DECISION SYSTEMS INTERNATIONAL
Atlanta, Georgia, USA

Artificial Neural Networks in Electric Power Systems

Madrid, Spain
September 9~11, 1992

Day 1, Wednesday, September 9

13:00 - 14:00	Registration
14:00 - 15:30	Introduction to Neural Networks—*Marks*
15:30 - 16:00	Break
16:00 - 17:30	Hopfield Neural Networks—*Marks*

Day 2, Thursday, September 10

8:30 - 10:00	The Layered Perceptron—*Marks*
10:00 - 10:30	Break
10:30 - 12:00	The Layered Perceptron—*Marks*
12:00 - 14:00	Lunch
14:00 - 15:30	Neural Network Implementation—*Marks*
15:30 - 16:00	Break
16:00 - 17:30	Feature Extraction—*El-Sharkawi*
20:00 - 22:00	Banquet

Day 3, Friday, September 11

8:30 - 10:00	Applications to Power Systems—*El-Sharkawi*
10:00 - 10:30	Break
10:30 - 12:00	Applications to Power Systems—*El-Sharkawi*
12:00 - 14:00	Lunch
14:00 - 15:30	Applications to Power Systems—*El-Sharkawi*
15:30 - 16:00	Break
16:00 - 17:30	Case Study—*El-Sharkawi*

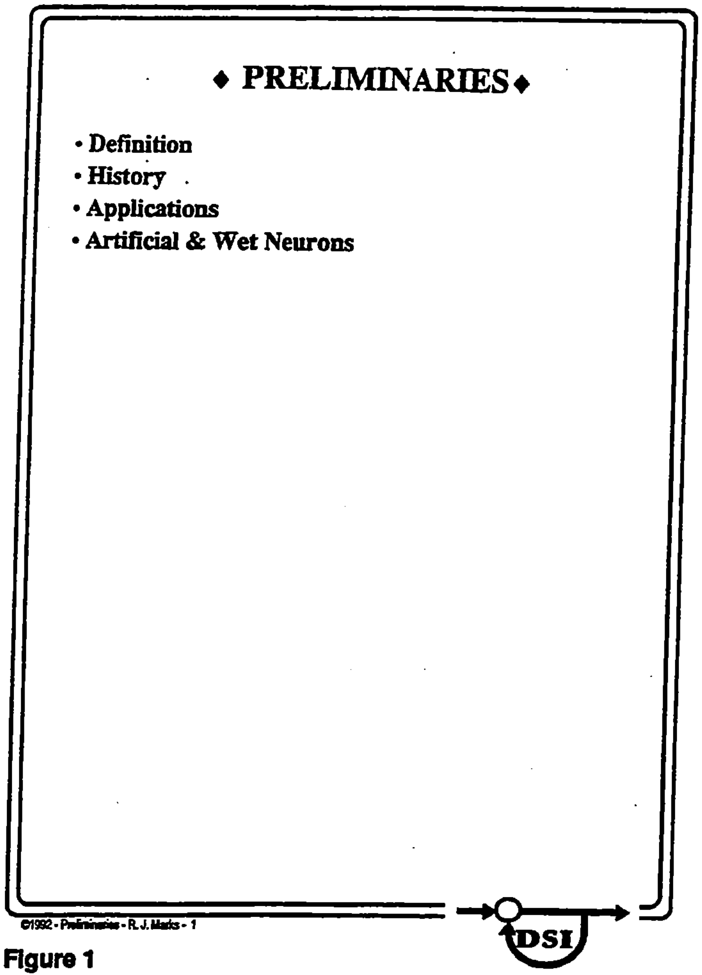
Preliminaries

Lecture 1

Preliminaries

Robert J. Marks, II

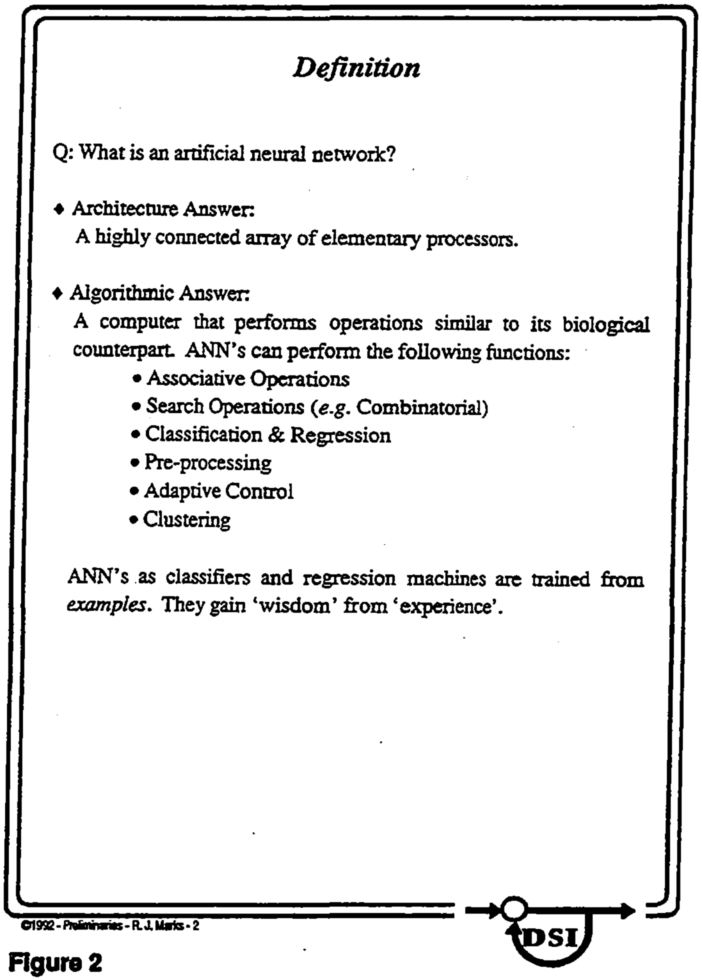
Notes Notes Notes



♦ PRELIMINARIES ♦

- Definition
- History
- Applications
- Artificial & Wet Neurons

Figure 1



Definition

Q: What is an artificial neural network?

♦ Architecture Answer:
A highly connected array of elementary processors.

♦ Algorithmic Answer:
A computer that performs operations similar to its biological counterpart. ANN's can perform the following functions:

- Associative Operations
- Search Operations (*e.g.* Combinatorial)
- Classification & Regression
- Pre-processing
- Adaptive Control
- Clustering

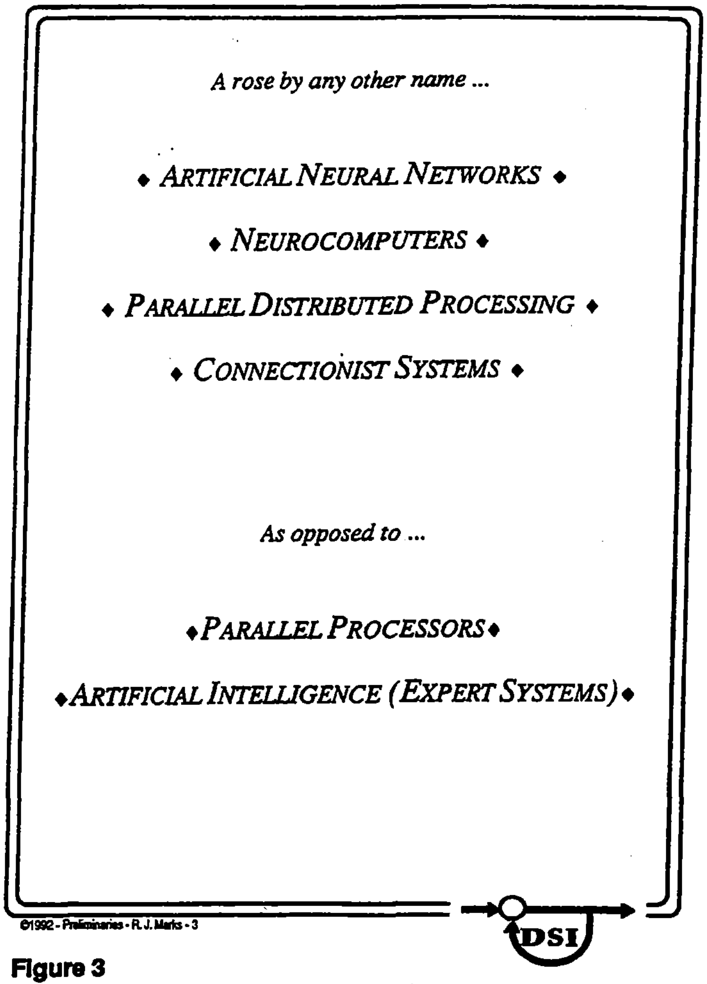
ANN's as classifiers and regression machines are trained from *examples*. They gain 'wisdom' from 'experience'.

©1992 - Preliminaries - R. J. Marks - 2

DSI

Figure 2

Artificial neural networks are typically defined either from their architecture or from the operations they perform. Architecturally, the ANN loosely resembles the biological neural network. Algorithmically, neural networks perform operations similar to certain biological neural network functions.



A rose by any other name ...

♦ *ARTIFICIAL NEURAL NETWORKS* ♦

♦ *NEUROCOMPUTERS* ♦

♦ *PARALLEL DISTRIBUTED PROCESSING* ♦

♦ *CONNECTIONIST SYSTEMS* ♦

As opposed to ...

♦*PARALLEL PROCESSORS*♦

♦*ARTIFICIAL INTELLIGENCE (EXPERT SYSTEMS)*♦

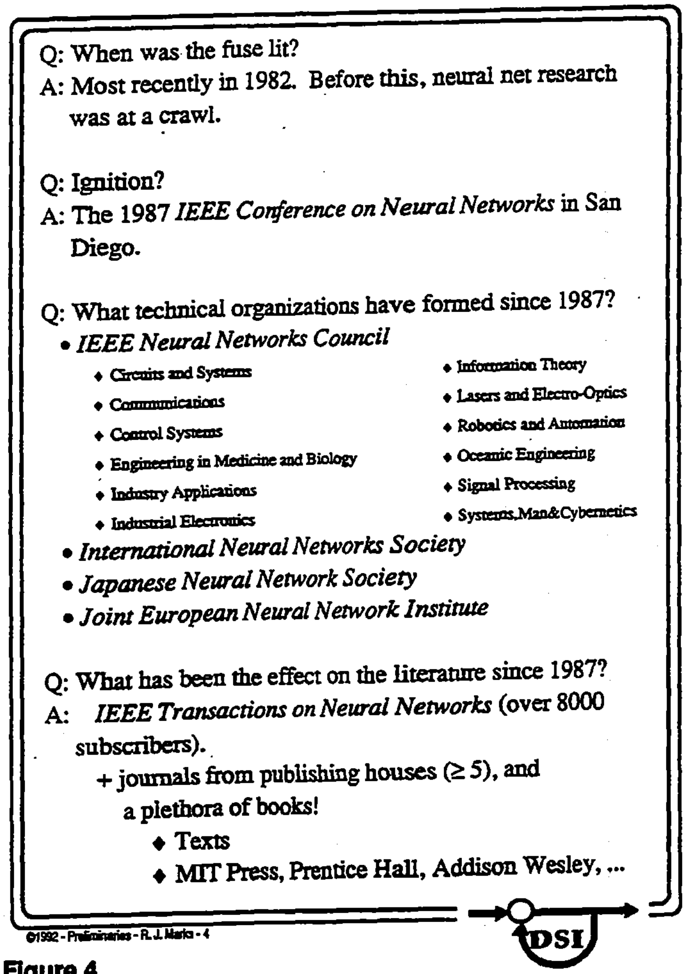
©1992 - Preliminaries - R. J. Marks - 3

Figure 3

Artificial neural networks go by many names. Other fields, which have names that apply to artificial neural networks in the generic sense, are fields that are, in fact and practice, quite different than artificial neural networks

Notes Notes Notes

Notes Notes Notes



Q: When was the fuse lit?

A: Most recently in 1982. Before this, neural net research was at a crawl.

Q: Ignition?

A: The 1987 *IEEE Conference on Neural Networks* in San Diego.

Q: What technical organizations have formed since 1987?

- *IEEE Neural Networks Council*
  - ♦ Circuits and Systems
  - ♦ Communications
  - ♦ Control Systems
  - ♦ Engineering in Medicine and Biology
  - ♦ Industry Applications
  - ♦ Industrial Electronics
  - ♦ Information Theory
  - ♦ Lasers and Electro-Optics
  - ♦ Robotics and Automation
  - ♦ Oceanic Engineering
  - ♦ Signal Processing
  - ♦ Systems,Man&Cybernetics
- *International Neural Networks Society*
- *Japanese Neural Network Society*
- *Joint European Neural Network Institute*

Q: What has been the effect on the literature since 1987?

A: *IEEE Transactions on Neural Networks* (over 8000 subscribers).

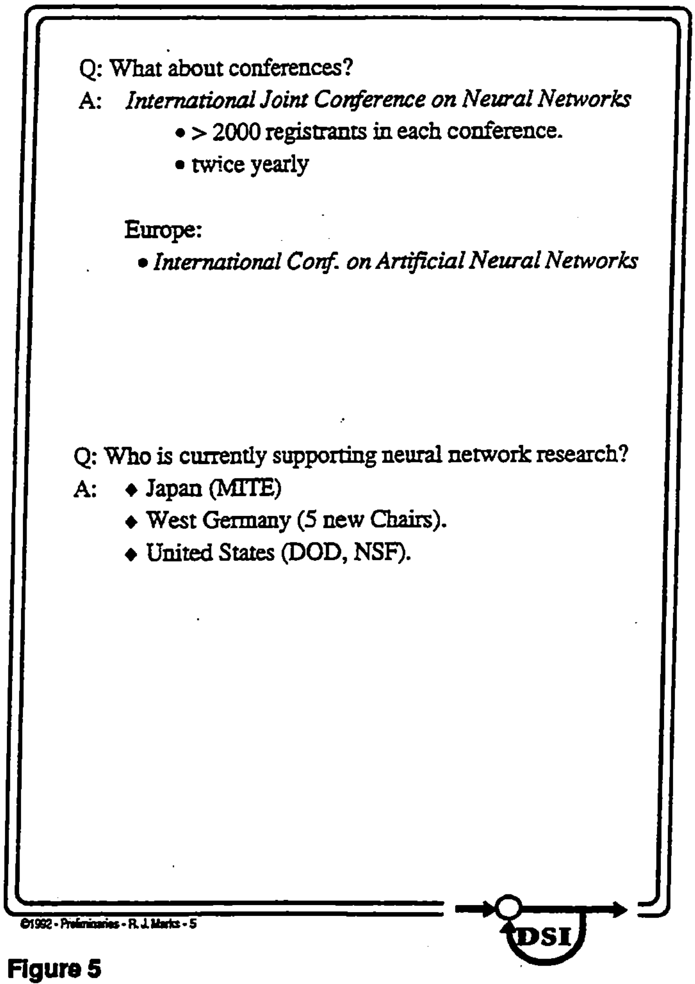
+ journals from publishing houses (≥ 5), and a plethora of books!

- ♦ Texts
- ♦ MIT Press, Prentice Hall, Addison Wesley, ...

©1992 - Preliminaries - R. J. Marks - 4

DSI

Figure 4



Q: What about conferences?

A: *International Joint Conference on Neural Networks*

- > 2000 registrants in each conference.
- twice yearly

Europe:

- *International Conf. on Artificial Neural Networks*

Q: Who is currently supporting neural network research?

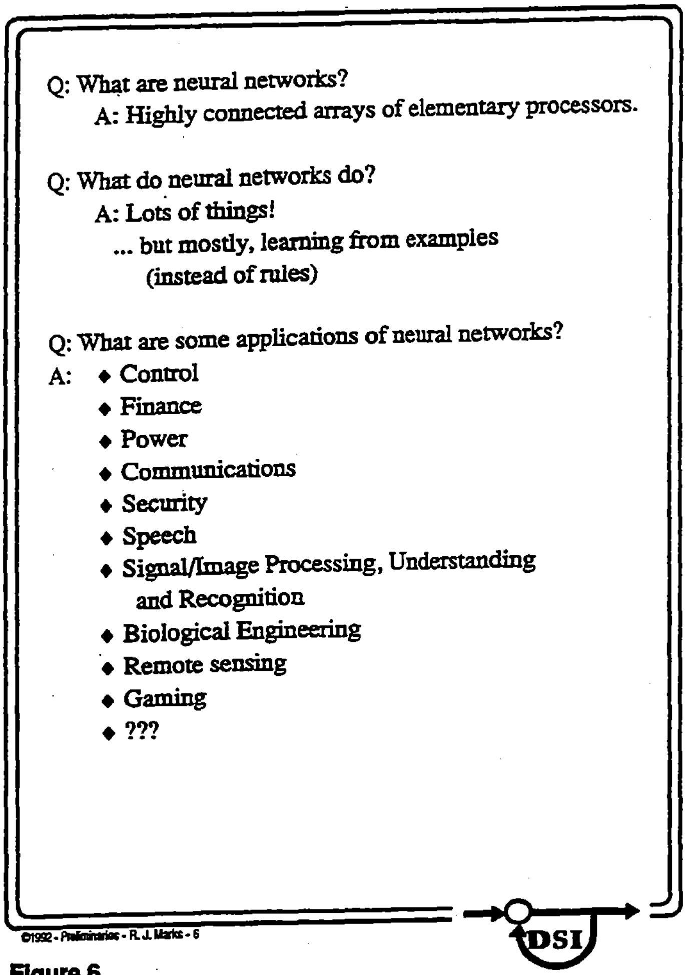
A:
- ◆ Japan (MITE)
- ◆ West Germany (5 new Chairs).
- ◆ United States (DOD, NSF).

©1992 - Preliminaries - R. J. Marks - 5

Figure 5

Notes Notes Notes

Notes Notes Notes



Q: What are neural networks?

A: Highly connected arrays of elementary processors.

Q: What do neural networks do?

A: Lots of things!

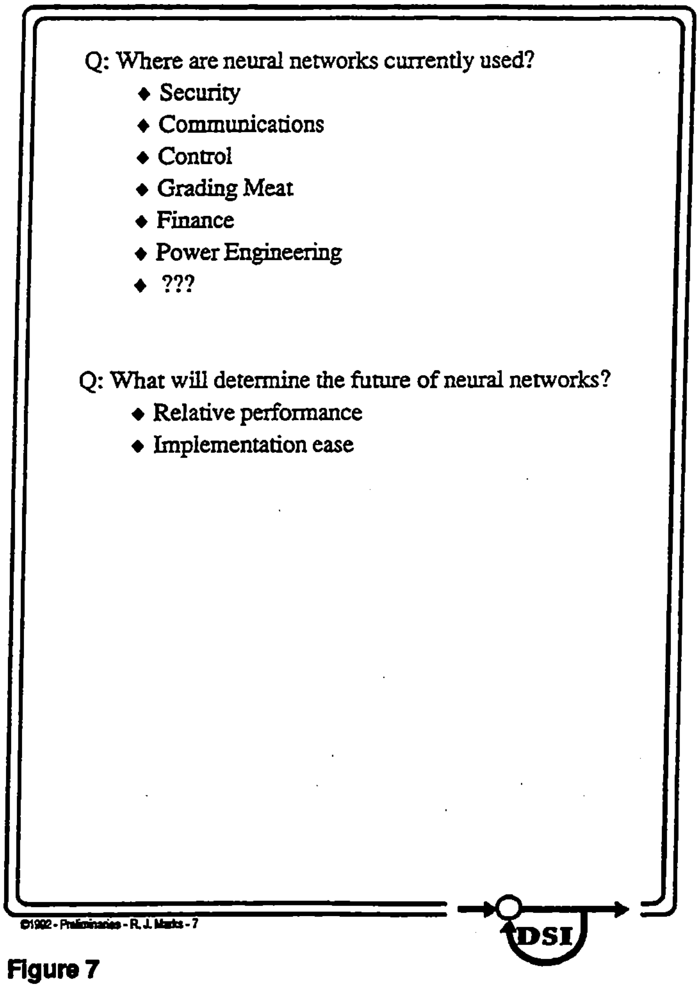
... but mostly, learning from examples
(instead of rules)

Q: What are some applications of neural networks?

A:
- Control
- Finance
- Power
- Communications
- Security
- Speech
- Signal/Image Processing, Understanding and Recognition
- Biological Engineering
- Remote sensing
- Gaming
- ???

©1992 - Preliminaries - R. J. Marks - 6

Figure 6



Q: Where are neural networks currently used?

- Security
- Communications
- Control
- Grading Meat
- Finance
- Power Engineering
- ???

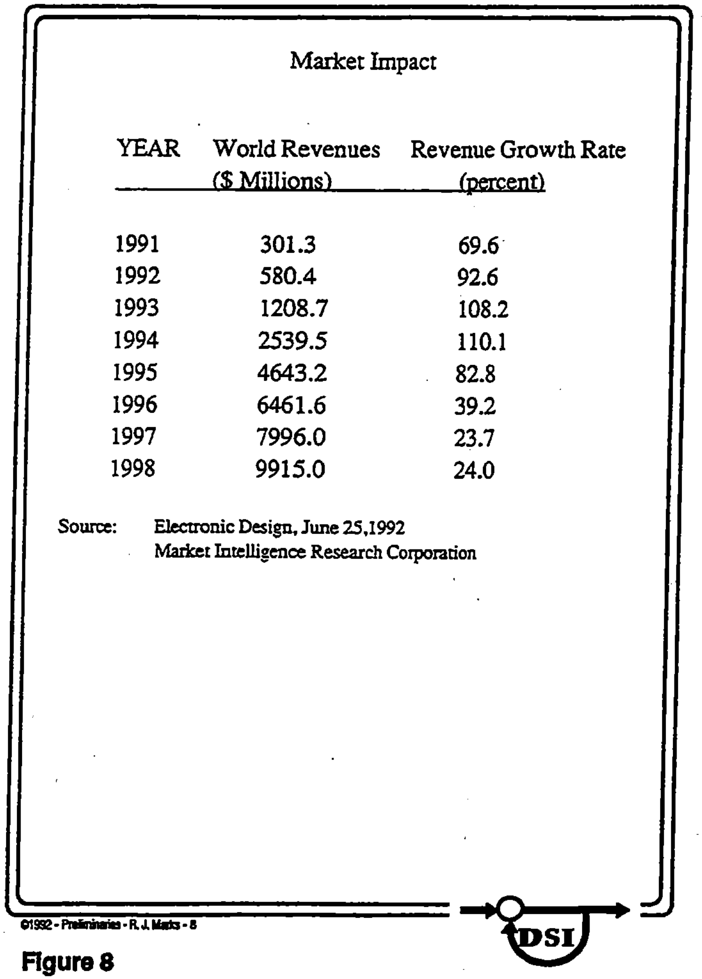
Q: What will determine the future of neural networks?

- Relative performance
- Implementation ease

©1992 - Preliminaries - R. J. Marks - 7

Figure 7

As with many newly emerging technologies, neural networks have been applied to numerous problems, from predicting stock markets to grading meat in slaughter houses. The most promising applications include forecasting, control, and fuzzy systems.



Market Impact

YEAR	World Revenues ($ Millions)	Revenue Growth Rate (percent)
1991	301.3	69.6
1992	580.4	92.6
1993	1208.7	108.2
1994	2539.5	110.1
1995	4643.2	82.8
1996	6461.6	39.2
1997	7996.0	23.7
1998	9915.0	24.0

Source: Electronic Design, June 25,1992
Market Intelligence Research Corporation

©1992 - Preliminaries - R.J. Marks - 8

Figure 8

Neural networks' applications to power engineering are considered so diverse as to warrant individual forums on the topic. The next is scheduled for Japan in 1993.

Scientists create thinking computers to forecast loads

Notes Notes Notes

Working with a team of faculty engineers and graduate students at the University of Washington, Casey Brace, senior engineer, Engineering Applications and Analysis, is developing a neural network computer model that has potential for substantial savings to Puget Power.

