nb,6]];If[j[[a]]==30,Nb={-7,-6,-1,0}];
                    If[j[[a]]==31,Nb={-6,-5,0,1}];Print["Nb= ",Nb];
                    Do[s=j[[a]]+Nb[[i]];
                      If[1<=s<=m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],{i,Length[Nb]}]
                    (*End do*),{a,Length[x]}](*End Do*); Print["j= ",j];
                Print["jnew=j ",(jnew==j)];Print["--------------------------"];
                Print["α= ",N[α=0.6-0.003*t,2]];If[t>=120,Nb={0}];done=((jnew==j)&&(α<0));
                jnew=j](*End For*)];Som[33,92];
```

Case 6 (page 101)

```
x=ReadList["a:/facelvq.txt"];x=Flatten[x];x=Partition[x,186];w=ReadList["a:/mt1.txt"];w=Flatten[w];w
=Partition[w,186];done=False;α=0.1;c={1,2,3,4,5,6,7,8,9};
  For[i=1,done==False,i++,Print["Epoch ",i];
    Do[xnew=Table[x[[a]],{Length[w]}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,Length[w]}];
      j=Position[norm,Min[norm]][[1,1]];
```

```
If[1<=a<=7,T=1,If[8<=a<=14,T=2,If[15<=a<=21,T=3,If[22<=a<=28,T=4,
  If[29<=a<=35,T=5,If[36<=a<=42,T=6,
    If[43<=a<=49,T=7,If[50<=a<=56,T=8,If[57<=a<=63,T=9]]]]]]]];
Print["Closest to node ",j];If[c[[j]]==T,w[[j]]=w[[j]]+α (x[[a]]-w[[j]]),
(*Else*)w[[j]]=w[[j]]-α (x[[a]]-w[[j]])],{a,Length[x]}];(*End Do*)
Print["α= ",α=0.1-(i/500)];done=(α<0.01)];
```

Case 7 (page 103)

```
Som[m_,n_]:=
 Module[{x,a,j,i,d,w,α,jnew,done,t,xnew,norm,Nb,s,μ,σ2},
        x=ReadList["a:/facelvq.txt"];x=Flatten[x];x=Partition[x,n];
        α=0.6;done=False;j=jnew=Table[0,{Length[x]}];
        w=Table[N[Random[Real,{0.1,0.9}],1],{m},{n}];
      For[t=1,done==False,t++,Print["Epoch ",t];
        Do[xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
        j[[a]]=Position[norm,Min[norm]][[1,1]];Nb={-11,-10,-9,-1,0,1,9,10,11};
       If[j[[a]]==1,Nb={0,1,10,11}]; If[j[[a]]==10,Nb={-1,0,9,10}];
       If[(j[[a]]==11)||(j[[a]]==31)||(j[[a]]==21)||(j[[a]]==41)||(j[[a]]==51),
       Nb={-10,-9,0,1,10,11}];
       If[(j[[a]]==20)||(j[[a]]==30)||(j[[a]]==40)||(j[[a]]==50)||(j[[a]]==60),
       Nb={-11,-10,-1,0,9,10}];
       If[j[[a]]==61,Nb={-10,-9,0,1}];If[j[[a]]==70,Nb={-11,-10,-1,0}];
         Do[s=j[[a]]+Nb[[i]];If[1<=s<=m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],{i,Length[Nb]}]
         (*End do*),{a,Length[x]}](*End Do*); Print["j= ",j];
      Print["jnew=j  ",(jnew==j)];Print["--------------------------"];
      Print["α= ",N[α=0.6-0.003*t,2]];If[t>=120,Nb={0}];
      done=((jnew==j)&&(α<0));jnew=j](*End For*)];Som[70,186];
```

Case 8 (page 109)

```
x=ReadList["a:/p17r16.txt"];x=Flatten[x];x=Partition[x,184]; c={1,2,3};
w=ReadList["a:/f3grouplvq.txt"];w=Flatten[w];w=Partition[w,184];done=False;α=0.1;
For[i=1,done==False,i++,Print["Epoch ",i];
Do[xnew=Table[x[[a]],{Length[w]}];d=xnew-w;
  norm=Table[d[[j]].d[[j]],{j,Length[w]}];j=Position[norm,Min[norm]][[1,1]];
   If[1<=Mod[a,9]<=3,T=1,If[4<=Mod[a,9]<=6,T=2,If[7<=Mod[a,9]<=8,T=3,
    If[Mod[a,9]==0,T=3]]]];Print["Closest to node ",j];
      If[c[[j]]==T,w[[j]]=w[[j]]+α (x[[a]]-w[[j]]),(*Else*)
     w[[j]]=w[[j]]-α (x[[a]]-w[[j]])],{a,Length[x]}];(*End Do*)
         Print["α= ",α=0.1-(i/500)];done=(α<0.01)];(*End For*)
Print["time used= ",TimeUsed[]];
```

Case 9 (page 111)