This neural network predicts short-term loads to help Puget Power's power scheduler, Lloyd Reed, estimate needs as accurately as possible. After only two months of work, the group is ready to try a neural network forecast trial.

Neural networks are a class of mathematical models that mimic the brain. Uses as diverse as loan application analysis and power load security assessment are in the research stages at over 100 companies. Puget Power is one of the first to use neural networks in utility applications.

Puget Power and the UW team have been working together closely on this project, says Brace. "They look to us to know how a utility functions. We've got the knowledge of this business as well as the computer data. The students need to know our requirements so they can translate them into a useful model," says Brace.

"The difference between neural networks and common linear computing is like the difference between learning and memorizing," says Robert J. Marks II, professor of electrical engineering at the University of Washington. The network will need to learn and judge effects of weather, times of year and local events like school vacations and wood burning bans before forecasting the load.

"I need something that can work fast, and get information to me before 10 in the morning," says Reed. "We have an obligation to our customers to meet the load. We must have early indication of what we'll need, especially if the demand will be high. If we need energy beyond what we can supply ourselves, we may need to purchase outside energy before it is bought up by other utilities."

The research team is investigating a neural network application for use at Colstrip as well. The generation site has a transmission disturbance detection, evaluation, and decision-making scheme in place now. When there is a fault followed by an outage, the scheme decides whether or not to trip the generators to protect them against damage. Early detection is the key to maintaining the stability of a power system and better

Notes Notes Notes

protecting generators. Theoretically, a neural network could provide a better detection scheme of transmission system disturbances.

But even the most promising models being developed today don't have the brain-power of a common housefly.

"When researchers reach the level of a fly," says Brace, "science will be making amazing strides. Sure, flies are not intelligent, but they can recognize food and danger. And they can fly. We could do so much with just that level of recognition."

Reprinted with permission of Puget Sound Power & Light Co.

Neural net regulates electric power grid

Notes Notes Notes

Vancouver, B.C. — Neural Systems Inc.'s software has been managing the local power grid here since July. The system has reduced voltage fluctuations by one hundredfold, thanks to its neural simulation software, called Genesis.

Proprietary method

Unlike other neural simulators that require the user to specify the desired result, Genesis uses a proprietary method called "response learning," which can a find a solution from training data even when the desired result is unknown at the outset.

Genesis balances the load on British Columbia Hydro Authority (BC Hydro) by adaptively controlling four synchronous Voltage-Amperes-Reactance (VAR) machines of unequal capacity. The VAR machines are supposed to make the line voltage into one of those ideal voltage generators that is studied in electronics courses — generators hose voltage remains the same no matter how much current is drawn. But real-line voltage must be maintained by reacting to the fluctuating power on the line to smooth out any variations.

VAR machines are giant synchronous electric motors that run constantly but drive nothing. They provide real-time voltage regulation whenever a customer switches a heavy load on or off. The machines also come into play when a utility brings a generator on-line. Any such change sends ripples through the power grid, causing the load to become unbalanced among the various VAR machines feeding the grid. Without redistributing the load, unnecessary losses are incurred.

Real-time operation

Traditionally, these loses are continually corrected by humans who manually balance the load among the generators. Now Genesis handles that task at BC Hydro's Vancouver Island Terminal. The neural network operates in real time, whereas the human operators balanced the load whenever they noticed it was out of balance. The load is measured in mega-VARs — with human operators it is out of balance about 20 MVARs a day, but with the neural network it is only off .2 MVARS, according to Gary Josin, president of Neural Systems.

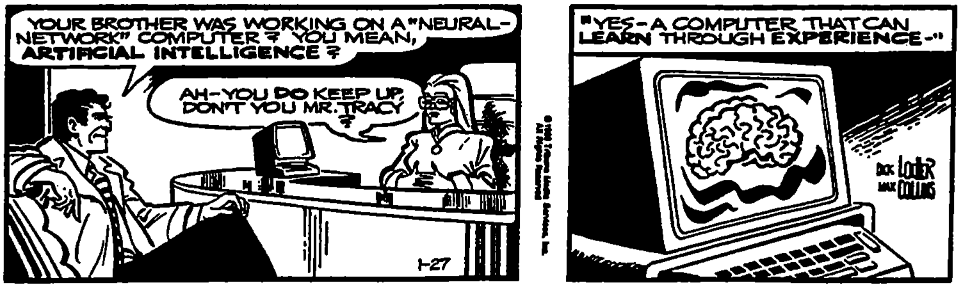
To deploy the application, Genesis was installed on a shop floor PC that acquires data directly from the power grid and outputs control signals to the VAR machines. Eventually it will be deployed in a dedicated controller, but for now it is still running on the PC. "After proving that it worked, we tried to take the PC-based system away while we put the software into a dedicated system, but the operators wouldn't let us. They said they needed it now," revealed Geoff Neily, protection and control supervisor.

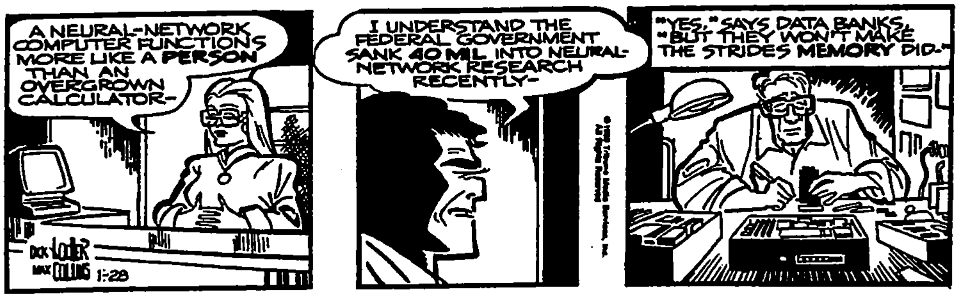
Now that Neural Systems has acquired expertise in power systems, it has identified several problem areas where a neural network might be able to do what traditional controllers have been unable to accomplish. For example, vibrations are introduced into the power spectrum whenever a large load is switched on or off. These vibrations can cause oscillations at several resonant frequencies. "Most of these are below 2 Hz and are damped out by an analog controller, but it is not totally effective," Neily explained. Genesis, though, could set up to adaptively change the frequency of the damping circuitry to increase the system's effectiveness.

—R. Colin Johnson

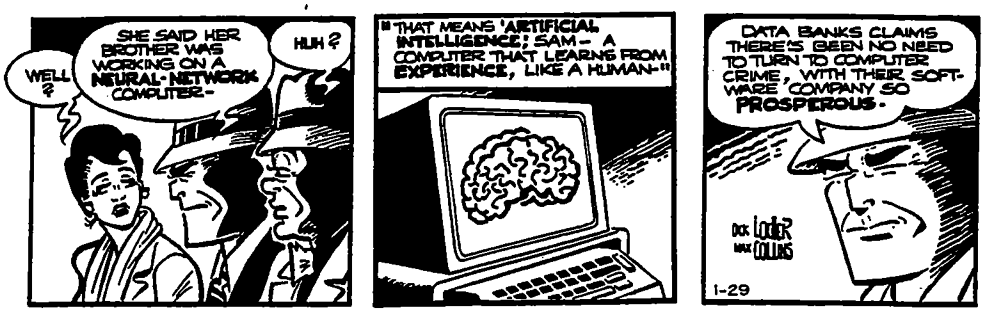
DICK TRACY Dick Locher and Max Collins

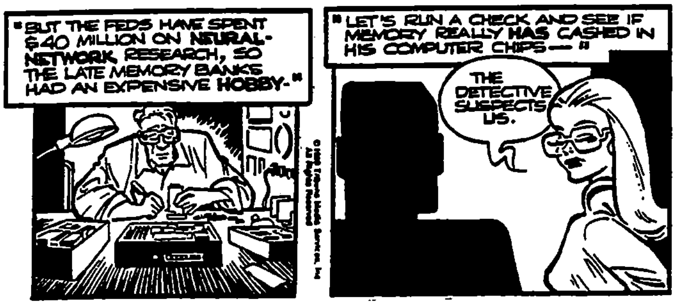
Notes Notes Notes


YOUR BROTHER WAS WORKING ON A "NEURAL-NETWORK" COMPUTER ? YOU MEAN, ARTIFICIAL INTELLIGENCE ?
AH-YOU DO KEEP UP DON'T YOU MR. TRACY ?
1-27
"YES-A COMPUTER THAT CAN LEARN THROUGH EXPERIENCE-"
DICK LOCHER MAX COLLINS


A NEURAL-NETWORK COMPUTER FUNCTIONS MORE LIKE A PERSON THAN AN OVERGROWN CALCULATOR-
DICK LOCHER MAX COLLINS 1-28
I UNDERSTAND THE FEDERAL GOVERNMENT SANK 40 MIL INTO NEURAL-NETWORK RESEARCH RECENTLY-
"YES," SAYS DATA BANKS, "BUT THEY WON'T MAKE THE STRIDES MEMORY DID-"

THANK YOU FOR YOUR TIME, MS. BANKS-SORRY TO HEAR OF YOUR BROTHER'S DEATH.
THAT'S KIND OF YOU, MR. TRACY-NO OFFENSE MEANT WHERE HATCHET-JAW IS CONCERNED.
"NONE TAKEN," SAYS TRACY.
HATCHET-JAW


WELL ?
SHE SAID HER BROTHER WAS WORKING ON A NEURAL-NETWORK COMPUTER-
HUH ?
"THAT MEANS 'ARTIFICIAL INTELLIGENCE,' SAM- A COMPUTER THAT LEARNS FROM EXPERIENCE, LIKE A HUMAN-"
DATA BANKS CLAIMS THERE'S BEEN NO NEED TO TURN TO COMPUTER CRIME, WITH THEIR SOFTWARE COMPANY SO PROSPEROUS.
DICK LOCHER MAX COLLINS
1-29


"BUT THE FEDS HAVE SPENT $40 MILLION ON NEURAL-NETWORK RESEARCH, SO THE LATE MEMORY BANKS HAD AN EXPENSIVE HOBBY-"
"LET'S RUN A CHECK AND SEE IF MEMORY REALLY HAS CASHED IN HIS COMPUTER CHIPS —"
THE DETECTIVE SUSPECTS US.

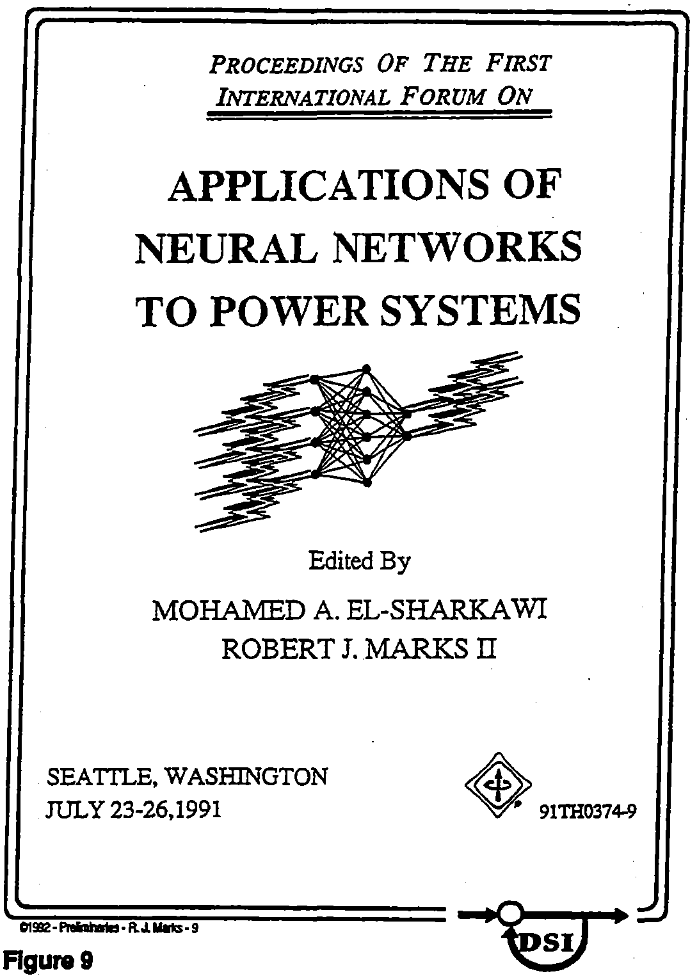
Reprinted by permission:

Tribune Media Services

Permission valid until

May 1, 1993

Notes Notes Notes



Proceedings Of The First International Forum On

APPLICATIONS OF NEURAL NETWORKS TO POWER SYSTEMS

Edited By

MOHAMED A. EL-SHARKAWI
ROBERT J. MARKS II

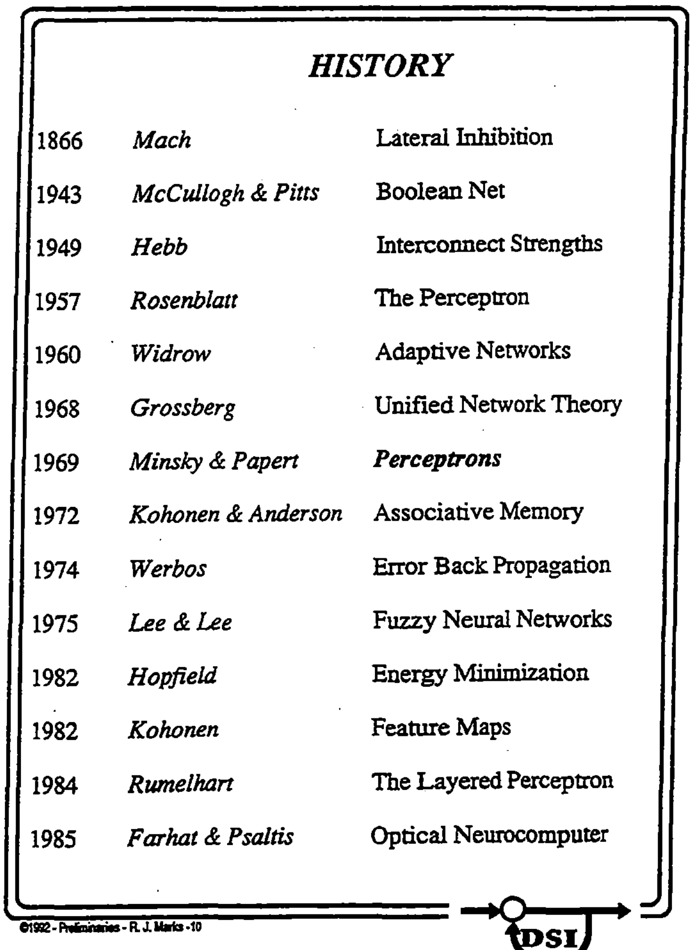
SEATTLE, WASHINGTON
JULY 23-26,1991

91TH0374-9

©1992 - Preliminaries - R.J. Marks - 9

DSI

Figure 9



HISTORY

1866	*Mach*	Lateral Inhibition
1943	*McCullogh & Pitts*	Boolean Net
1949	*Hebb*	Interconnect Strengths
1957	*Rosenblatt*	The Perceptron
1960	*Widrow*	Adaptive Networks
1968	*Grossberg*	Unified Network Theory
1969	*Minsky & Papert*	***Perceptrons***
1972	*Kohonen & Anderson*	Associative Memory
1974	*Werbos*	Error Back Propagation
1975	*Lee & Lee*	Fuzzy Neural Networks
1982	*Hopfield*	Energy Minimization
1982	*Kohonen*	Feature Maps
1984	*Rumelhart*	The Layered Perceptron
1985	*Farhat & Psaltis*	Optical Neurocomputer

©1992 - Preliminaries - R. J. Marks - 10

DSI

Figure 10

Here are some of the key events in the development of neural networks. In each case, the neuron, either artificial or biological, was modeled mathematically.

Mach is the same person for whom the speed of sound is named. Mach showed how lateral inhibition among neurons could account for the optical illusion now known as Mach bands.

Hebb demonstrated what is now known as Hebbian learning. The more an interconnect between neurons is used, the stronger it grows.

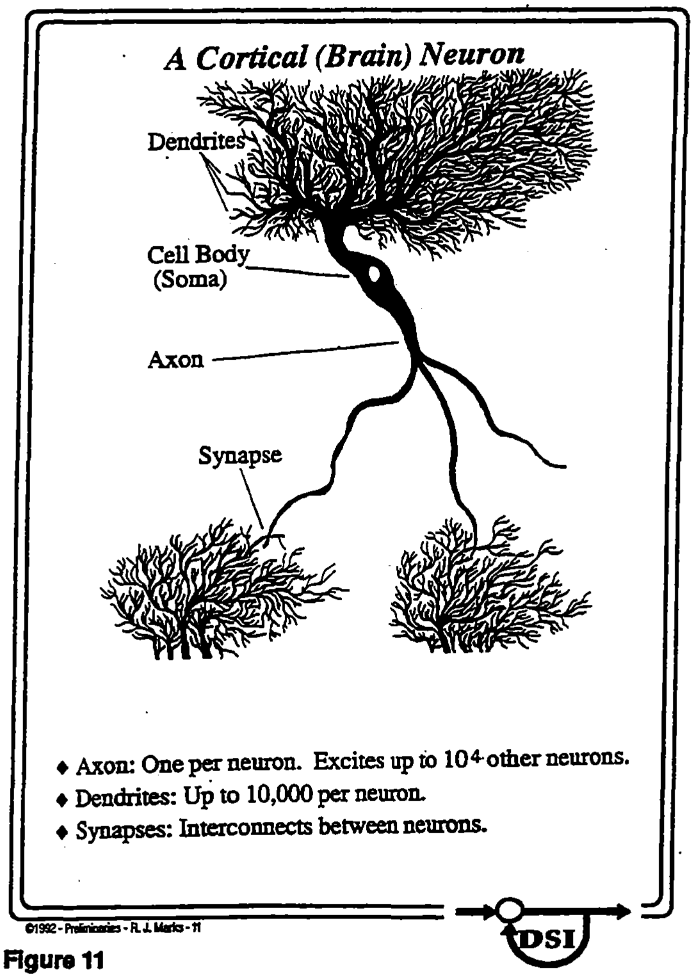
Rosenblatt and Widrow both investigated a version of the perceptron. Grossberg proposes his unified network theory of neural networks.

The book ***Perceptrons*** illustrated some severe limitations of the perceptron. Many of the important limitations were overcome in the work of Rumelhart. Backpropagation training of neural networks was pro-

Notes Notes Notes

posed by Rumelhart. Backpropagation was independently discovered by Werbos and a number of other researchers.

Neural networks today are recognized as natural complements to fuzzy systems. Lee and Lee first made the connection in 1975.



A Cortical (Brain) Neuron

- ♦ Axon: One per neuron. Excites up to 10^4 other neurons.
- ♦ Dendrites: Up to 10,000 per neuron.
- ♦ Synapses: Interconnects between neurons.

©1992 - Preliminaries - R. J. Marks - 11

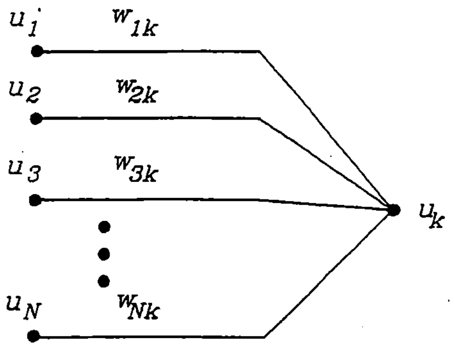
Figure 11

The architecture of artificial neural networks is based loosely on that of the biological neural network. The *axon* is the output of the neuron and connects to other neurons. The *dendrites* provide the input to the neuron. The *synapse* is the interconnect between two neurons and is analogous to the *weights* in the artificial neural network. Biological neuron types are many and varied.

Notes Notes Notes

THE NEURON: An Elementary Processor

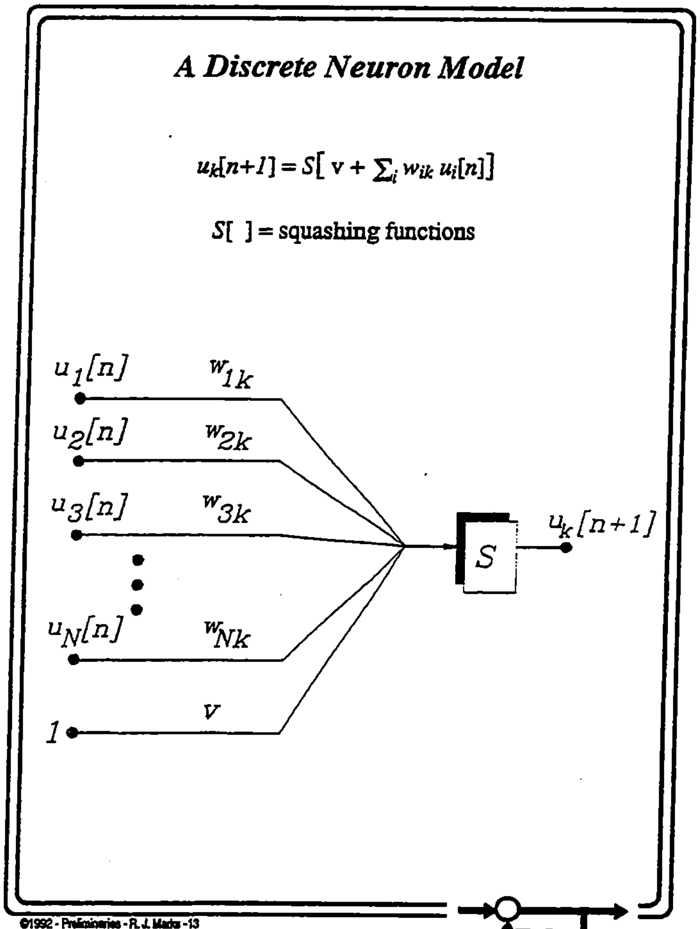
- ♦ ***Neuron (nodes or neurodes):*** An elementary processor which, typcally, sums its inputs, and performs a nonlinear operation on this sum. The result is the the neuron's state. The nonlinear function is referred to as a *sigmoid* nonlinearity or a *squashing function.*
- ♦ ***State:*** Each neoron has a *state*. The state changes with time. As illustrated below, the state of the *k*th neuron is $u_k(t)$. A state is simply a number associated with the neuron. The state is determined by the other neurons to which it is connected.
- ♦ ***Weight:*** The strength by which one neuron is connected to another is specified by the connecting *weight*. The weight between neurons *n* to *k* is denoted by w_{nk}.



©1992 - Preliminaries - R. J. Marks - 12

Figure 12

Here, we define and illustrate the fundamental terminology of neural networks.



A Discrete Neuron Model

$$u_k[n+1] = S[\,\text{v} + \textstyle\sum_i w_{ik}\, u_i[n]]$$

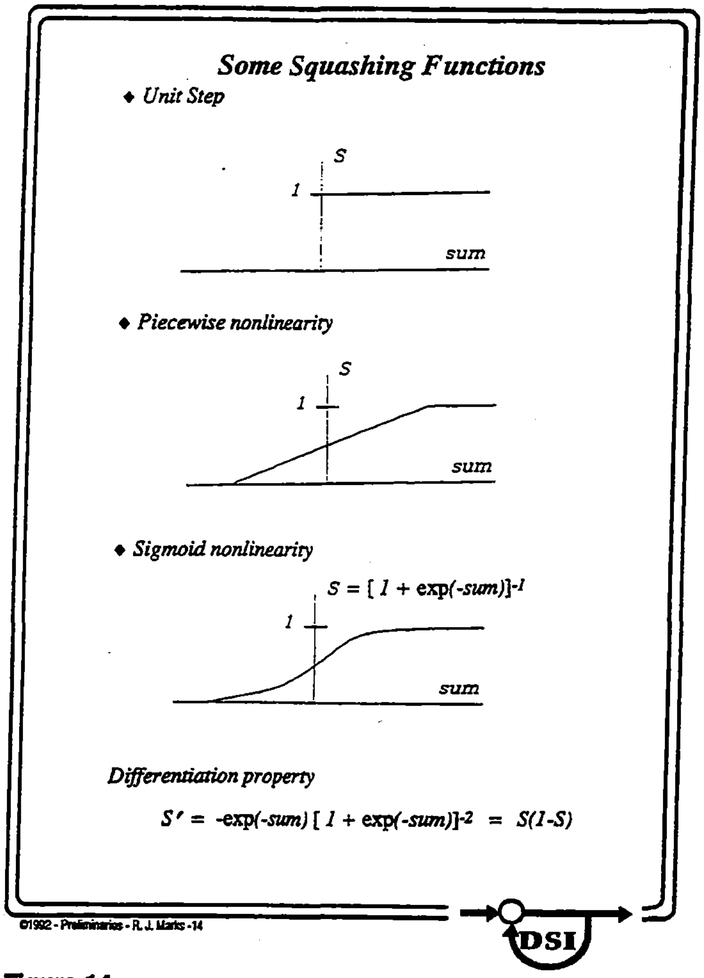
$S[\ ]$ = squashing functions

©1992 - Preliminaries - R. J. Marks - 13

Figure 13

Notes Notes Notes

For the discrete neuron model shown here, *n* parameterizes discrete time. The state of the neuron at time *n* + 1 is equal to a non-linear function of the weighted sum of the other neural states at time *n*.



Some Squashing Functions

- *Unit Step*

- *Piecewise nonlinearity*

- *Sigmoid nonlinearity*

$S = [\,1 + \exp(-sum)]^{-1}$

Differentiation property

$$S' = -\exp(-sum)\,[\,1 + \exp(-sum)]^{-2} = S(1-S)$$

©1992 - Preliminaries - R. J. Marks -14

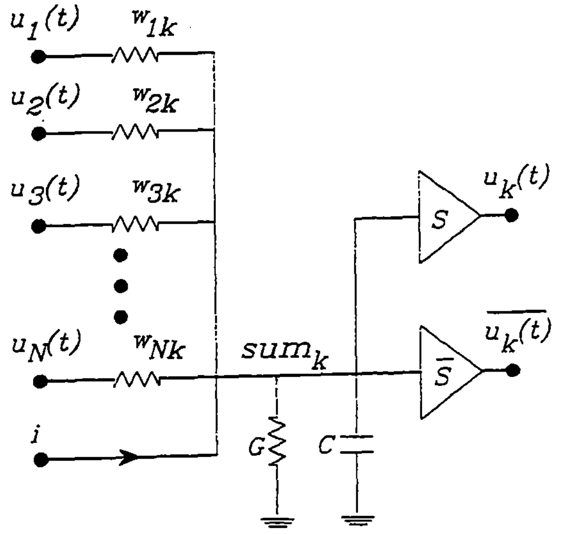
Figure 14

Shown are two popularly used squashing functions used as the neural non-linearities. The choice of the sigmoid non-linearity is motivated by the ease by which its derivative is computed. This derivative is needed in certain training algorithms. Similar relations hold for other functions. If, for example, $S(x) = [\tanh(x) + 1]/2$, then $S' = -2S(S-1)$.

An Analog Neuron Model

$C\ \mathrm{sum}_k'(t) = -\sum_n w_{nk} [\ \mathrm{sum}_k(t) - u_n(t)\ ] - G\ \mathrm{sum}_k(t) + i$

$u_n(t) = S[\ \mathrm{sum}_k(t)\ ]$ $\qquad S[\cdot]$ = squashing function

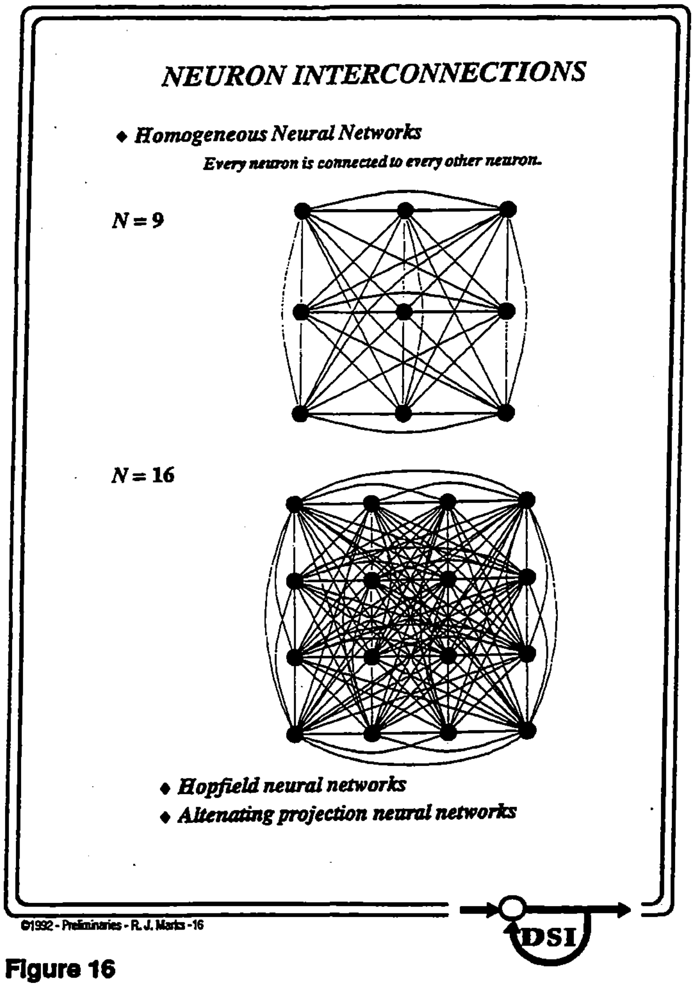


Steady state solution:

$u_k(t) = S[\ c_1 + c_2 \sum_n w_{nk}\ u_n(t)\ ]$

Figure 15

This is a model of a single continuous time (analog) neuron. It has the indices of the kth neuron. In steady state, the neural state, $u_k(t)$, is a non-linear function of the sum of the inputs. The numbers c_1 and c_2 are constants.



NEURON INTERCONNECTIONS

- ***Homogeneous Neural Networks***
  Every neuron is connected to every other neuron.

$N = 9$

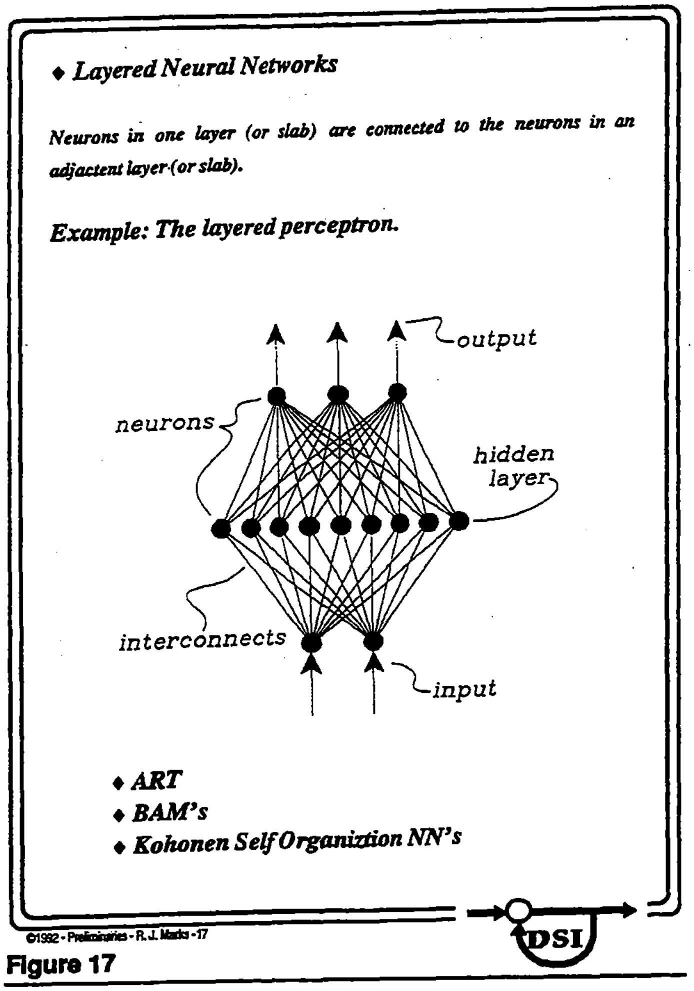
$N = 16$

- ***Hopfield neural networks***
- ***Alternating projection neural networks***

©1992 - Preliminaries - R. J. Marks - 16

Figure 16

Most neural networks are either homogeneous or layered. In homogeneous neural networks, every neuron is connected to every other neuron. For N neurons, there are N^2 interconnections.



♦ *Layered Neural Networks*

Neurons in one layer (or slab) are connected to the neurons in an adjactent layer (or slab).

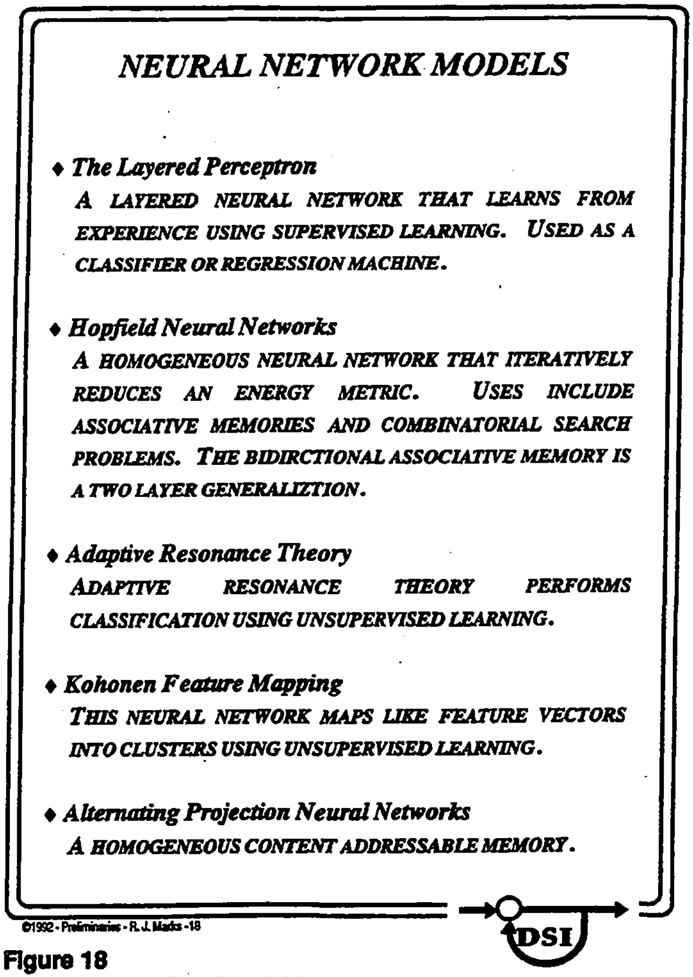
Example: The layered perceptron.

- ♦ *ART*
- ♦ *BAM's*
- ♦ *Kohonen Self Organiziion NN's*

©1992 - Preliminaries - R. J. Marks - 17

Figure 17

In layered neural networks, each layer of neurons typically performs a function which can be different than that of a different layer. *ART* means *adaptive resonance theory* and *BAM* means *bidirectional associative memory.*



NEURAL NETWORK MODELS

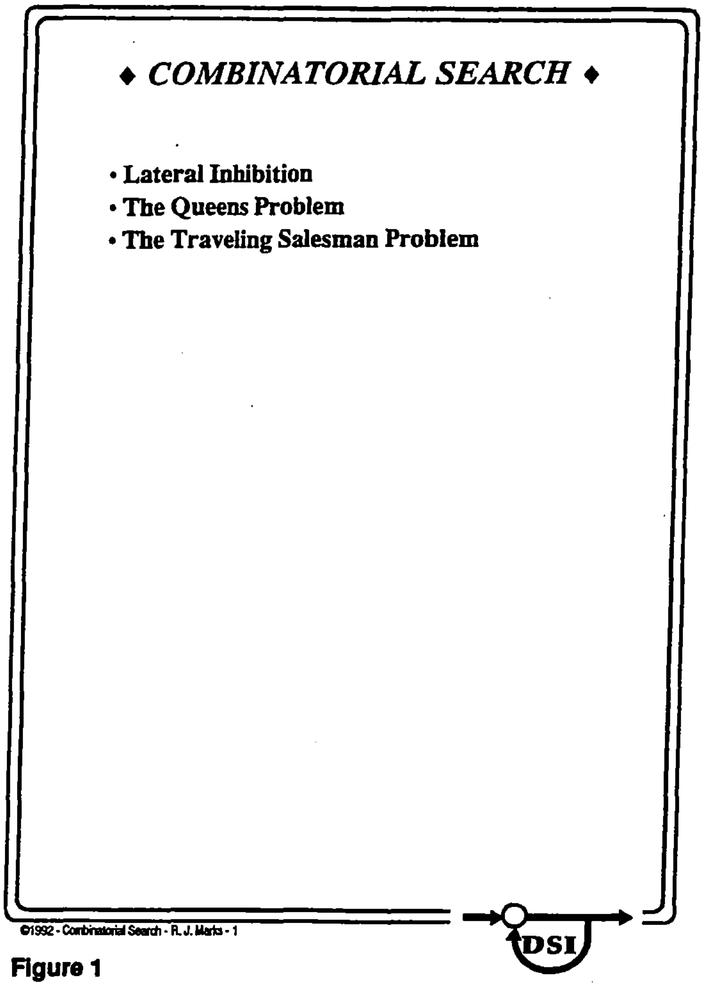
- ***The Layered Perceptron***
  A layered neural network that learns from experience using supervised learning. Used as a classifier or regression machine.
- ***Hopfield Neural Networks***
  A homogeneous neural network that iteratively reduces an energy metric. Uses include associative memories and combinatorial search problems. The bidirctional associative memory is a two layer generaliztion.
- ***Adaptive Resonance Theory***
  Adaptive resonance theory performs classification using unsupervised learning.
- ***Kohonen Feature Mapping***
  This neural network maps like feature vectors into clusters using unsupervised learning.
- ***Alternating Projection Neural Networks***
  A homogeneous content addressable memory.

©1992 - Preliminaries - R. J. Marks - 18

Figure 18

These are the four most commonly used artificial neural network models.

Combinatorial Search



♦ *COMBINATORIAL SEARCH* ♦

- Lateral Inhibition
- The Queens Problem
- The Traveling Salesman Problem

Figure 1

Notes Notes Notes

Notes Notes Notes

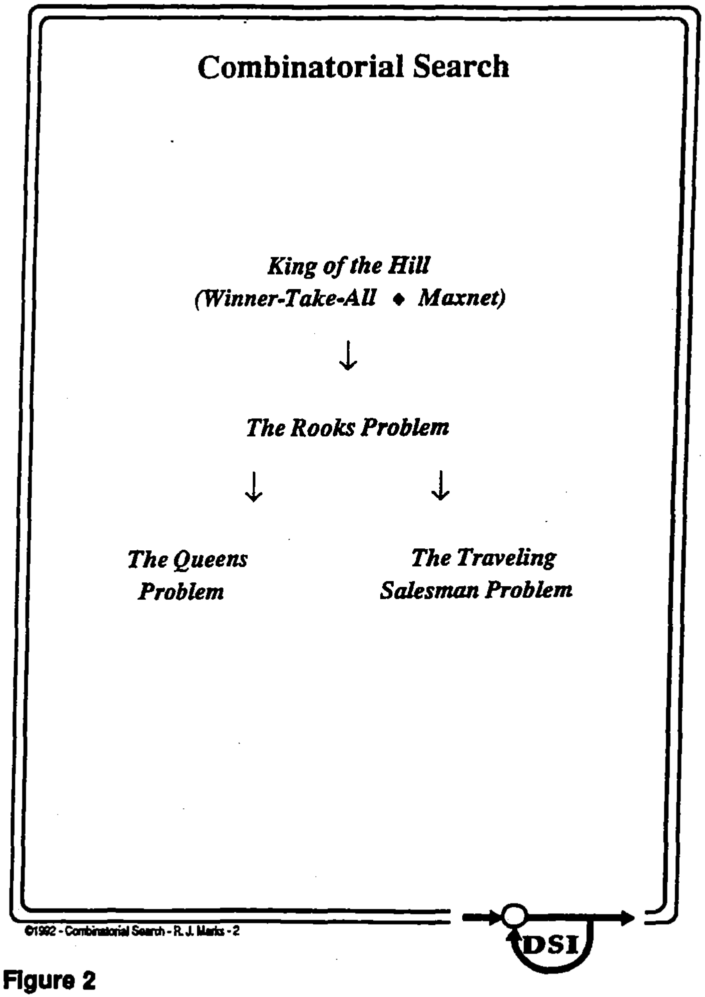
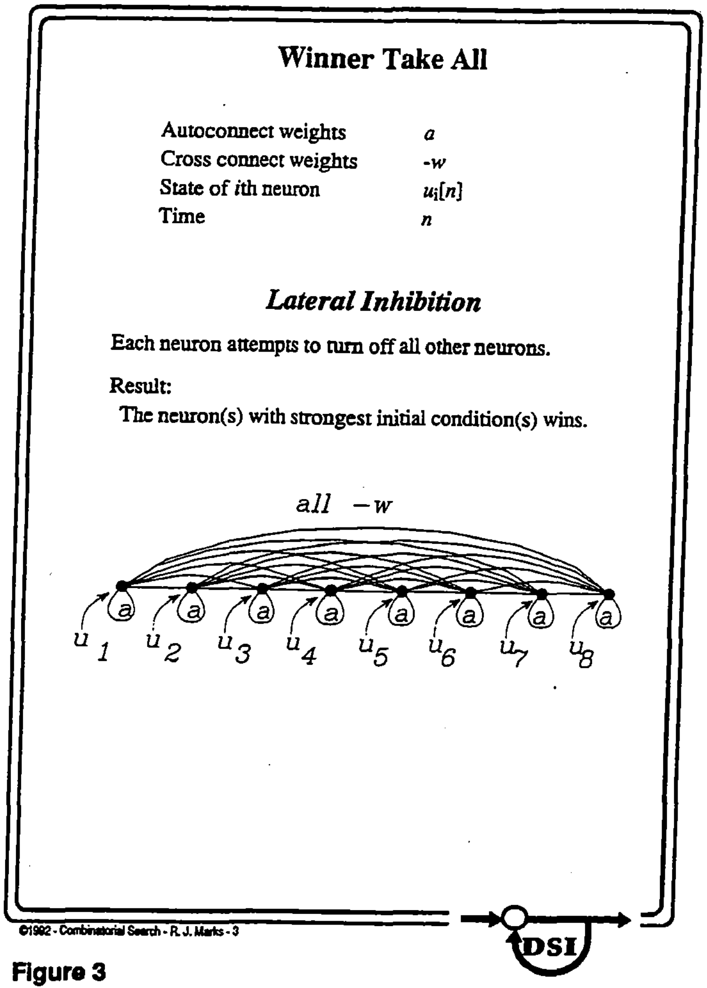


Figure 2

The king-of-the-hill problem will be generalized to solve the simple combinatorial search required by the Rooks Problem. The rooks problem will be shown to straightforwardly generalize into solution of the Queens and the Traveling Salesman problems.



Winner Take All

Autoconnect weights	a
Cross connect weights	$-w$
State of ith neuron	$u_i[n]$
Time	n

Lateral Inhibition

Each neuron attempts to turn off all other neurons.

Result:
The neuron(s) with strongest initial condition(s) wins.

Figure 3

In the King-of-the-Hill (or Winner-Take-All or Maxnet) neural network, each neuron attempts to 'turn off' all other neurons while reinforcing itself. When the contest is over, the strongest neuron or neurons win with a numerically larger state than the loosing neurons.

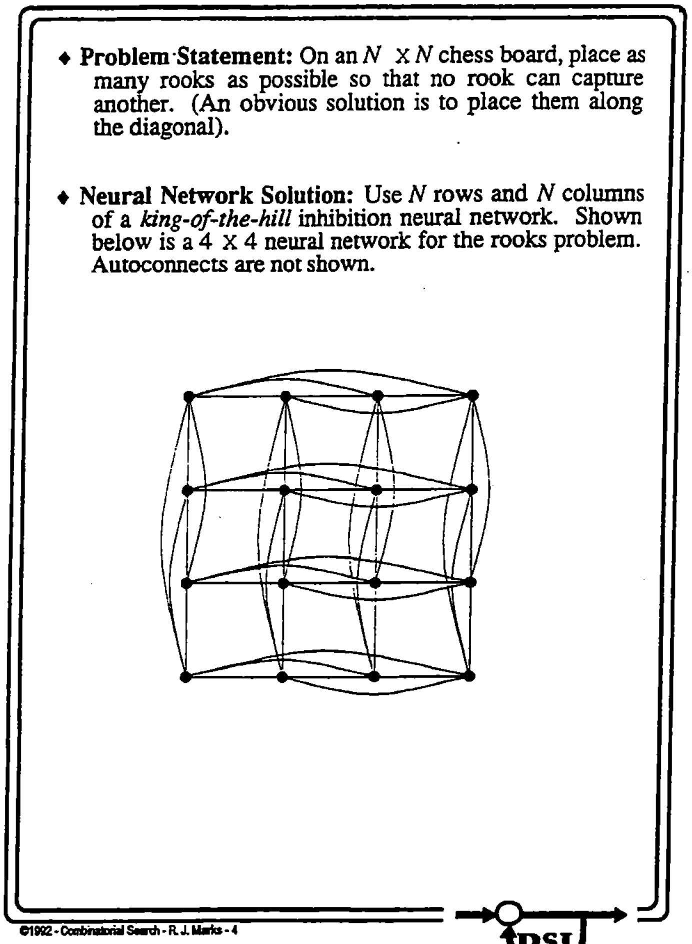
Specifically, consider the linear array of neurons illustrated here. The interconnect weights between all of the neurons is $-w$ and the autoconnection of a neuron to itself will be denoted as a. We will assume both w and a are positive. Typically, a is much larger than w.

An inspection of the above equations reveals the dynamics of the competitive nature of this simple neural network. As an example, the student is invited to try a simple 3 neuron example with w=0.1 and a=1.1. For initial states, [0.9,0.5,0.1], convergence occurs in less that ten iterations of each neuron.

Notes Notes Notes

For obvious reasons, such neural networks are referred to as *winner take all* nets. They have also been referred to as *maxnets* and *king of the hill* neural networks. Note that we can view the operation of finding a maximum a simple search problem.

Notes Notes Notes



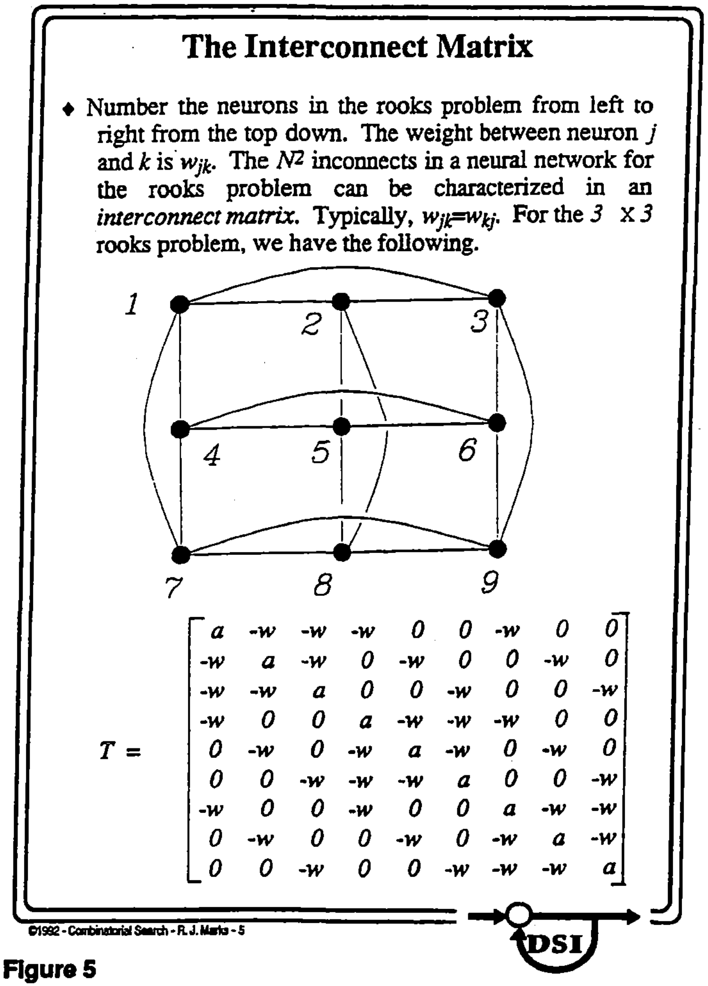
- **Problem Statement:** On an $N \times N$ chess board, place as many rooks as possible so that no rook can capture another. (An obvious solution is to place them along the diagonal).
- **Neural Network Solution:** Use N rows and N columns of a *king-of-the-hill* inhibition neural network. Shown below is a 4×4 neural network for the rooks problem. Autoconnects are not shown.