```
Som[m_,n_]:=Module[{x,a,j,i,d,w,α,jnew,done,t,xnew,norm,Nb,s,μ,σ2},
        x=ReadList["a:/p17r16.txt"];x=Flatten[x];x=Partition[x,n];
        done=False;j=jnew=Table[0,{Length[x]}];
        w=Table[N[Random[Real,{0.1,0.9}],1],{m},{n}];
      For[t=1,done==False,t++,Print["Epoch ",t];
        Do[xnew=Table[x[[a]],{m}];d=xnew-w;norm=Table[d[[j]].d[[j]],{j,m}];
        j[[a]]=Position[norm,Min[norm]][[1,1]];Nb={-10,-9,-8,-1,0,1,8,9,10};
        If[j[[a]]==1,Nb={0,1,9,10}];If[j[[a]]==9,Nb={-1,0,8,9}];
        If[(j[[a]]==10)||(j[[a]]==19)||(j[[a]]==28)||(j[[a]]==37)||(j[[a]]==46)||(j[[a]]==55)||
        (j[[a]]==64)||(j[[a]]==73)||(j[[a]]==82)||(j[[a]]==91)||(j[[a]]==100),Nb={-9,-8,0,1,9,10}];
        If[(j[[a]]==18)||(j[[a]]==27)||(j[[a]]==36)||(j[[a]]==45)||( j[[a]]==54)||(j[[a]]==63)||
        (j[[a]]==72)||(j[[a]]==81)||(j[[a]]==90)||(j[[a]]==99)||(j[[a]]==108),Nb={-9,-1,0,8,9}];
```

```
            If[j[[a]]==109,Nb={-9,-8,0,1}];If[j[[a]]==117,Nb={-10,-9,-1,0}];
            Do[s=j[[a]]+Nb[[i]];If[1<=s<=m,w[[s]]=w[[s]]+α (x[[a]]-w[[s]])],
            {i,Length[Nb]}](*End do*),{a,Length[x]}](*End Do*); Print["j= ",j];
    Print["jnew=j  ",(jnew==j)];Print["----------------------------"];
    Print["α= ",N[α=0.7-0.00233*t,2]];If[t>=180,Nb={0}];
done=((jnew==j)&&(α<0.001));jnew=j](*End For*)];Som[117,184];
Print[“time used= “,TimeUsed[]];
```

References

[1] N.K.Bose, P. Liang, Neural Network Fundamentals with Graphs, Algorithm, and Applications, McGraw-Hill, 1996.

[2] C. M. Bishop, Neural Networks for Pattern Recognition, Clarendon Press Oxford, 1995.

[3] D. Nelson, The penguin dictionary of Mathematics, 2nd edition, Penguin books, 1998.

[4] http://casaxps.csc.net/FactorAnalysis.htm.
http://www.isbe.man.ac.uk/~bim/Models/pdms.html.
http://www.fon.hum.uva.nl/praat/manual/Principal_component_analysis.html

[5] I. Pratt, Artificial Intelligence, Macmillan, pp 216-231, 1994.

[6] J. T. Tou, R. C. Gonzales, Pattern Recognition Principles, Addison-Wesley, 1974.

[7] L. Fausett, Fundamentals of Neural Networks: Architectures, Algorithms, and Applications, Prentice Hall, 1994.

[8] M. A. Boden, Artificial Intelligence, Academic Press, 1996.

[9] M. James, Artificial Intelligence in Basic, Newness Microcomputer Books, 1984.

[10] P. Picton, Introduction to Neural Networks, Macmillan, 1994.

[11] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley, 1989.

[12] R. R. Beale, T. Jackson, Neural Computing: an intoduction, Adam Hilger, 1990.

[13] R. Rojas, Neural Networks: A Systematic Introduction, Springer, 1996.

[14] S. G. Hoggar, Mathematics for Computation Graphics (2nd impression) to appear, Cambridge University Press, 2001.

[15] T. F. Cootes, C.J. Taylor, D. H. Cooper, and J. Graham, Active Shape Models-Their Training and Application, Computer Vision and Image Understanding, Vol. 61, No. 1, Jan., pp. 38-59, 1995.

[16] T. Leonard, John S.J. Hsu, Bayesian Methods: An Analysis for Statisticians and Interdisciplinary Researchers, Cambridge University Press, 1999.