©1992 - Combinatorial Search - R. J. Marks - 4

Figure 4

A simple combinatorial search problem is the rooks problem. On an $N \times N$ chess board, we wish to place as many rooks as possible so that no rook can capture another. The maximum number of rooks that can be thus placed is N. One clear solution is to place N rooks on the diagonals. Although the rooks problem is simple, its discussion allows easy conceptualization to the more complicated Queens and Traveling Salesman problems.

To solve the Rooks problem, we form an $N \times N$ array of neurons. Each row of N neurons will be connected in a winner-take-all configuration. Also, each column is connected in a winner-take-all configuration. Our aim is to require the $N \times N$ net to settle onto a solution that has, in steady state, only one neuron at a high state for each row and each column. The result is clearly a solution to the Rooks problem. The initial states of the N^2 neurons can be chosen randomly.



The Interconnect Matrix

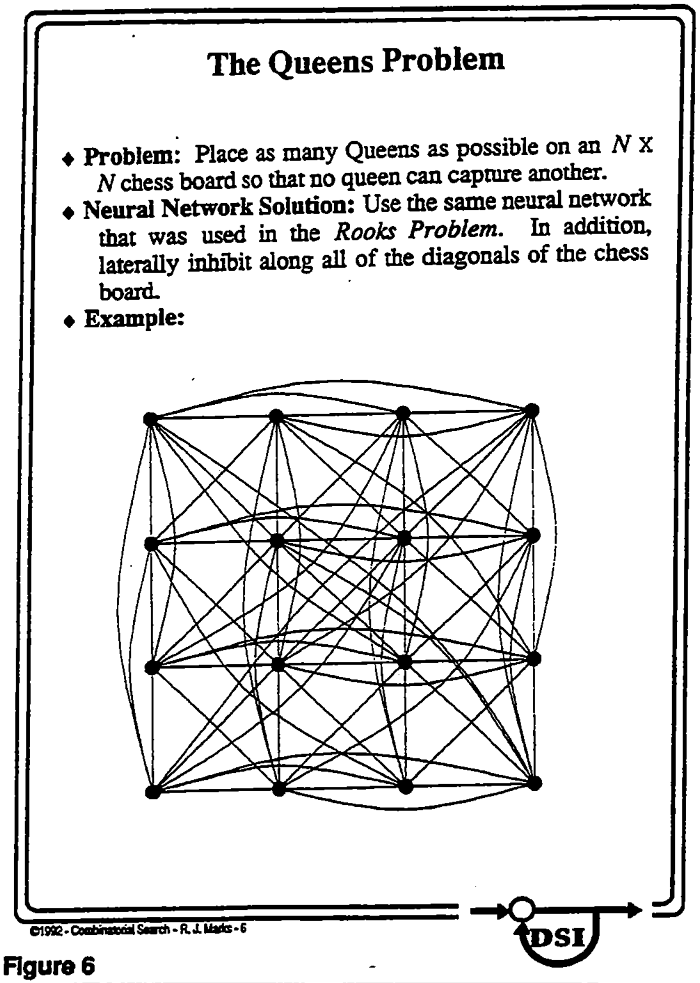
- Number the neurons in the rooks problem from left to right from the top down. The weight between neuron *j* and *k* is w_{jk}. The N^2 inconnects in a neural network for the rooks problem can be characterized in an *interconnect matrix*. Typically, $w_{jk}=w_{kj}$. For the *3* x *3* rooks problem, we have the following.

$$T = \begin{bmatrix} a & -w & -w & -w & 0 & 0 & -w & 0 & 0 \\ -w & a & -w & 0 & -w & 0 & 0 & -w & 0 \\ -w & -w & a & 0 & 0 & -w & 0 & 0 & -w \\ -w & 0 & 0 & a & -w & -w & -w & 0 & 0 \\ 0 & -w & 0 & -w & a & -w & 0 & -w & 0 \\ 0 & 0 & -w & -w & -w & a & 0 & 0 & -w \\ -w & 0 & 0 & -w & 0 & 0 & a & -w & -w \\ 0 & -w & 0 & 0 & -w & 0 & -w & a & -w \\ 0 & 0 & -w & 0 & 0 & -w & -w & -w & a \end{bmatrix}$$

©1992 - Combinatorial Search - R. J. Marks - 5

Figure 5

The interconnect matrix is a table of the interconnect weights between neuron pairs.



The Queens Problem

- **Problem:** Place as many Queens as possible on an N x N chess board so that no queen can capture another.
- **Neural Network Solution:** Use the same neural network that was used in the *Rooks Problem.* In addition, laterally inhibit along all of the diagonals of the chess board.
- **Example:**

©1992 - Combinatorial Search - R. J. Marks - 6

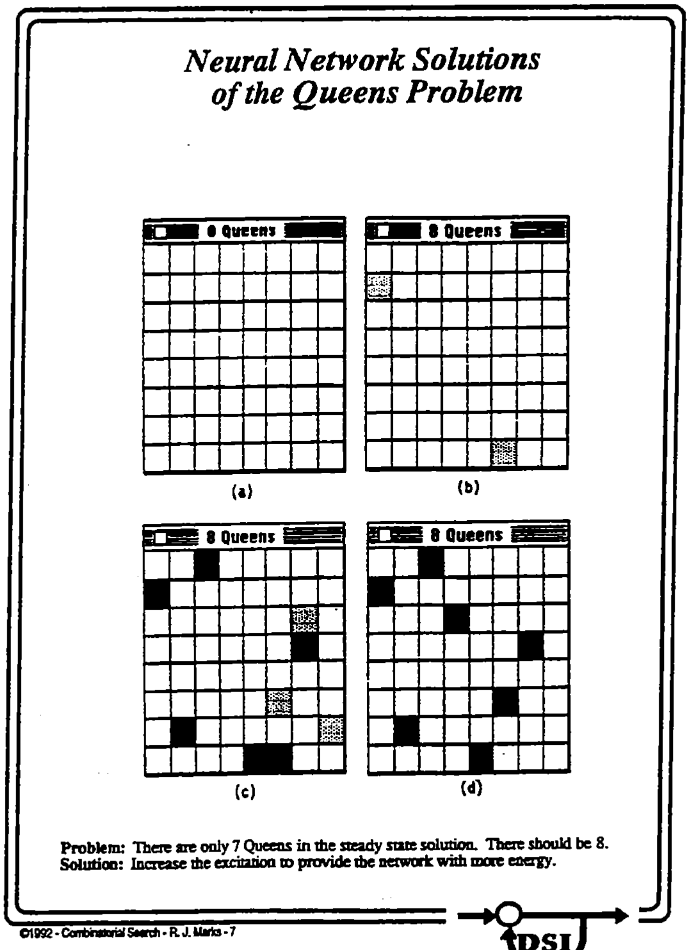
DSI

Figure 6

The Queens problem is analogous to the Rooks problem, except that queens, rather than rooks, are used. We must now provide, in addition, winner-take-all neural networks along each diagonal. If two neurons are connected by weights from two different winner-take-all nets, the composite weight is just the sum of the components.

We illustrate the working of the Queens neural network by borrowing results from McDonnell *et.al.* The neural net was randomly initialized.

If $N > 3$, a total of N Queens can be placed on a chess board so that no queen can capture another.



Neural Network Solutions of the Queens Problem

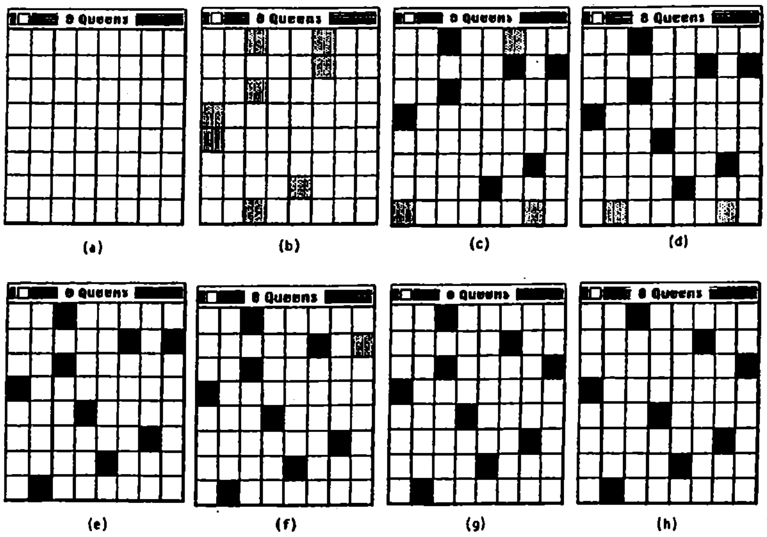
Problem: There are only 7 Queens in the steady state solution. There should be 8.
Solution: Increase the excitation to provide the network with more energy.

©1992 - Combinatorial Search - R. J. Marks - 7

Figure 7

These are snapshots of a simulation of the analog solution of the queens problem on a standard chess board. Initialization was random. Each pixel is a neuron. A clear pixel corresponds to a zero. A darkened pixel corresponds to a value of one. Shaded pixels correspond to intermediate values. The excitation was for a current of i = 0.15. The final solution, although valid, is one queen shy of the maximum number of eight queens. The current needs to be increased a bit.

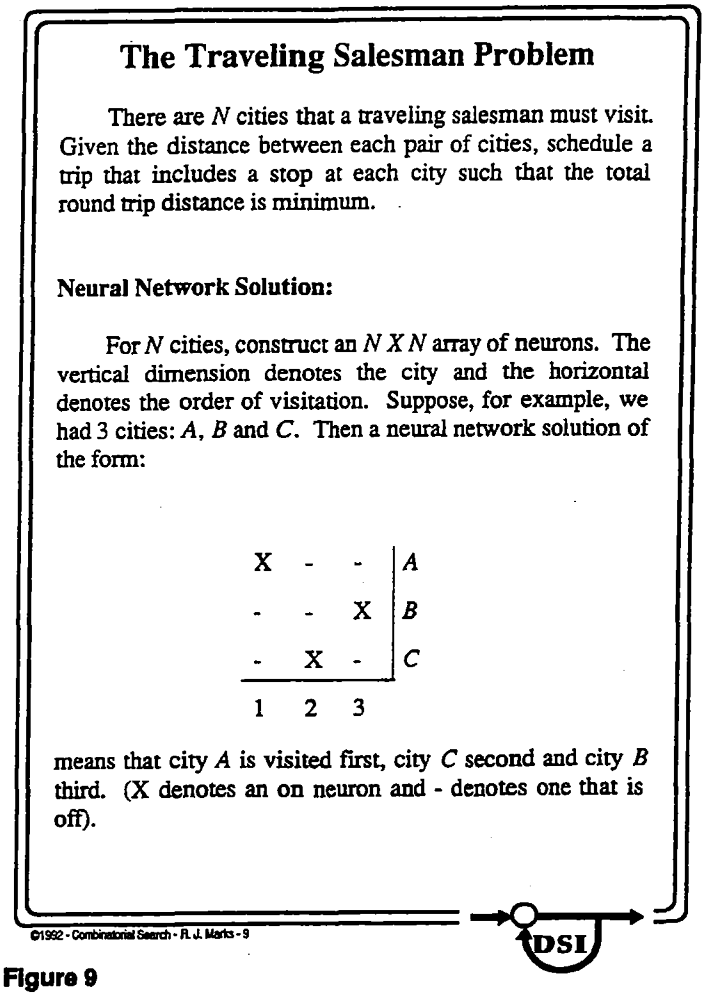
Another Neural Network Solution



©1992 - Combinatorial Search - R. J. Marks - 8

Figure 8

Here, the neural excitation has been increased to i = 0.25. Interestingly, in (g), there are two 'on' neurons in the third column, each trying to turn the other off. Which one wins? Because there are also two 'on' neurons in the third row, the neuron in the third row and third column is simply outnumbered. It looses.



The Traveling Salesman Problem

There are N cities that a traveling salesman must visit. Given the distance between each pair of cities, schedule a trip that includes a stop at each city such that the total round trip distance is minimum.

Neural Network Solution:

For N cities, construct an $N X N$ array of neurons. The vertical dimension denotes the city and the horizontal denotes the order of visitation. Suppose, for example, we had 3 cities: A, B and C. Then a neural network solution of the form:

$$\begin{array}{ccc|c} \text{X} & - & - & A \\ - & - & \text{X} & B \\ - & \text{X} & - & C \\ \hline 1 & 2 & 3 & \end{array}$$

means that city A is visited first, city C second and city B third. (X denotes an on neuron and - denotes one that is off).

©1992 - Combinatorial Search - R. J. Marks - 9

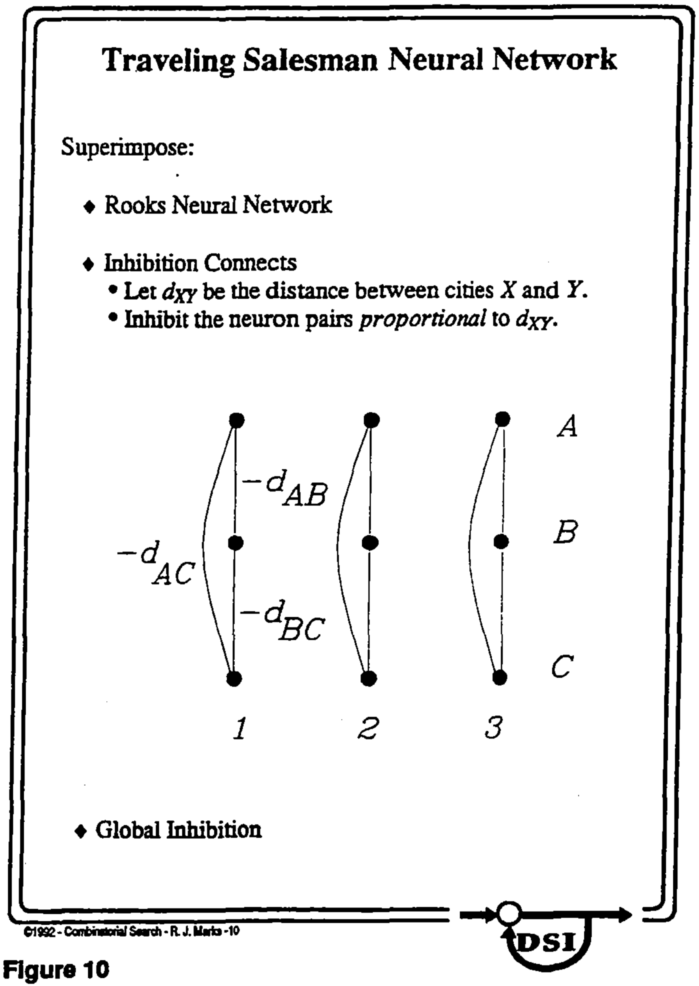
DSI

Figure 9

The Traveling Salesman problem can also be viewed as an extension of the Rooks problem. We have, say, N cities denoted by **A, B, C, D, E...** . The physical separation between cities **C** and **A** is $d_{AC}=d_{CA}$. We wish to arrange these cities in such a manner that a global round trip will be of minimum distance.

We will solve the Traveling Salesman problem with the use of an N x N neural network. How do we set up such a net? Note, first of all, that the solution must satisfy the rooks problem. In other words, only one neuron can be on in each row and in each column. Thus, we start our net by using a Rooks problem neural network. In addition, we would like to discourage cities that are far apart to be listed together. This is accomplished by lateral inhibition of adjacent cities proportional to their separation. A large separation thus results in a large inhibition.

Use of neural networks to solve the traveling salesman problem was first suggested by Hopfield.



Traveling Salesman Neural Network

Superimpose:

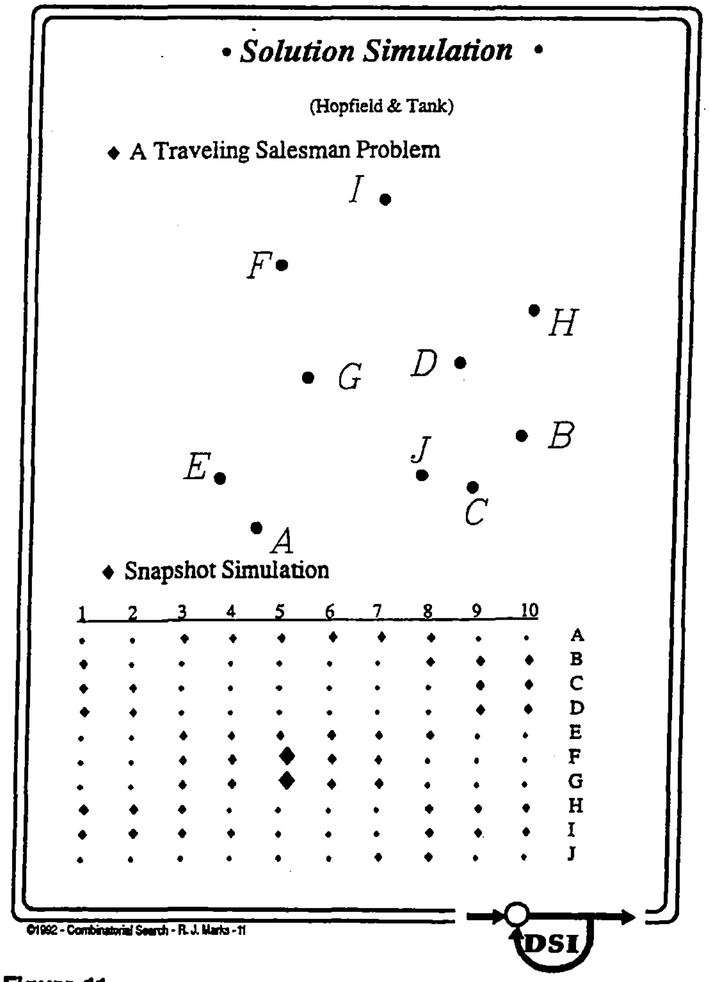
- Rooks Neural Network
- Inhibition Connects
  - Let d_{XY} be the distance between cities X and Y.
  - Inhibit the neuron pairs *proportional* to d_{XY}.

- Global Inhibition

Figure 10

Neuron pairs corresponding to cities that are close together should inhibit each other less than those corresponding to cities that are far away. Thus, superimposed on the Rooks-type inhibition interconnects are interconnects between neuron rows that are proportional to the distance between the corresponding pair of cities. Additional homogeneous global inhibition interconnects also prove useful in the network's performance.

Notes Notes Notes



• *Solution Simulation* •

(Hopfield & Tank)

◆ A Traveling Salesman Problem

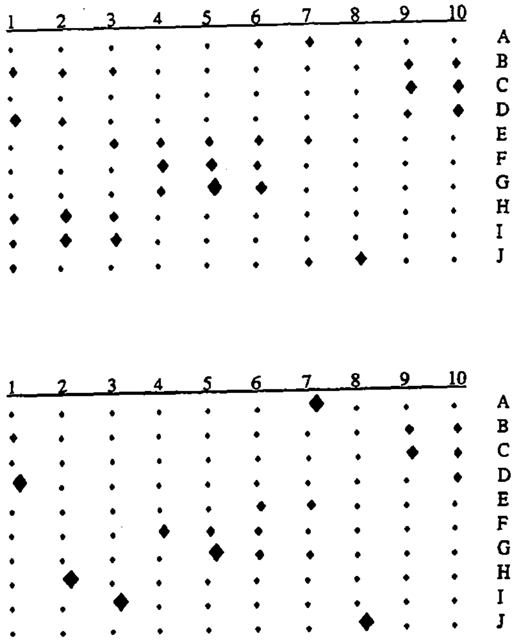
◆ Snapshot Simulation

©1992 - Combinatorial Search - R. J. Marks - 11

Figure 11

These are snapshots of the evolution of the solution of a traveling salesman problem.

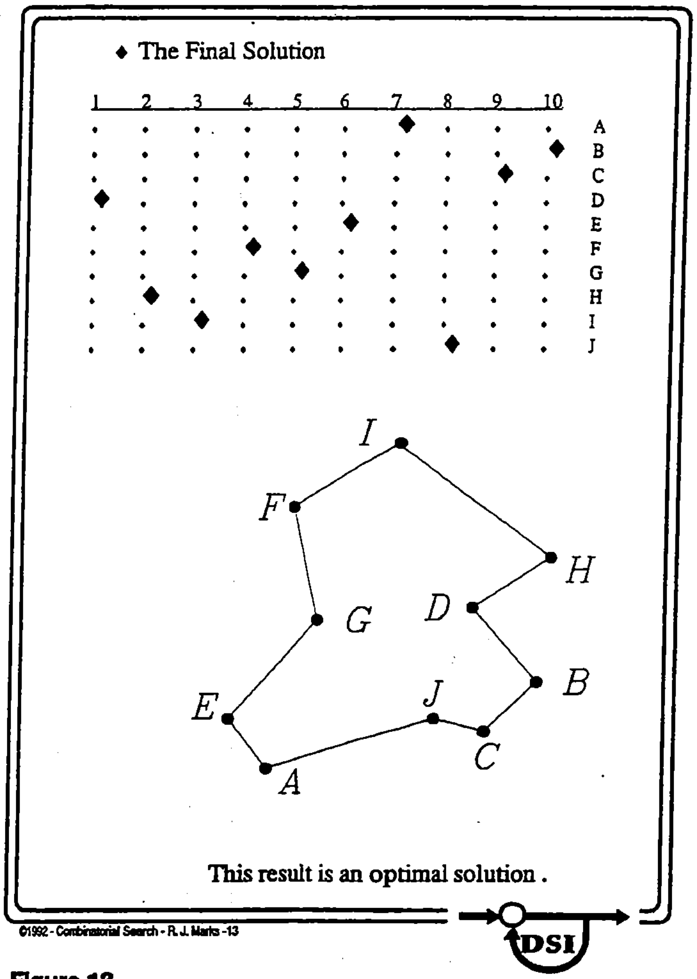
Continued:



©1992 - Combinatorial Search - R. J. Marks - 12

Figure 12

Notes Notes Notes



◆ The Final Solution

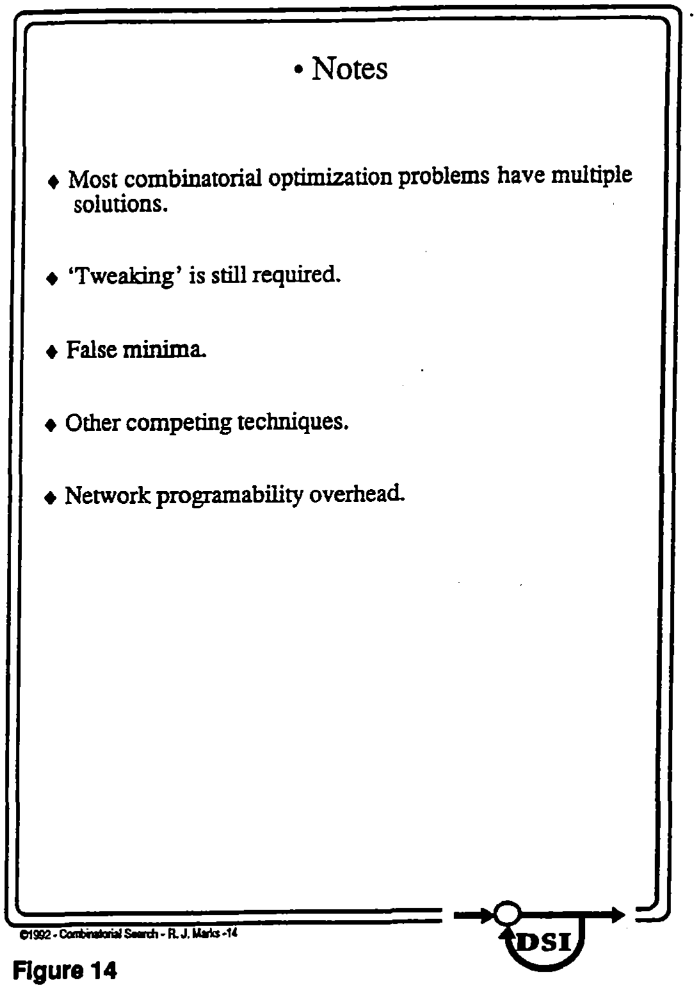
1	2	3	4	5	6	7	8	9	10	
•	•	•	•	•	•	◆	•	•	•	A
•	•	•	•	•	•	•	•	•	◆	B
•	•	•	•	•	•	•	•	◆	•	C
◆	•	•	•	•	•	•	•	•	•	D
•	•	•	•	•	◆	•	•	•	•	E
•	•	•	◆	•	•	•	•	•	•	F
•	•	•	•	◆	•	•	•	•	•	G
•	◆	•	•	•	•	•	•	•	•	H
•	•	◆	•	•	•	•	•	•	•	I
•	•	•	•	•	•	•	◆	•	•	J

This result is an optimal solution .

©1992 - Combinatorial Search - R. J. Marks -13

Figure 13

The solution is optimal. In practice, we generally do not have the luxury to know whether the result is optimal.



• Notes

- ♦ Most combinatorial optimization problems have multiple solutions.
- ♦ 'Tweaking' is still required.
- ♦ False minima.
- ♦ Other competing techniques.
- ♦ Network programability overhead.

©1992 - Combinatorial Search - R. J. Marks -14

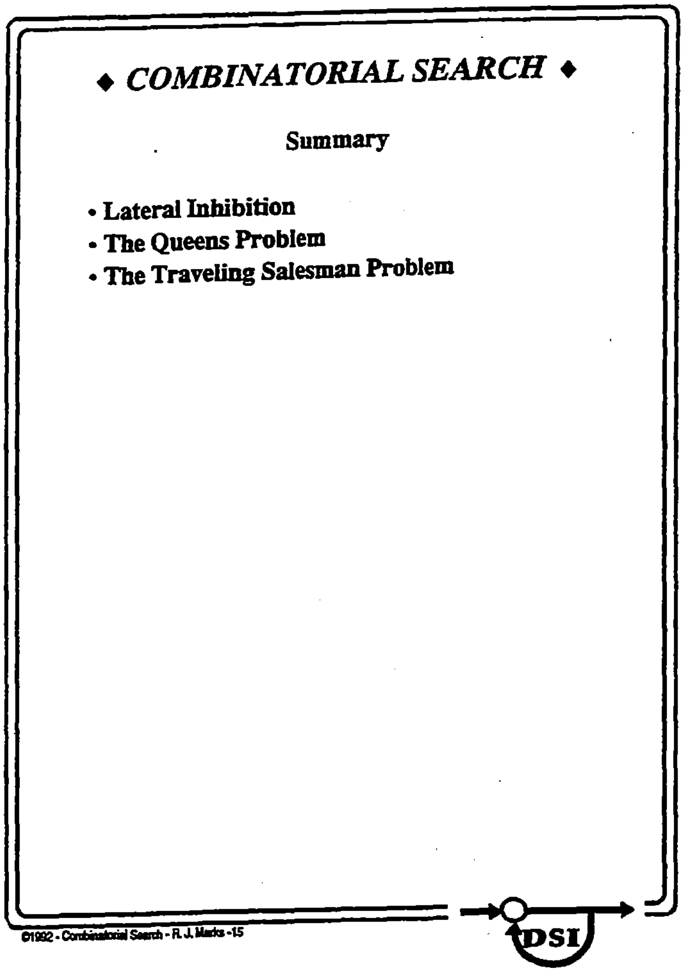
Figure 14

1. With the exception of *the King of the Hill Problem,* all of the problems thus far considered have more than one solution. As a result, we can *clamp* some neurons of the neurons to *on* and, if consistent with the problem solution, the network will produce a consistent steady state result. If, for example, we clamped the upper right neuron in *the Queens Problem* to one, then the neural network will converge to a solution that specifies that a Queen be in the upper right hand corner of the chess board.

2. Neural networks require *tweaking* in order to generate optimal results in combinatorial search problems. In certain cases, such as the *Queens Problem,* inspection of the solution lets us know if the result is optimum. In other cases, such as *the Traveling Salesman Problem,* we are not allowed this luxury. The parameters are a, w, the city distance proportionality constant and the global inhibition constant. At this time, choosing the free parameters of such neural networks is more of an art than a science. Nevertheless, the neural

network will generally give a *good* rather than an optimal answer. Such performance is also seen in certain neural network associative memories.

3. The iteration can become stuck in false minima.
4. The jury is still out on whether neural networks will be competitive with other techniques in the performance of search algorithms. *The Queens Problem*, for example, can be solved with a few lines of code in *LISP*.
5. *Network programmability* must be taken into account in the comparative evaluation of neural networks.



◆ COMBINATORIAL SEARCH ◆

Summary

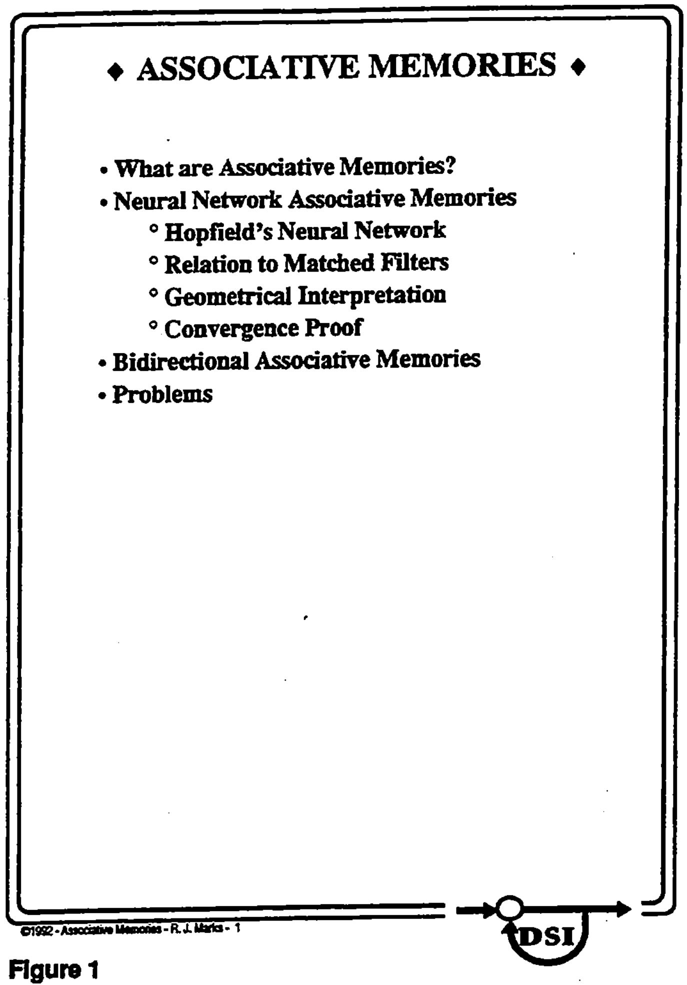
- **Lateral Inhibition**
- **The Queens Problem**
- **The Traveling Salesman Problem**

©1992 - Combinatorial Search - R. J. Marks -15

Figure 15

Notes Notes Notes

Associative Memories

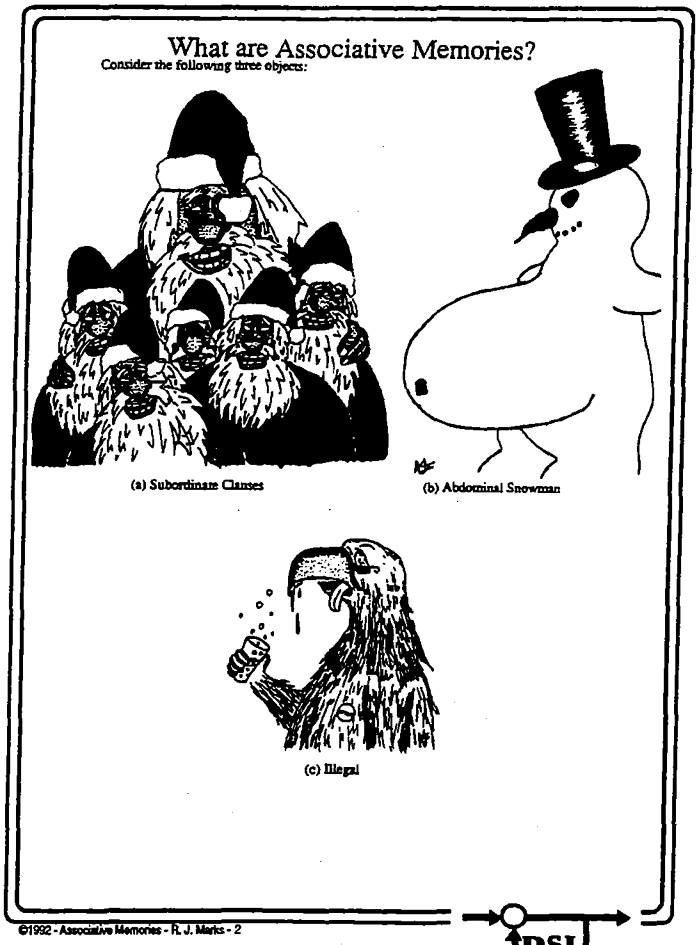


♦ ASSOCIATIVE MEMORIES ♦

- What are Associative Memories?
- Neural Network Associative Memories
  - ° Hopfield's Neural Network
  - ° Relation to Matched Filters
  - ° Geometrical Interpretation
  - ° Convergence Proof
- Bidirectional Associative Memories
- Problems

Figure 1

Notes Notes Notes



What are Associative Memories?

Consider the following three objects:

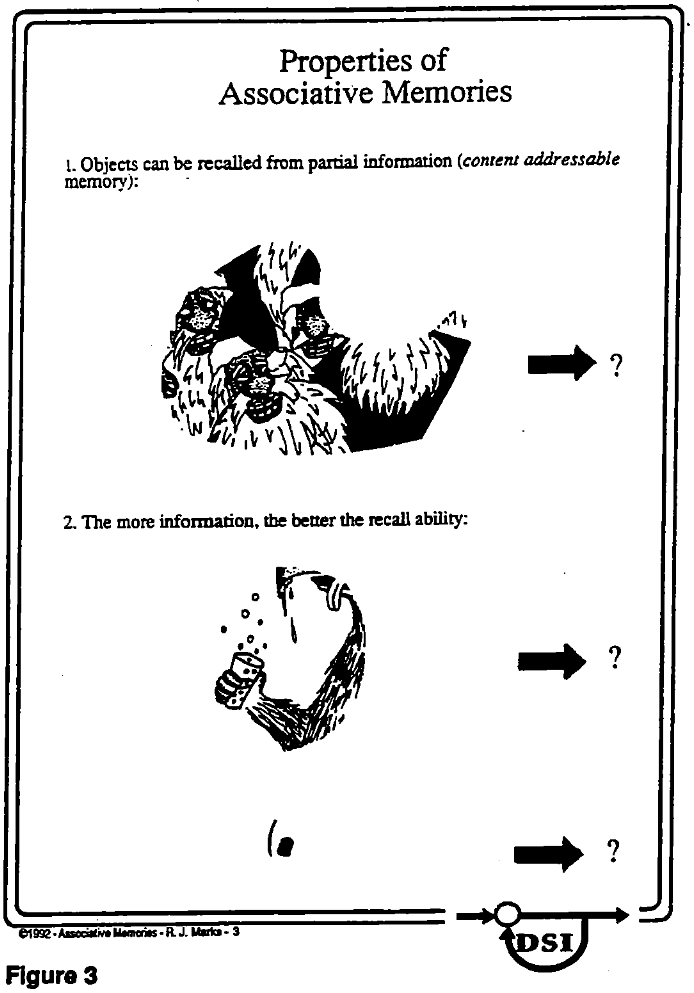
(a) Subordinate Clauses

(b) Abdominal Snowman

(c) Illegal

Figure 2

Properties of the neural network associative memory will now be illustrated with your own associative memory. By looking at these images, you are programming your own associative memory.



Properties of Associative Memories

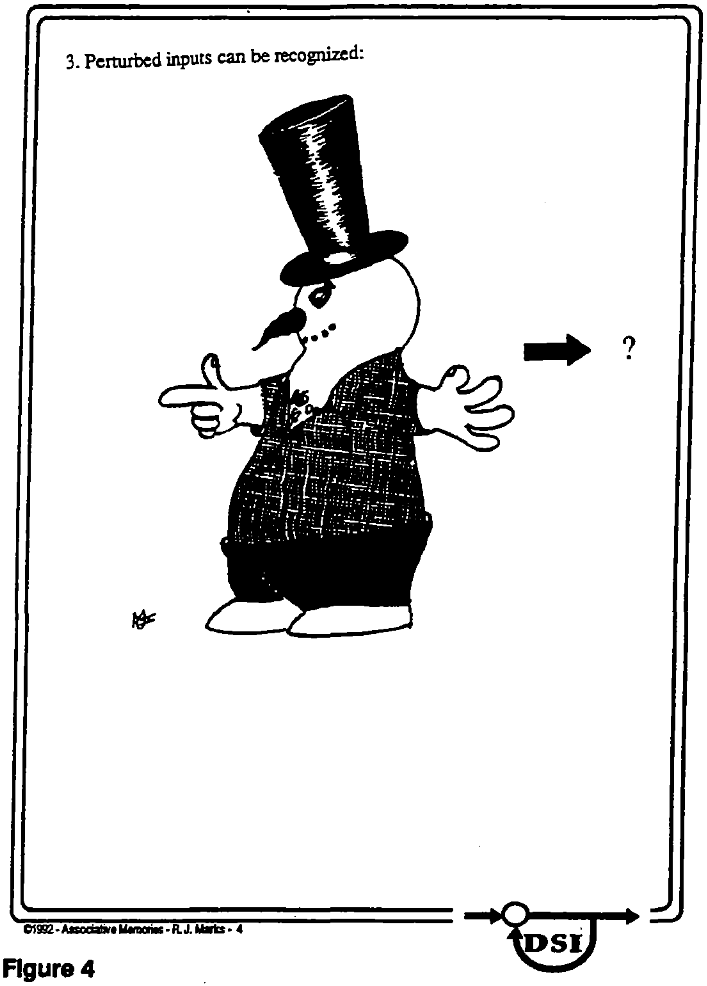
1. Objects can be recalled from partial information (*content addressable* memory):

2. The more information, the better the recall ability:

Figure 3

1. Recall from the library can be performed by knowledge of only a part of the information. In essence, we are performing a *content addressable memory* operation.

2. If the known portion of the information is too small, we cannot recall the library element.

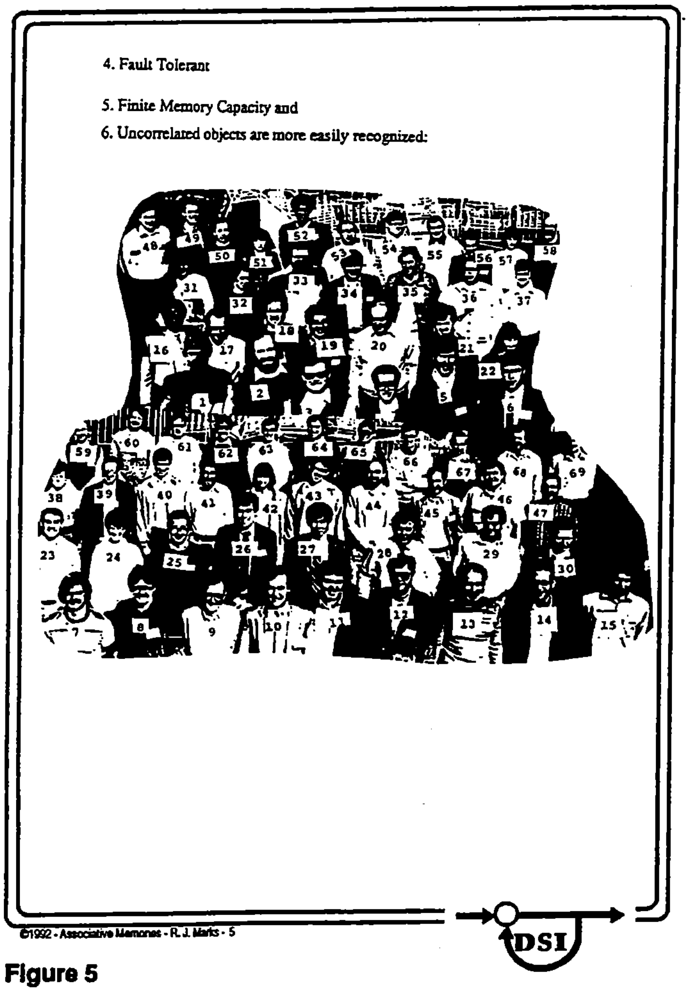


3. Perturbed inputs can be recognized:

©1992 - Associative Memories - R.J. Marks - 4

Figure 4

This picture deviates significantly from that memorized. It can still, however, be recognized.



4. Fault Tolerant

5. Finite Memory Capacity and

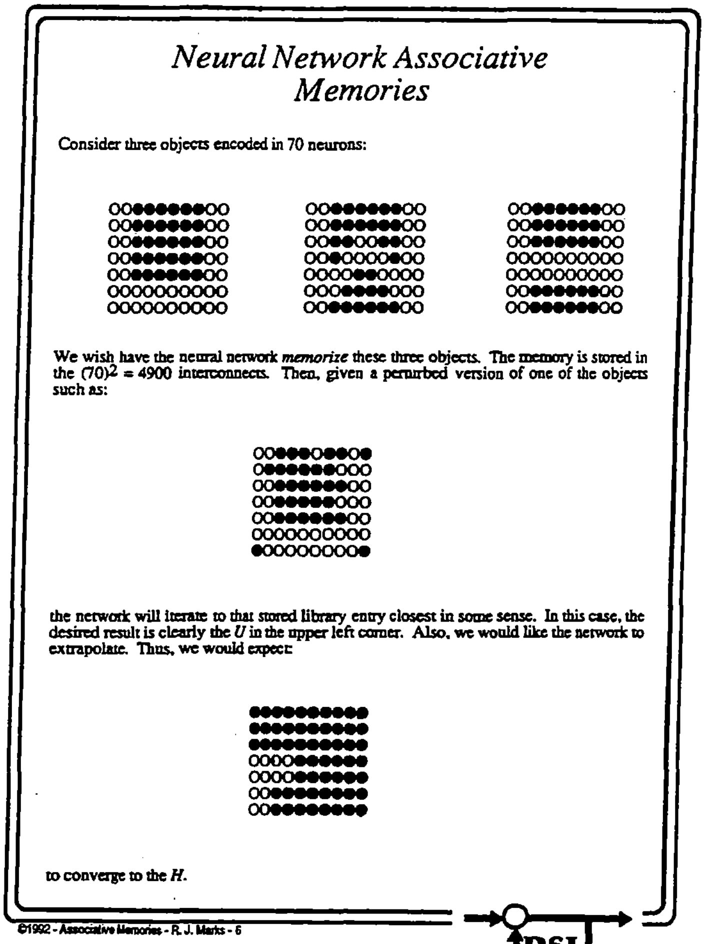
6. Uncorrelated objects are more easily recognized:

©1992 - Associative Memories - R. J. Marks - 5

Figure 5

Notes Notes Notes

3. Portions of biological brains can be removed and the neural network still works. The same is true of certain artificial neural networks.
4. There is a maximum memory capacity. (Hopfield is #49. Marks is #33, E. Leith #20.)
5. Uncorrelated objects (both in the mathematical and the dictionary sense of the word) are less recognizable.



Neural Network Associative Memories

Consider three objects encoded in 70 neurons:

```
○○●●●●●●○○   ○○●●●●●●○○   ○○●●●●●●○○
○○●●●●●●○○   ○○●●●●●●○○   ○○●●●●●●○○
○○●●●●●●○○   ○○●●○○●●○○   ○○●●●●●●○○
○○●●●●●●○○   ○○●○○○○●○○   ○○○○○○○○○○
○○●●●●●●○○   ○○○○●●○○○○   ○○○○○○○○○○
○○○○○○○○○○   ○○○●●●●○○○   ○○●●●●●●○○
○○○○○○○○○○   ○○●●●●●●○○   ○○●●●●●●○○
```

We wish have the neural network *memorize* these three objects. The memory is stored in the $(70)^2 = 4900$ interconnects. Then, given a perturbed version of one of the objects such as:

```
○○●●●○●●○●
○●●●●●●○○○
○○●●●●●●○○
○○●●●●●○○○
○○●●●●●●○○
○○○○○○○○○○
●○○○○○○○○●
```

the network will iterate to that stored library entry closest in some sense. In this case, the desired result is clearly the *U* in the upper left corner. Also, we would like the network to extrapolate. Thus, we would expect:

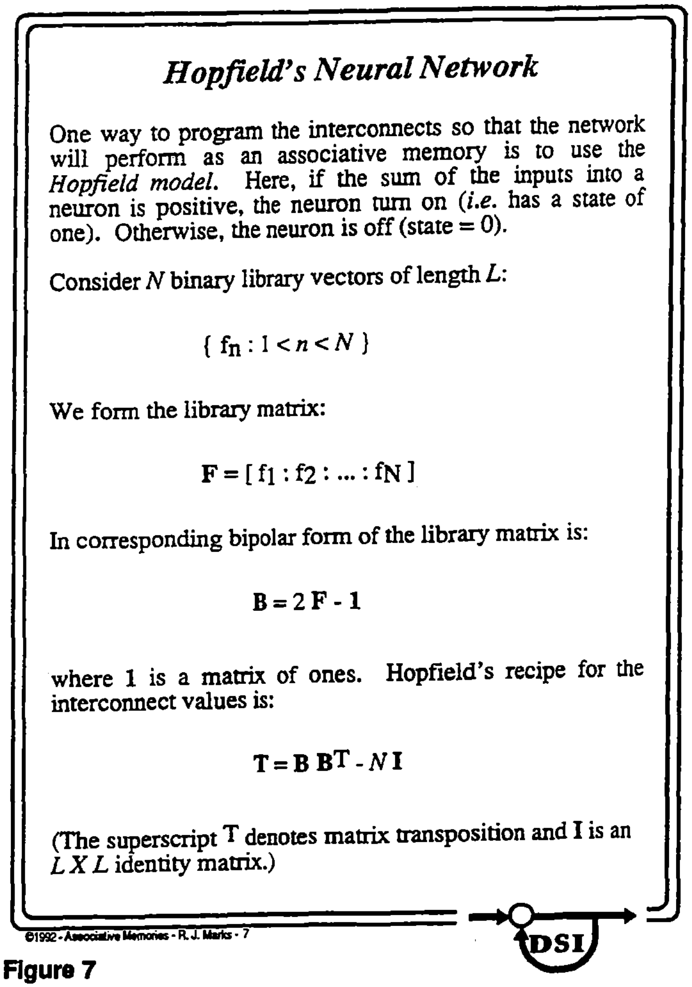
```
●●●●●●●●●●
●●●●●●●●●●
●●●●●●●●●●
○○○○●●●●●●
○○○○●●●●●●
○○●●●●●●●●
○○●●●●●●●●
```

to converge to the *H*.

©1992 - Associative Memories - R. J. Marks - 6

Figure 6

In the case of the original Hopfield neural network, neurons in steady state are either on or off. Pattern information is actually stored in the interconnects. When the neural network is initialized 'close' to a stored pattern, the network ideally converges to that pattern.



Hopfield's Neural Network

One way to program the interconnects so that the network will perform as an associative memory is to use the *Hopfield model.* Here, if the sum of the inputs into a neuron is positive, the neuron turn on (*i.e.* has a state of one). Otherwise, the neuron is off (state = 0).

Consider N binary library vectors of length L:

$$\{ f_n : 1 < n < N \}$$

We form the library matrix:

$$\mathbf{F} = [f_1 : f_2 : \ldots : f_N]$$

In corresponding bipolar form of the library matrix is:

$$\mathbf{B} = 2\,\mathbf{F} - \mathbf{1}$$

where $\mathbf{1}$ is a matrix of ones. Hopfield's recipe for the interconnect values is:

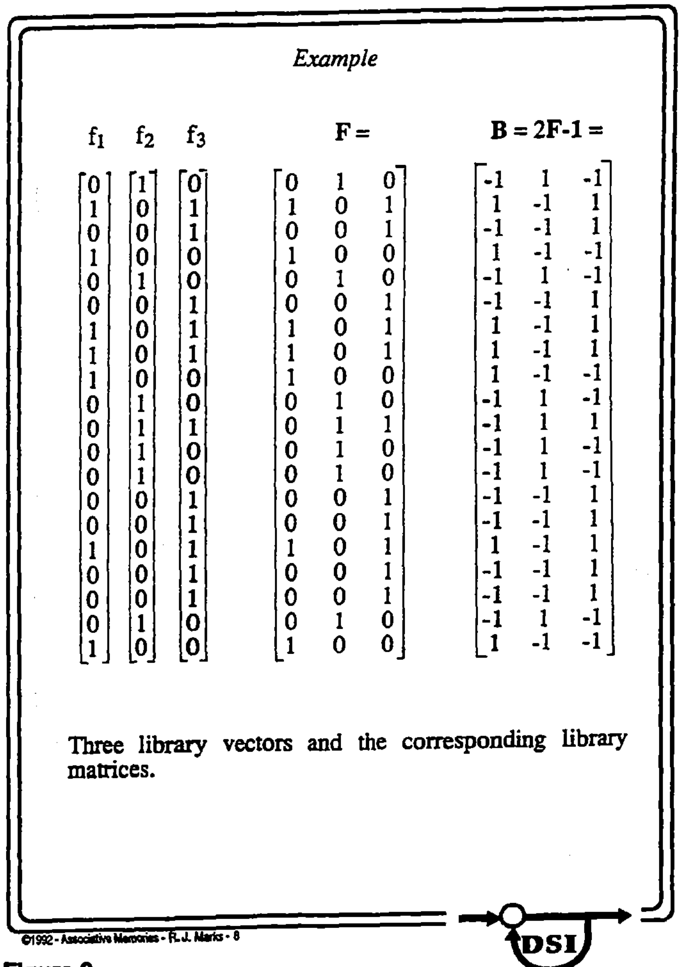
$$\mathbf{T} = \mathbf{B}\,\mathbf{B}^{T} - N\,\mathbf{I}$$

(The superscript ^{T} denotes matrix transposition and **I** is an $L \times L$ identity matrix.)

©1992 - Associative Memories - R. J. Marks - 7

Figure 7

These equations describe the manner in which the interconnect weights for the Hopfield neural network are chosen.



Example

$$\begin{array}{ccc} f_1 & f_2 & f_3 \\ \begin{bmatrix}0\\1\\0\\1\\0\\0\\1\\1\\1\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\1\end{bmatrix} & \begin{bmatrix}1\\0\\0\\0\\1\\0\\0\\0\\0\\1\\1\\1\\1\\0\\0\\0\\0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\1\\1\\0\\0\\1\\1\\1\\0\\0\\1\\0\\0\\1\\1\\1\\1\\1\\0\\0\end{bmatrix} \end{array} \qquad \mathbf{F} = \begin{bmatrix}0&1&0\\1&0&1\\0&0&1\\1&0&0\\0&1&0\\0&0&1\\1&0&1\\1&0&1\\1&0&0\\0&1&0\\0&1&1\\0&1&0\\0&1&0\\0&0&1\\0&0&1\\1&0&1\\0&0&1\\0&0&1\\0&1&0\\1&0&0\end{bmatrix} \qquad \mathbf{B = 2F\text{-}1} = \begin{bmatrix}-1&1&-1\\1&-1&1\\-1&-1&1\\1&-1&-1\\-1&1&-1\\-1&-1&1\\1&-1&1\\1&-1&1\\1&-1&-1\\-1&1&-1\\-1&1&1\\-1&1&-1\\-1&1&-1\\-1&-1&1\\-1&-1&1\\1&-1&1\\-1&-1&1\\-1&-1&1\\-1&1&-1\\1&-1&-1\end{bmatrix}$$

Three library vectors and the corresponding library matrices.

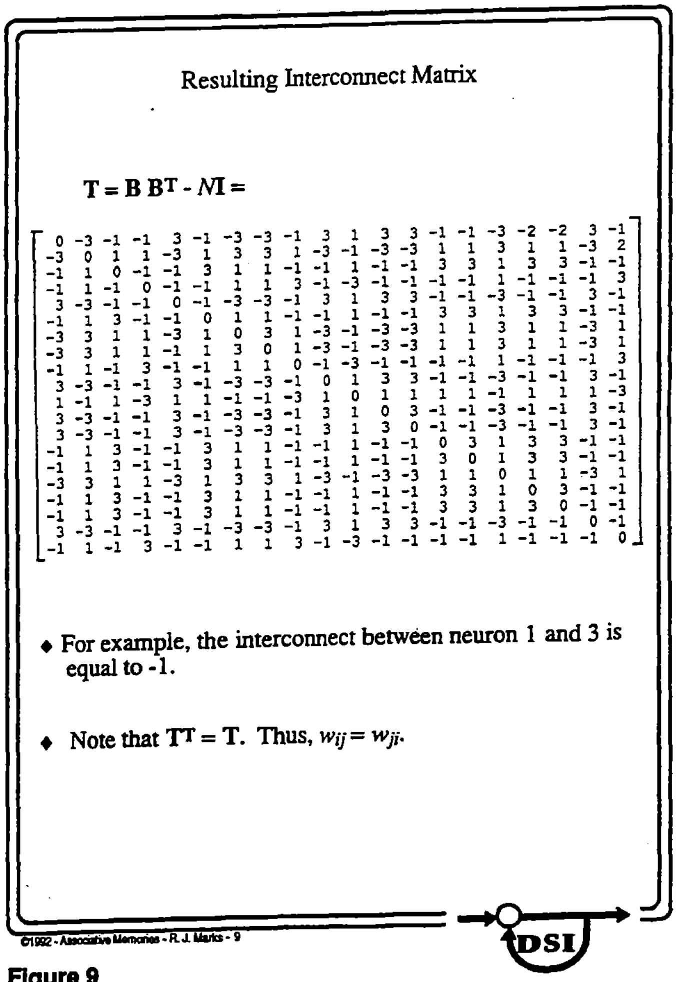
©1992 - Associative Memories - R. J. Marks - 8

DSI

Figure 8

Shown are three example library vectors and the corresponding library (**F**) and **B** matrices.

Notes Notes Notes



Resulting Interconnect Matrix

$$\mathbf{T} = \mathbf{B}\,\mathbf{B}^{\mathrm{T}} - M\mathbf{I} =$$

$$\begin{bmatrix}
0 & -3 & -1 & -1 & 3 & -1 & -3 & -3 & -1 & 3 & 1 & 3 & 3 & -1 & -1 & -3 & -2 & -2 & 3 & -1 \\
-3 & 0 & 1 & 1 & -3 & 1 & 3 & 3 & 1 & -3 & -1 & -3 & -3 & 1 & 1 & 3 & 1 & 1 & -3 & 2 \\
-1 & 1 & 0 & -1 & -1 & 3 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & 3 & 3 & 1 & 3 & 3 & -1 & -1 \\
-1 & 1 & -1 & 0 & -1 & -1 & 1 & 1 & 3 & -1 & -3 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 3 \\
3 & -3 & -1 & -1 & 0 & -1 & -3 & -3 & -1 & 3 & 1 & 3 & 3 & -1 & -1 & -3 & -1 & -1 & 3 & -1 \\
-1 & 1 & 3 & -1 & -1 & 0 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & 3 & 3 & 1 & 3 & 3 & -1 & -1 \\
-3 & 3 & 1 & 1 & -3 & 1 & 0 & 3 & 1 & -3 & -1 & -3 & -3 & 1 & 1 & 3 & 1 & 1 & -3 & 1 \\
-3 & 3 & 1 & 1 & -1 & 1 & 3 & 0 & 1 & -3 & -1 & -3 & -3 & 1 & 1 & 3 & 1 & 1 & -3 & 1 \\
-1 & 1 & -1 & 3 & -1 & -1 & 1 & 1 & 0 & -1 & -3 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 3 \\
3 & -3 & -1 & -1 & 3 & -1 & -3 & -3 & -1 & 0 & 1 & 3 & 3 & -1 & -1 & -3 & -1 & -1 & 3 & -1 \\
1 & -1 & 1 & -3 & 1 & 1 & -1 & -1 & -3 & 1 & 0 & 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -3 \\
3 & -3 & -1 & -1 & 3 & -1 & -3 & -3 & -1 & 3 & 1 & 0 & 3 & -1 & -1 & -3 & -1 & -1 & 3 & -1 \\
3 & -3 & -1 & -1 & 3 & -1 & -3 & -3 & -1 & 3 & 1 & 3 & 0 & -1 & -1 & -3 & -1 & -1 & 3 & -1 \\
-1 & 1 & 3 & -1 & -1 & 3 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & 0 & 3 & 1 & 3 & 3 & -1 & -1 \\
-1 & 1 & 3 & -1 & -1 & 3 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & 3 & 0 & 1 & 3 & 3 & -1 & -1 \\
-3 & 3 & 1 & 1 & -3 & 1 & 3 & 3 & 1 & -3 & -1 & -3 & -3 & 1 & 1 & 0 & 1 & 1 & -3 & 1 \\
-1 & 1 & 3 & -1 & -1 & 3 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & 3 & 3 & 1 & 0 & 3 & -1 & -1 \\
-1 & 1 & 3 & -1 & -1 & 3 & 1 & 1 & -1 & -1 & 1 & -1 & -1 & 3 & 3 & 1 & 3 & 0 & -1 & -1 \\
3 & -3 & -1 & -1 & 3 & -1 & -3 & -3 & -1 & 3 & 1 & 3 & 3 & -1 & -1 & -3 & -1 & -1 & 0 & -1 \\
-1 & 1 & -1 & 3 & -1 & -1 & 1 & 1 & 3 & -1 & -3 & -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 0
\end{bmatrix}$$

- For example, the interconnect between neuron 1 and 3 is equal to -1.

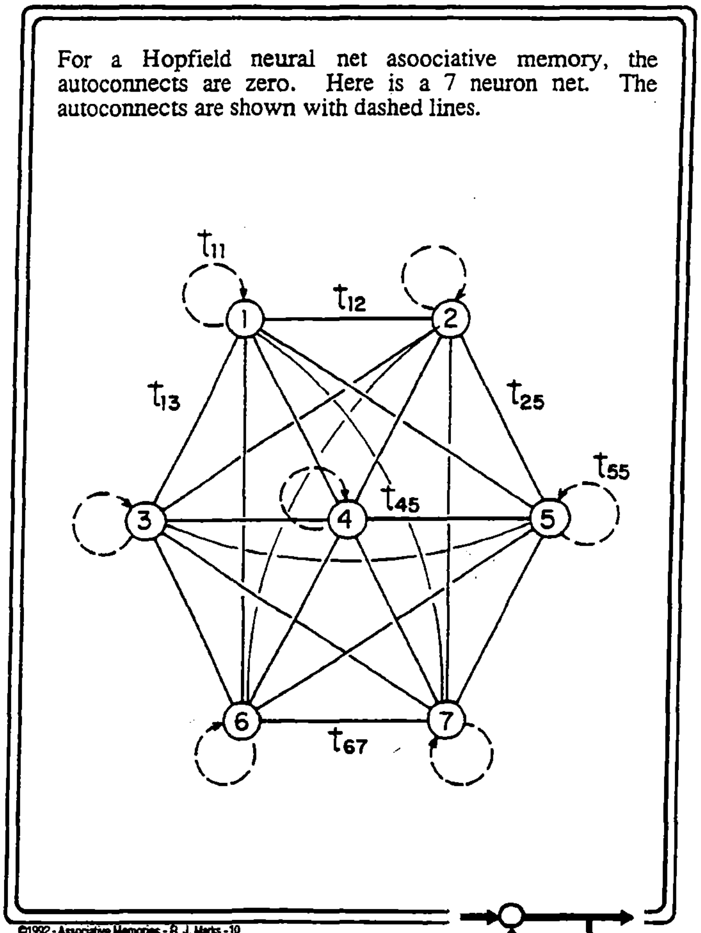
- Note that $\mathbf{T}^{\mathrm{T}} = \mathbf{T}$. Thus, $w_{ij} = w_{ji}$.

©1992 - Associative Memories - R. J. Marks - 9

DSI

Figure 9

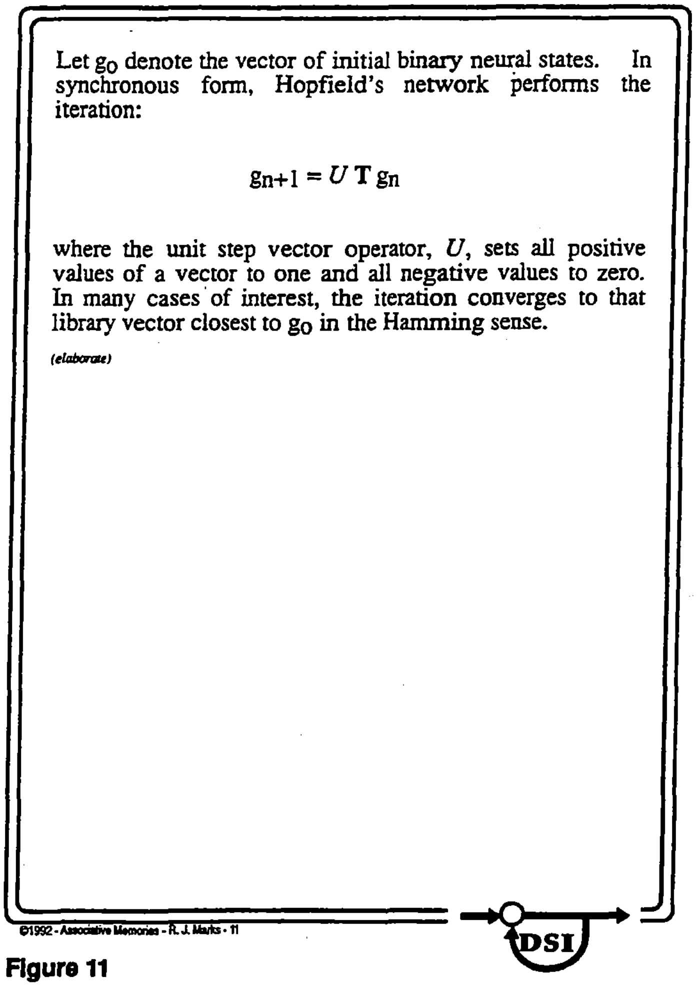
Shown is the interconnect matrix for the library vectors show on the previous page. The interconnect between neuron i and j is w_{ij}. By design, the matrix is symmetrical. This, the connection between neuron i and j is the same as that between neuron j and i.



For a Hopfield neural net asoociative memory, the autoconnects are zero. Here is a 7 neuron net. The autoconnects are shown with dashed lines.

Figure 10

The original Hopfield neural network had no autoconnects.



Let g_0 denote the vector of initial binary neural states. In synchronous form, Hopfield's network performs the iteration:

$$g_{n+1} = U\,\mathbf{T}\,g_n$$

where the unit step vector operator, U, sets all positive values of a vector to one and all negative values to zero. In many cases of interest, the iteration converges to that library vector closest to g_0 in the Hamming sense.

(elaborate)

©1992 - Associative Memories - R. J. Marks - 11

DSI

Figure 11

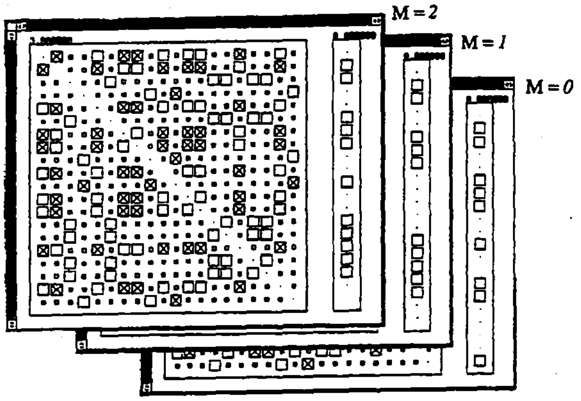
In discrete form, restoration is iterative. The neural states are updated until convergence. Ideally, convergence is to the library vector closest to the initialization.

Notes Notes Notes

Hopfield NN Example:

$$F = \begin{bmatrix} 01010011100000010001 \\ 10001000011110000010 \\ 01100111001001111100 \end{bmatrix}^T$$

☒ = *-1* □ = *1*



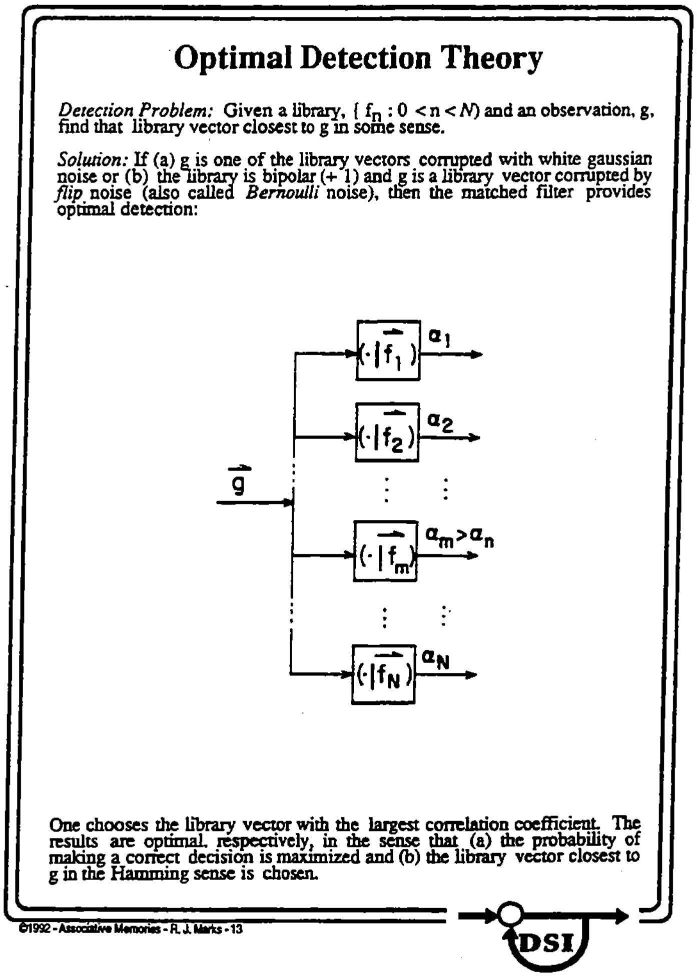
©1992 - Associative Memories - R. J. Marks - 12

DSI

Figure 12

The initialization of the net is shown as $M = 0$. Convergence to the third library vector occurs two iterations.

Notes Notes Notes



Optimal Detection Theory

Detection Problem: Given a library, { $f_n : 0 < n < N$) and an observation, g, find that library vector closest to g in some sense.

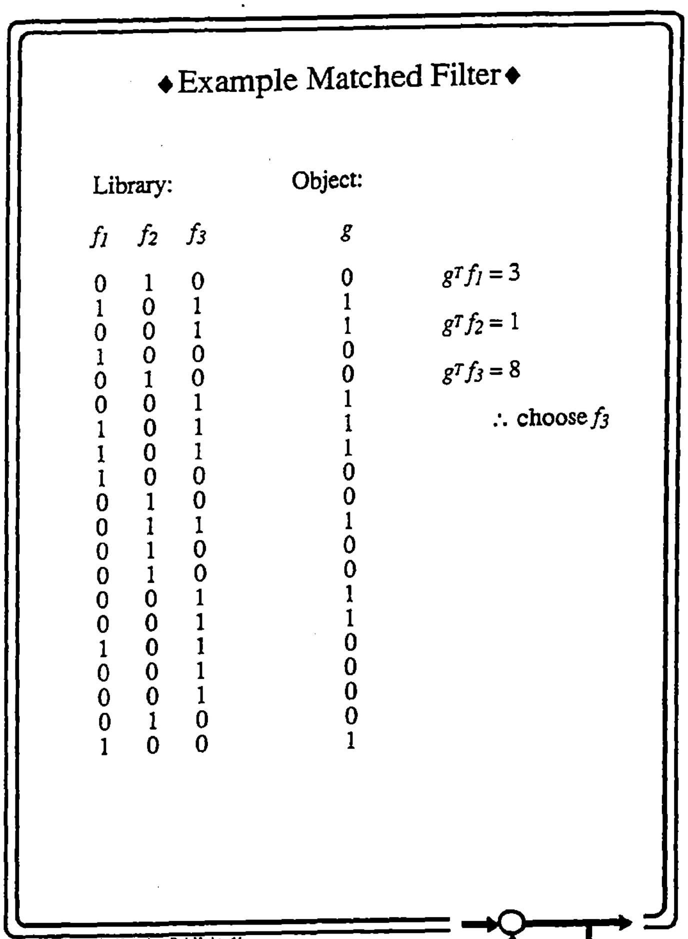
Solution: If (a) g is one of the library vectors corrupted with white gaussian noise or (b) the library is bipolar (+ 1) and g is a library vector corrupted by *flip* noise (also called *Bernoulli* noise), then the matched filter provides optimal detection:

One chooses the library vector with the largest correlation coefficient. The results are optimal, respectively, in the sense that (a) the probability of making a correct decision is maximized and (b) the library vector closest to g in the Hamming sense is chosen.

Figure 13

We will digress for a short time into the field of optimal detection theory. The Hopfield associative memory will evolve from the discussion.

Notes Notes Notes



♦Example Matched Filter♦

Library:			Object:
f_1	f_2	f_3	g
0	1	0	0
1	0	1	1
0	0	1	1
1	0	0	0
0	1	0	0
0	0	1	1
1	0	1	1
1	0	1	1
1	0	0	0
0	1	0	0
0	1	1	1
0	1	0	0
0	1	0	0
0	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
0	0	1	0
0	1	0	0
1	0	0	1

$g^T f_1 = 3$

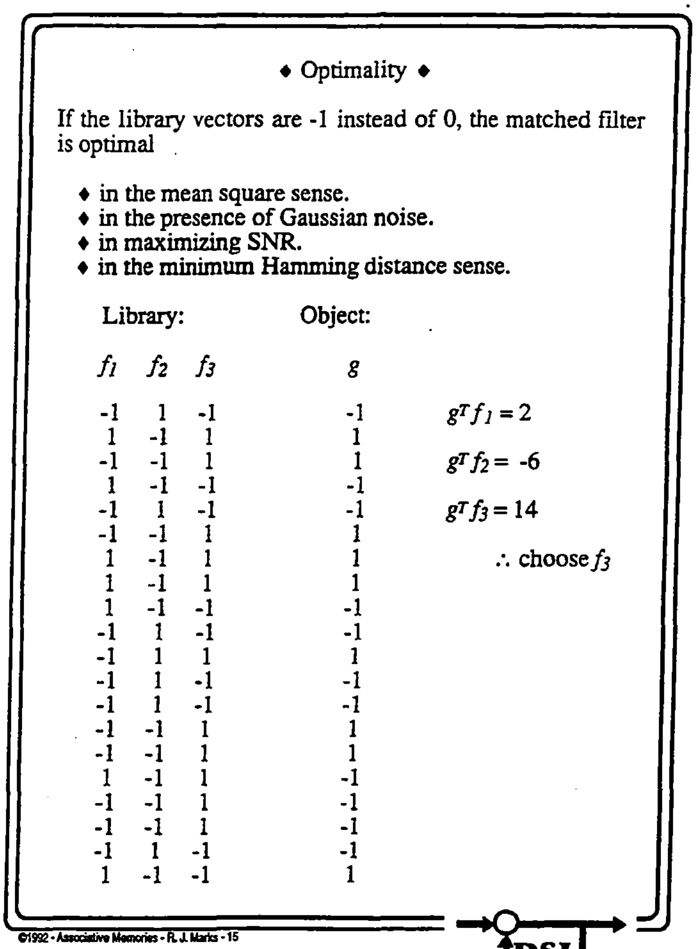
$g^T f_2 = 1$

$g^T f_3 = 8$

$\therefore$ choose f_3

Figure 14

The library function 'closest' to the object, g, yields the largest inner product.



◆ Optimality ◆

If the library vectors are -1 instead of 0, the matched filter is optimal

- ◆ in the mean square sense.
- ◆ in the presence of Gaussian noise.
- ◆ in maximizing SNR.
- ◆ in the minimum Hamming distance sense.

Library:			Object:
f_1	f_2	f_3	g
-1	1	-1	-1
1	-1	1	1
-1	-1	1	1
1	-1	-1	-1
-1	1	-1	-1
-1	-1	1	1
1	-1	1	1
1	-1	1	1
1	-1	-1	-1
-1	1	-1	-1
-1	1	1	1
-1	1	-1	-1
-1	1	-1	-1
-1	-1	1	1
-1	-1	1	1
1	-1	1	-1
-1	-1	1	-1
-1	-1	1	-1
-1	1	-1	-1
1	-1	-1	1

$g^T f_1 = 2$

$g^T f_2 = -6$

$g^T f_3 = 14$

$\therefore$ choose f_3

©1992 - Associative Memories - R. J. Marks - 15

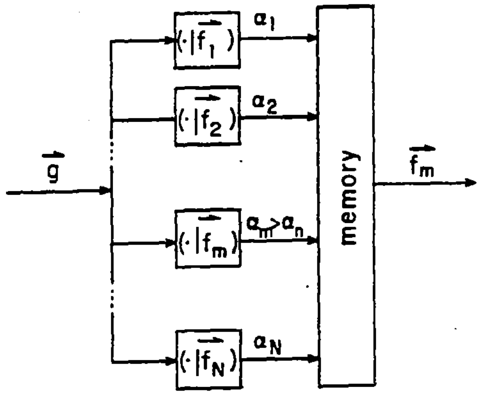
Figure 15

Optimality occurs in matched filtering when the sum of the squares of each of the elements in each library vector is the same. This occurs when all elements are +1 and –1.

Notes Notes Notes

An Optimal Associative Memory:

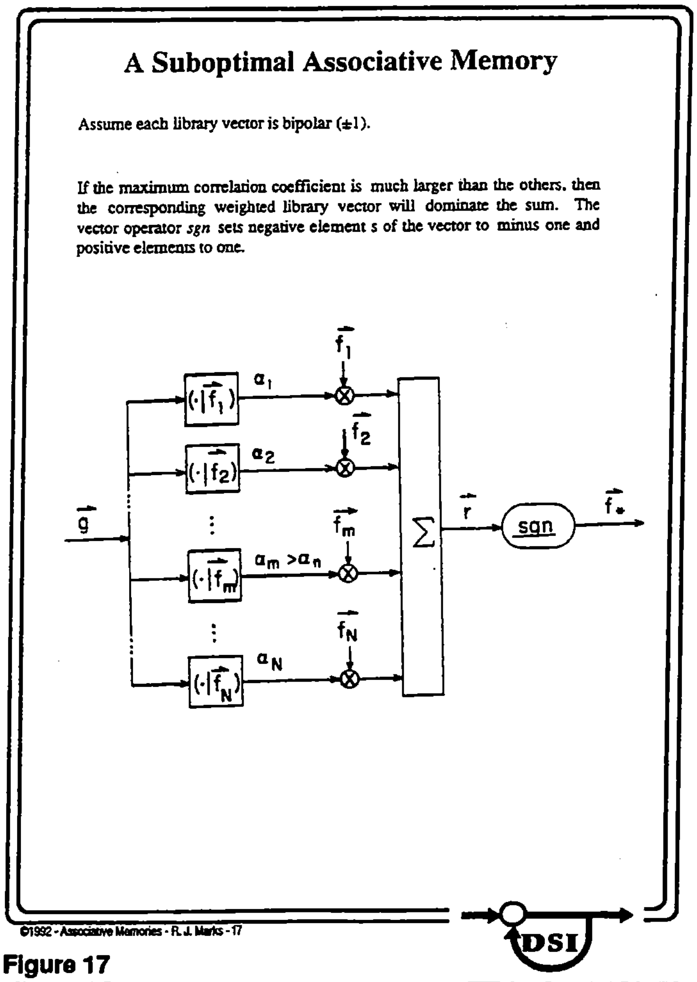
(elaborate)



©1992 - Associative Memories - R. J. Marks - 16

Figure 16

Shown here is an optimal associative memory. A matched filter computes the *correlation coefficients*, α_n. The largest correlation coefficient is used to access the corresponding library vector which is the memory's output. This approach is optimum when the matched filter is optimum. It is optimum in the same sense.



A Suboptimal Associative Memory

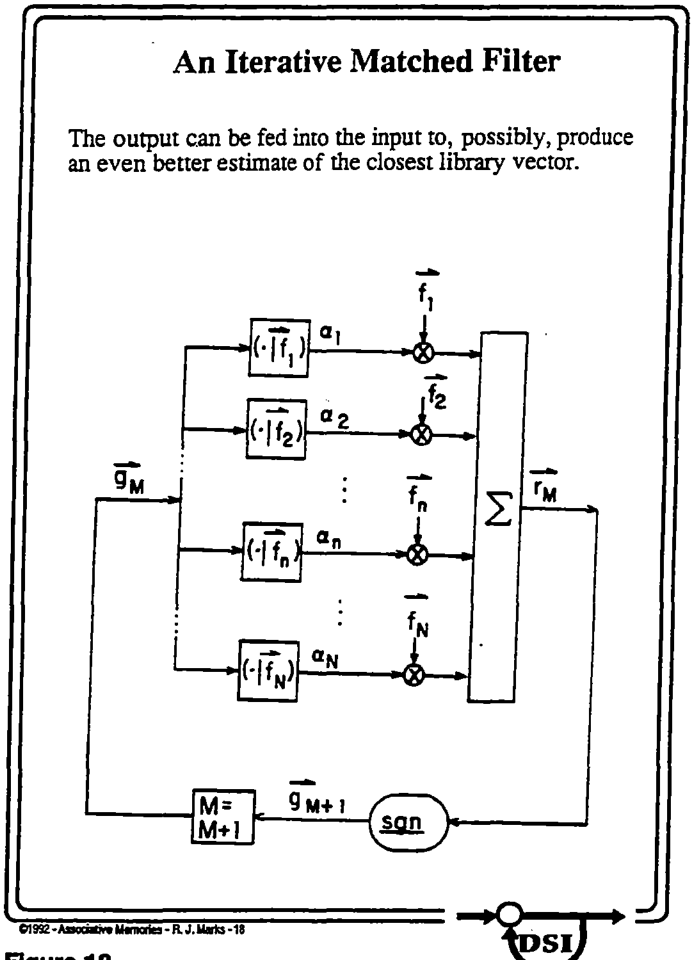
Assume each library vector is bipolar (±1).

If the maximum correlation coefficient is much larger than the others, then the corresponding weighted library vector will dominate the sum. The vector operator *sgn* sets negative element s of the vector to minus one and positive elements to one.

Figure 17

For the bipolar (±1) case, a suboptimal associative memory is that shown. If α_m>> than the other coefficients, then $\alpha_m f_m$ will dominate in the sum $r = \Sigma_n \alpha_n f_n$. If, indeed, α_m is sufficiently large, then $f_* = f_m = \text{sgn}[\Sigma_n \alpha_n f_n]$.

Notes Notes Notes

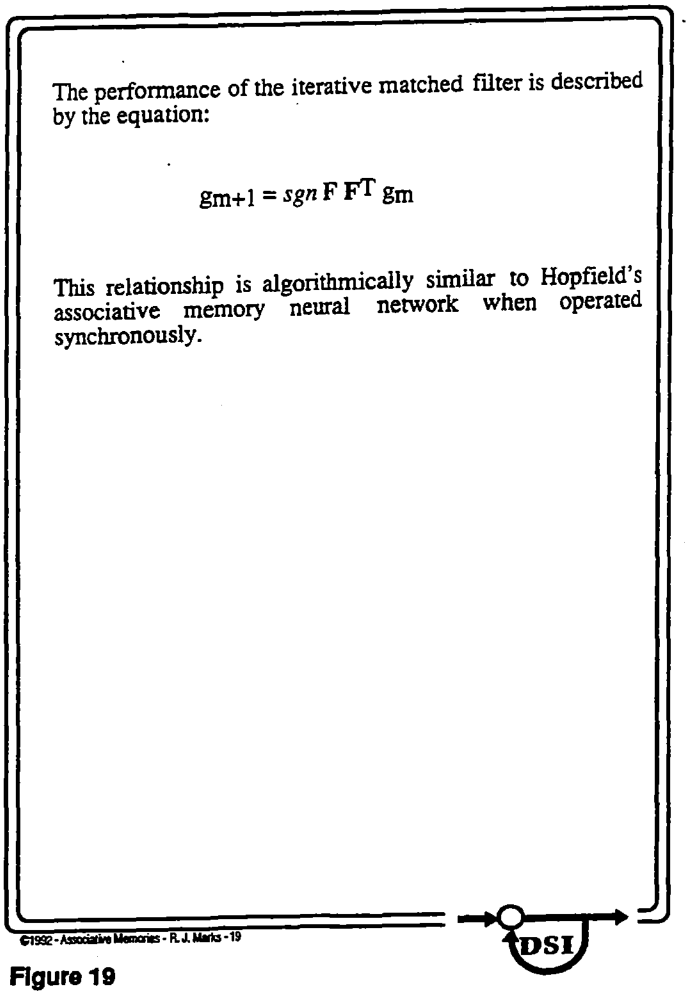


An Iterative Matched Filter

The output can be fed into the input to, possibly, produce an even better estimate of the closest library vector.

Figure 18

The estimate of the last filter could be closer to the desired result than the initial guess. By passing this result through the filter again, an even better result might occur. This is the motivation for the architecture shown



The performance of the iterative matched filter is described by the equation:

$$g_{m+1} = \mathit{sgn}\, F\, F^T g_m$$

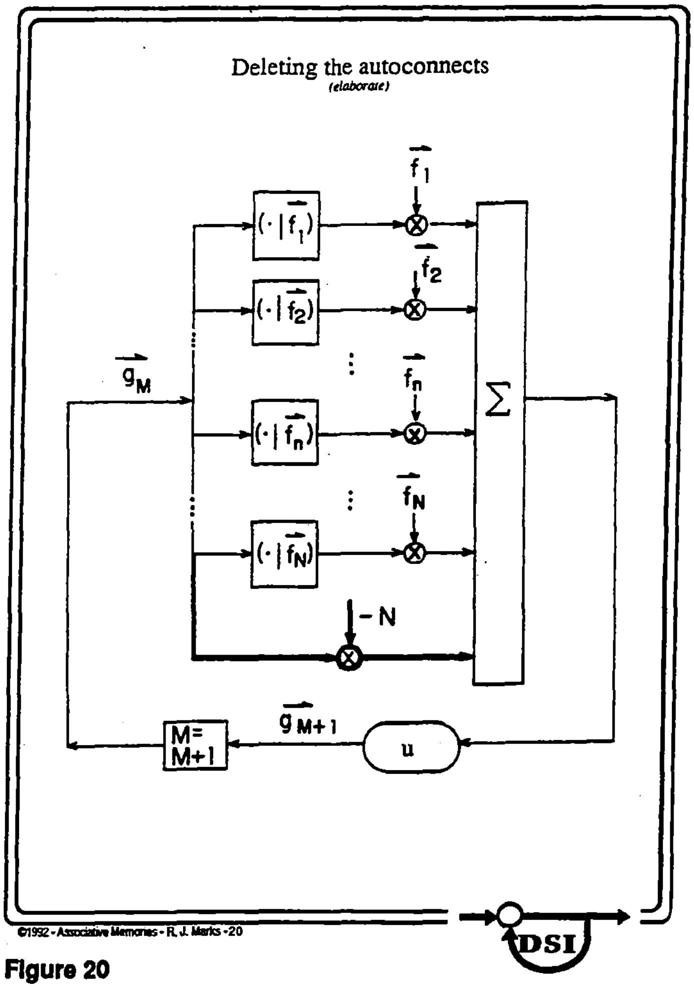
This relationship is algorithmically similar to Hopfield's associative memory neural network when operated synchronously.

©1992 - Associative Memories - R. J. Marks - 19

Figure 19

The equation shown describes the math behind the iterative matched filter.

Notes Notes Notes



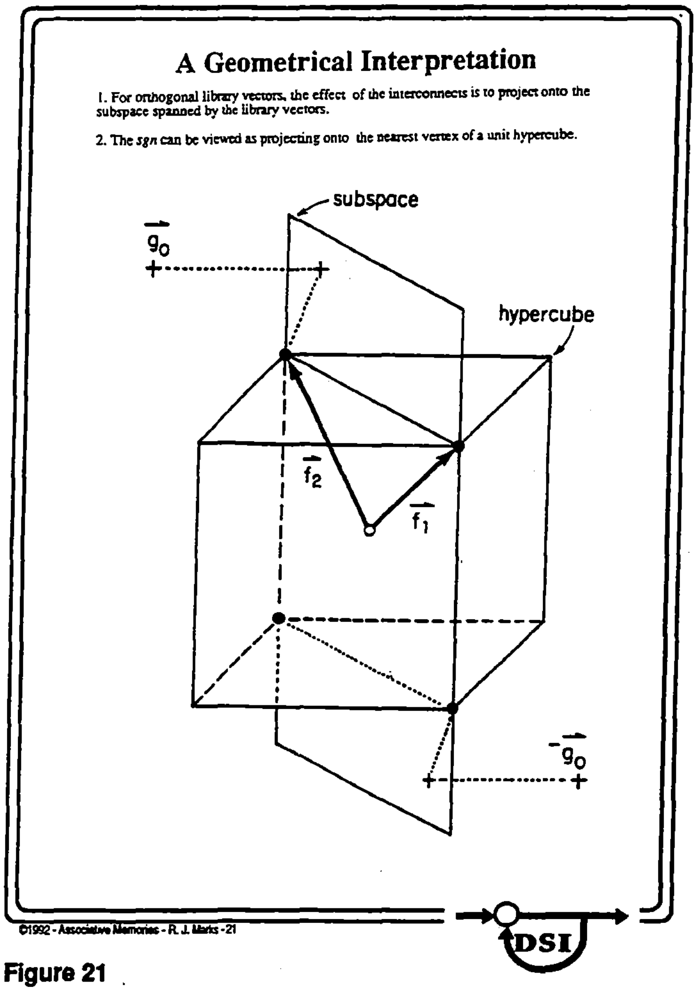
Deleting the autoconnects

(elaborate)

©1992 - Associative Memories - R. J. Marks - 20

Figure 20

This iterative matched filter, modified to 'delete the autoconnects,' performs algorithmically (although not architecturally) identical to the Hopfield associative memory.



A Geometrical Interpretation

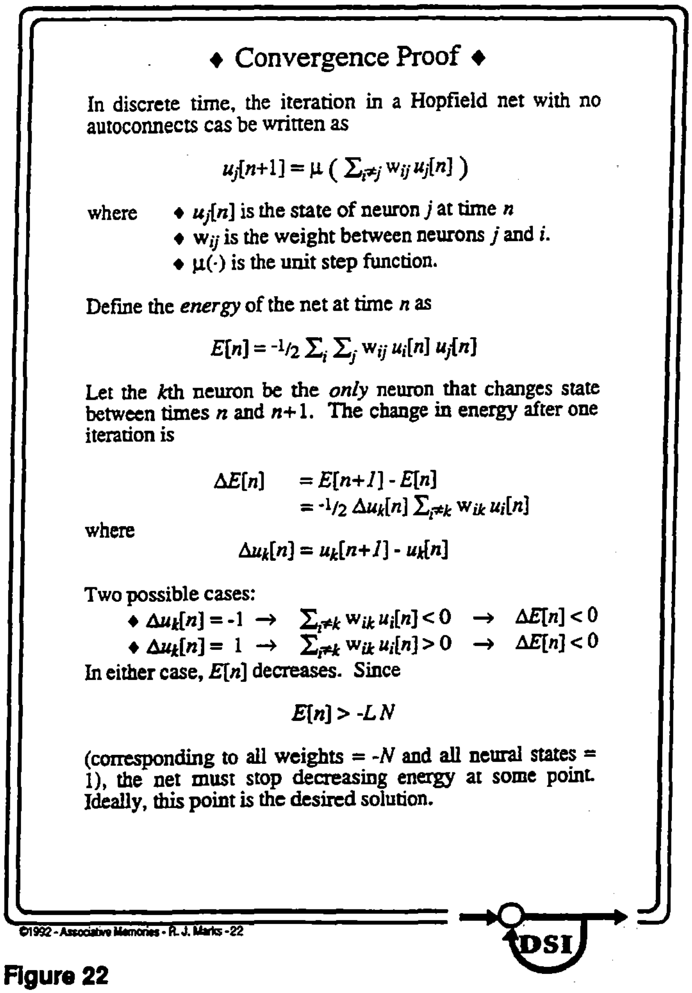
1. For orthogonal library vectors, the effect of the interconnects is to project onto the subspace spanned by the library vectors.

2. The *sgn* can be viewed as projecting onto the nearest vertex of a unit hypercube.

Figure 21

An interesting geometrical illustration of the iterative matched filter is shown here. The plane is the subspace formed by all of the linear combinations of the library vectors. Evaluation of the correlation coefficients is similar to projecting onto this plane from the initial guess, *g*. Performing the sgn operation projects onto the nearest vertice of the hypercube. The next iteration projects back upon the subspace, etc. Iteration is performed until convergence. (Note, by symmetry, the 'twin image' point.)

Notes Notes Notes



♦ Convergence Proof ♦

In discrete time, the iteration in a Hopfield net with no autoconnects cas be written as

$$u_j[n+1] = \mu\left(\sum_{i \neq j} w_{ij}\, u_j[n] \right)$$

where
- ♦ $u_j[n]$ is the state of neuron j at time n
- ♦ w_{ij} is the weight between neurons j and i.
- ♦ $\mu(\cdot)$ is the unit step function.

Define the *energy* of the net at time n as

$$E[n] = -{}^1\!/_2 \sum_i \sum_j w_{ij}\, u_i[n]\, u_j[n]$$

Let the kth neuron be the *only* neuron that changes state between times n and $n+1$. The change in energy after one iteration is

$$\begin{aligned} \Delta E[n] &= E[n+1] - E[n] \\ &= -{}^1\!/_2\, \Delta u_k[n] \sum_{i \neq k} w_{ik}\, u_i[n] \end{aligned}$$

where

$$\Delta u_k[n] = u_k[n+1] - u_k[n]$$

Two possible cases:
- ♦ $\Delta u_k[n] = -1 \rightarrow \sum_{i \neq k} w_{ik}\, u_i[n] < 0 \rightarrow \Delta E[n] < 0$
- ♦ $\Delta u_k[n] = 1 \rightarrow \sum_{i \neq k} w_{ik}\, u_i[n] > 0 \rightarrow \Delta E[n] < 0$

In either case, $E[n]$ decreases. Since

$$E[n] > -L\,N$$

(corresponding to all weights = $-N$ and all neural states = 1), the net must stop decreasing energy at some point. Ideally, this point is the desired solution.

©1992 - Associative Memories - R. J. Marks - 22

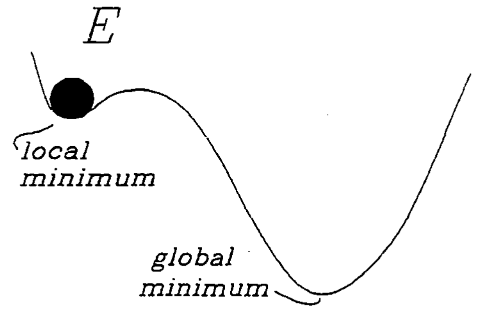
DSI

Figure 22

Shown here is a proof that the asynchronous Hopfield neural network will converge. Ideally, it will converge to the proper solution.

♦ Local Minima ♦

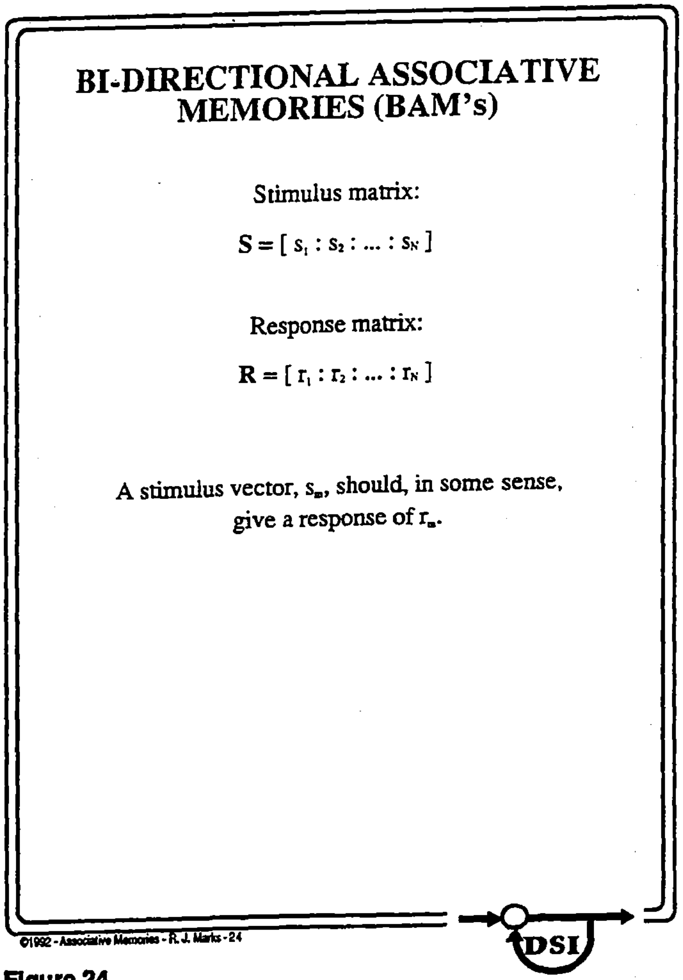
Problem: The iteration can be stuck in local minima.



Solution: Use *simulated annealing*. Application to the Hopfield network results in the *Boltzmann Machine*.

Figure 23

The proof of convergence of the Hopfield neural network does not require that the iteration reach the absolute minimum. It simply allows that any energy must decrease. As in many search problems, the iteration can be stuck in local minima. A technique to avoid this is simulated annealing. In metallurgy, the annealing schedule dictates how an alloy in molten form is to be cooled in order to assure certain attributes. Heat is associated with vibration. Visualize the surface above being shaken, first vigorously and then more and more gradually. The energy of the net, visualized as a marble on a rough surface, would then have the opportunity to jump out of the local minima. In the Hopfield neural network, this is achieved by allowing each neural state to take on the value other than that computed for it. The chance is dictated under some probability schedule that reduces over time. If the Boltzmann probability distribution is used, the resulting neural network has been referred to as a Boltzmann machine.



BI-DIRECTIONAL ASSOCIATIVE MEMORIES (BAM's)

Stimulus matrix:

$$\mathbf{S} = [\, s_1 : s_2 : \ldots : s_N \,]$$

Response matrix:

$$\mathbf{R} = [\, r_1 : r_2 : \ldots : r_N \,]$$

A stimulus vector, s_m, should, in some sense, give a response of r_m.

©1992 - Associative Memories - R. J. Marks - 24

Figure 24

In a BAM, we wish to associate vector pairs. The BAM is a generalization of the Hopfield associative memory.

Sum of Outer Products BAM:

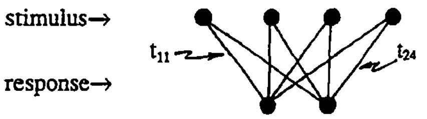
Construct the (non-square) interconnect matrix

$$\mathbf{T} = \mathbf{R}\,\mathbf{S}^T = \Sigma\, r_n\, s_n^T$$

Example:

$$\mathbf{S} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & -1 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 2 & 0 & 0 & -2 \\ 0 & 2 & -2 & 0 \end{bmatrix}$$

The corresponding neural net is:



For an input s_m, the *sgn* of the output for this example is r_m.

Why? Because the stimulus vectors are orthogonal:

$$\Sigma\, s_m^T s_n = 0\,;\; n \neq m$$

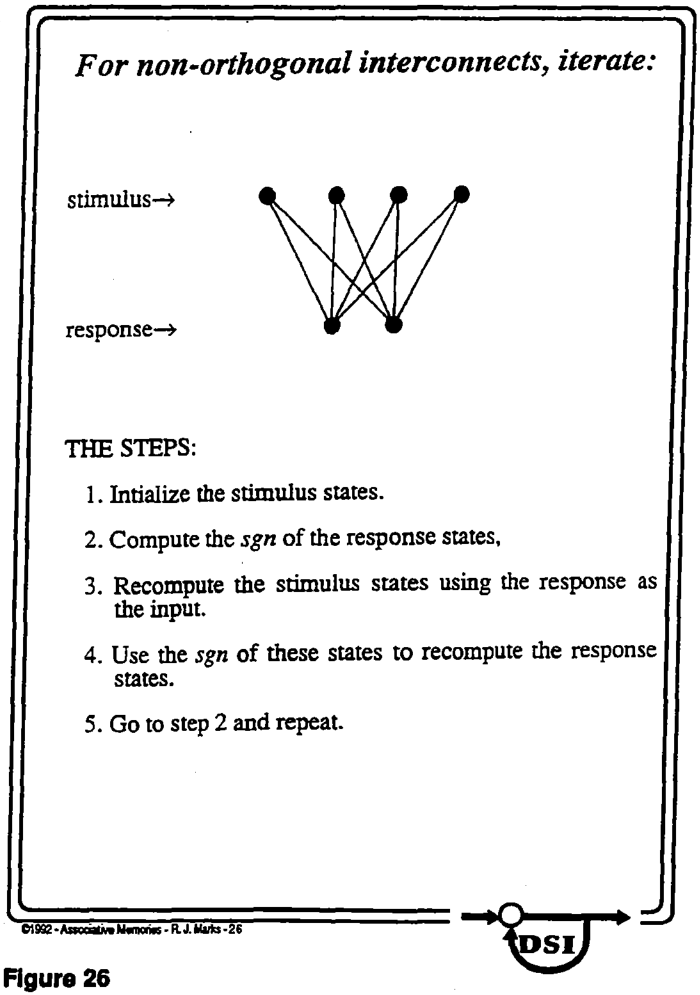
Thus

$$\mathbf{T}\, s_m = \Sigma\, r_n\, s_n^T s_m = c\, r_m$$

where $c = s_m^T s_m$ is a positive constant.

Figure 25

The BAM is a two layer network. The stimulus is propagated to the response level where each neuron is thresholded. The thresholded values are fed back to the stimulus level, thresholded, etc. Iteration occurs until convergence. Iteration in the example shown converges in one iteration because of orthogonality.



For non-orthogonal interconnects, iterate:

THE STEPS:

1. Intialize the stimulus states.
2. Compute the *sgn* of the response states,
3. Recompute the stimulus states using the response as the input.
4. Use the *sgn* of these states to recompute the response states.
5. Go to step 2 and repeat.

Figure 26

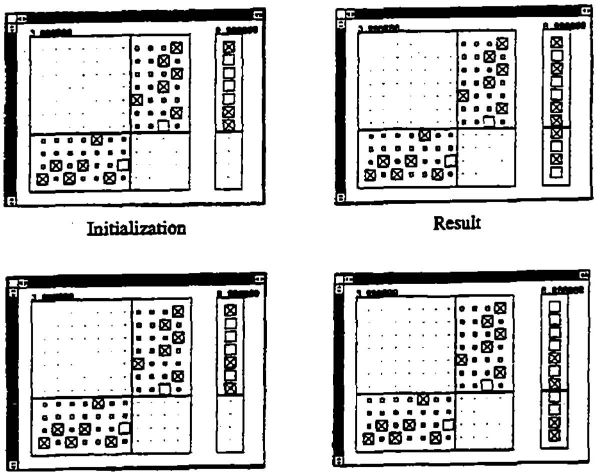
Here are the algorithmic steps for the BAM.

Notes Notes Notes

BAM Example:

$$S = \begin{bmatrix} -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & 1 \end{bmatrix}^T \qquad T = \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}^T$$

Key: ⊠ = -1 , □ = 1



Initialization

Result

Initialization

Result

Note: Twin Image

Figure 27

As with the Hopfield model, the BAM can have twin images. An example is shown here.

ASYNCHRONOUS BAM EXAMPLE [Kosko]

Stored Pairs: (S,E), (M,V) & (G,N). Approximately six neurons per picture. Initialization is random.

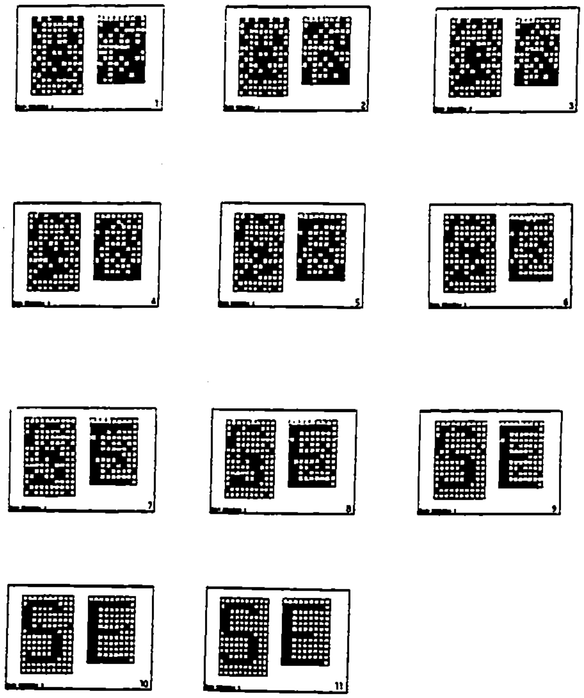
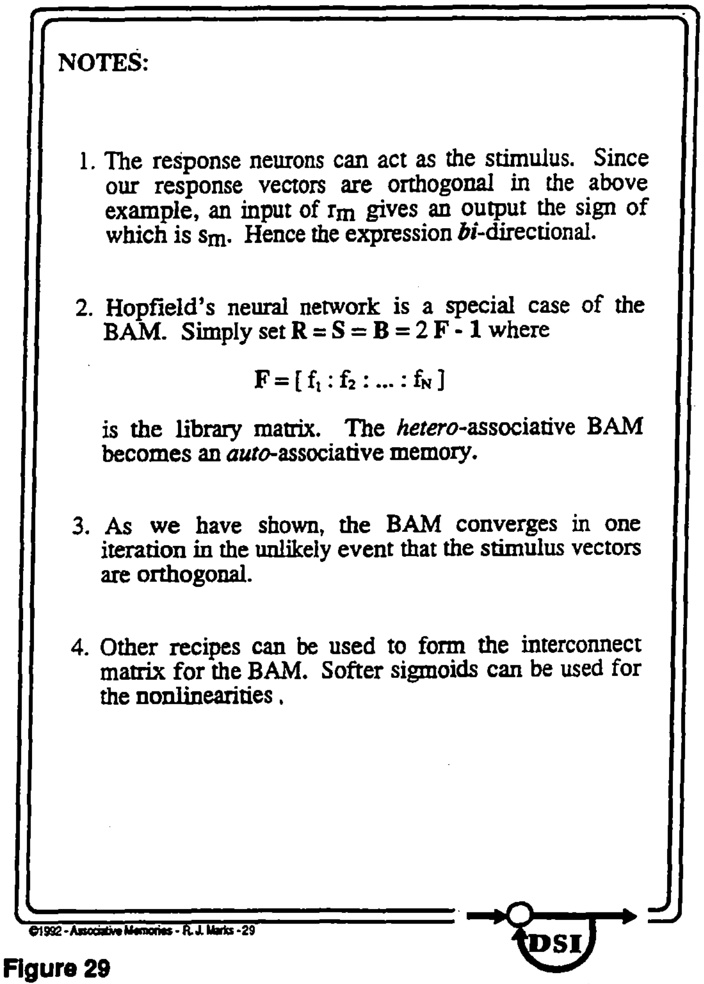


Figure 28

These are snapshots in the evolution of a BAM example.



NOTES:

1. The response neurons can act as the stimulus. Since our response vectors are orthogonal in the above example, an input of r_m gives an output the sign of which is s_m. Hence the expression *bi*-directional.

2. Hopfield's neural network is a special case of the BAM. Simply set $\mathbf{R} = \mathbf{S} = \mathbf{B} = 2\,\mathbf{F} - \mathbf{1}$ where

$$\mathbf{F} = [\, f_1 : f_2 : \dots : f_N \,]$$

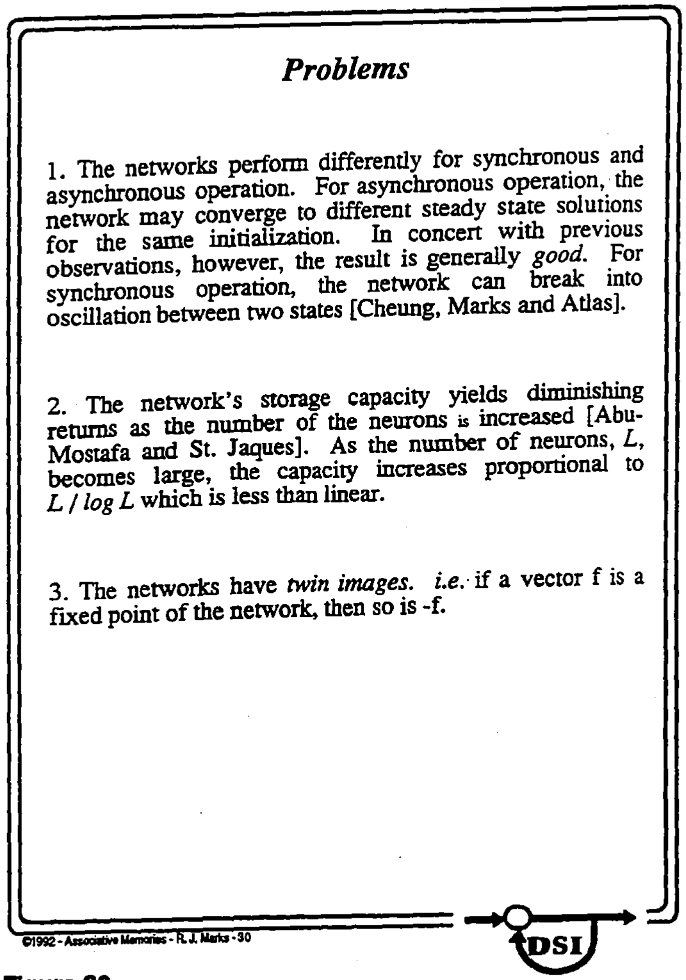
is the library matrix. The *hetero*-associative BAM becomes an *auto*-associative memory.

3. As we have shown, the BAM converges in one iteration in the unlikely event that the stimulus vectors are orthogonal.

4. Other recipes can be used to form the interconnect matrix for the BAM. Softer sigmoids can be used for the nonlinearities.

©1992 - Associative Memories - R. J. Marks - 29

Figure 29



Problems

1. The networks perform differently for synchronous and asynchronous operation. For asynchronous operation, the network may converge to different steady state solutions for the same initialization. In concert with previous observations, however, the result is generally *good*. For synchronous operation, the network can break into oscillation between two states [Cheung, Marks and Atlas].

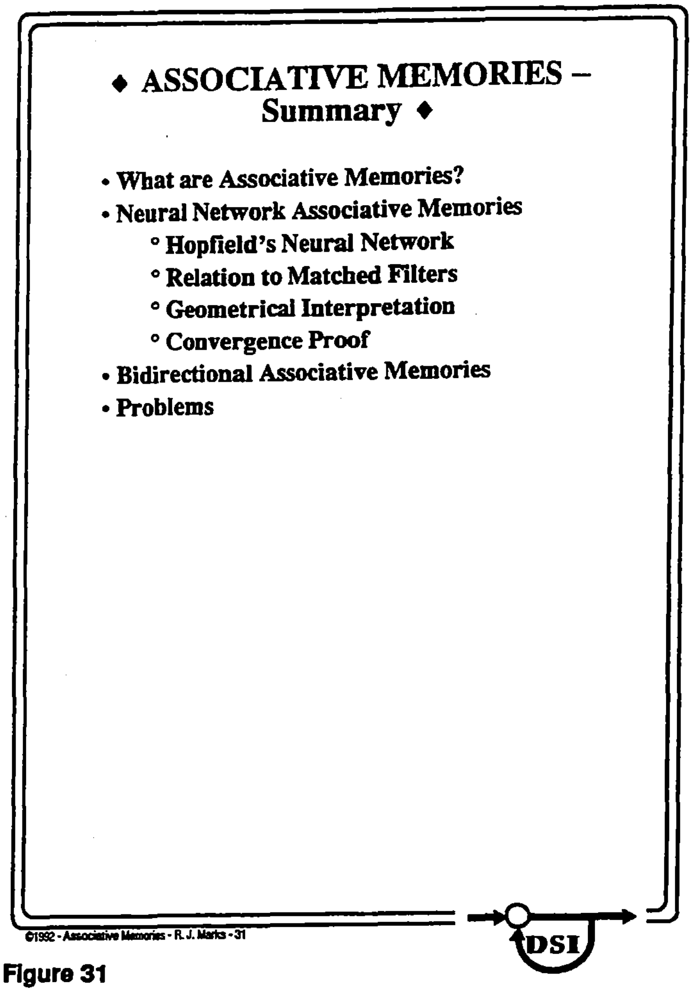
2. The network's storage capacity yields diminishing returns as the number of the neurons is increased [Abu-Mostafa and St. Jaques]. As the number of neurons, L, becomes large, the capacity increases proportional to $L \,/\, log\, L$ which is less than linear.

3. The networks have *twin images. i.e.* if a vector f is a fixed point of the network, then so is -f.

©1992 - Associative Memories - R. J. Marks - 30

Figure 30

In addition, the percentage of false states increases significantly as the size of the net grows.



◆ ASSOCIATIVE MEMORIES – Summary ◆

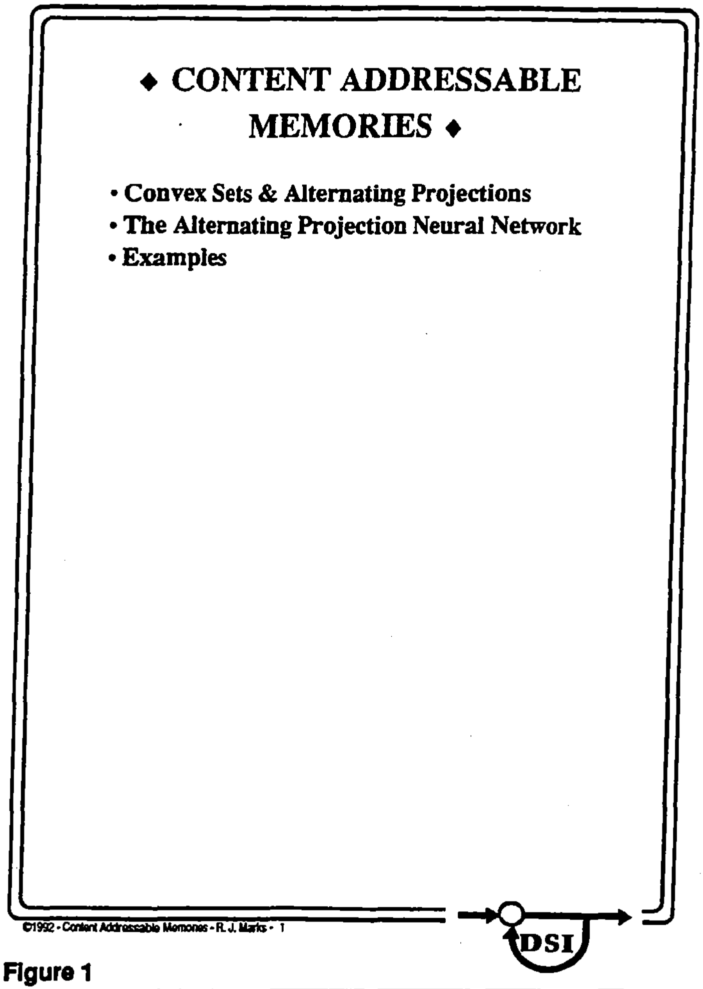
- What are Associative Memories?
- Neural Network Associative Memories
  - ° Hopfield's Neural Network
  - ° Relation to Matched Filters
  - ° Geometrical Interpretation
  - ° Convergence Proof
- Bidirectional Associative Memories
- Problems

©1992 - Associative Memories - R. J. Marks - 31

Figure 31

Notes Notes Notes

Content Addressable Memories

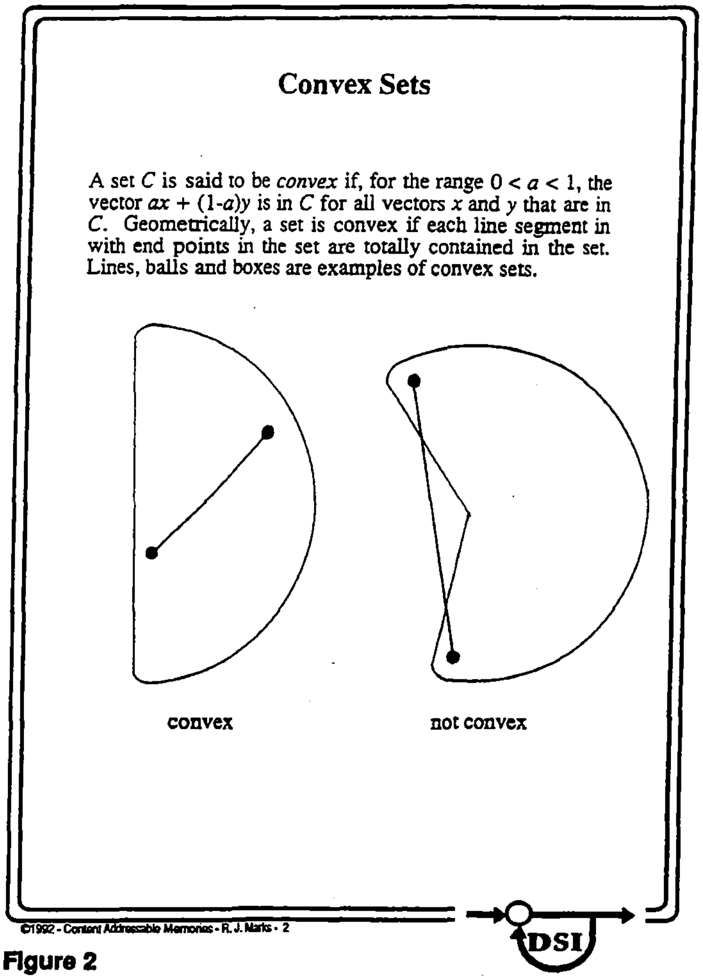


◆ CONTENT ADDRESSABLE MEMORIES ◆

- Convex Sets & Alternating Projections
- The Alternating Projection Neural Network
- Examples

Figure 1

Notes Notes Notes



Convex Sets

A set C is said to be *convex* if, for the range $0 < a < 1$, the vector $ax + (1-a)y$ is in C for all vectors x and y that are in C. Geometrically, a set is convex if each line segment in with end points in the set are totally contained in the set. Lines, balls and boxes are examples of convex sets.

©1992 - Content Addressable Memories - R. J. Marks - 2

Figure 2

A line segment between any two points within a convex set is totally subsumed within that set.

From a given point, a *projection* onto a convex set is to that point in the convex set that is closest:

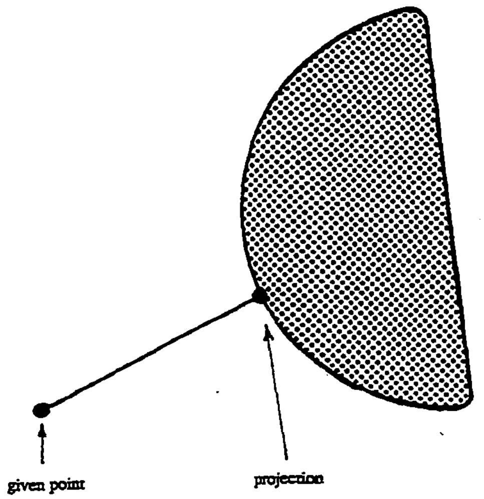
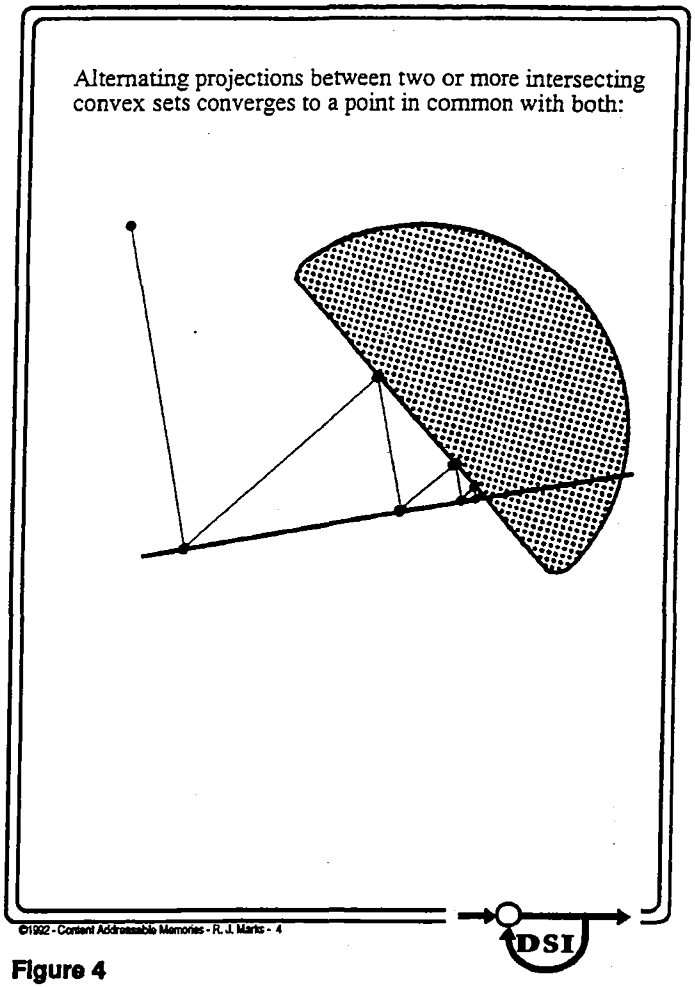


Figure 3



Alternating projections between two or more intersecting convex sets converges to a point in common with both:

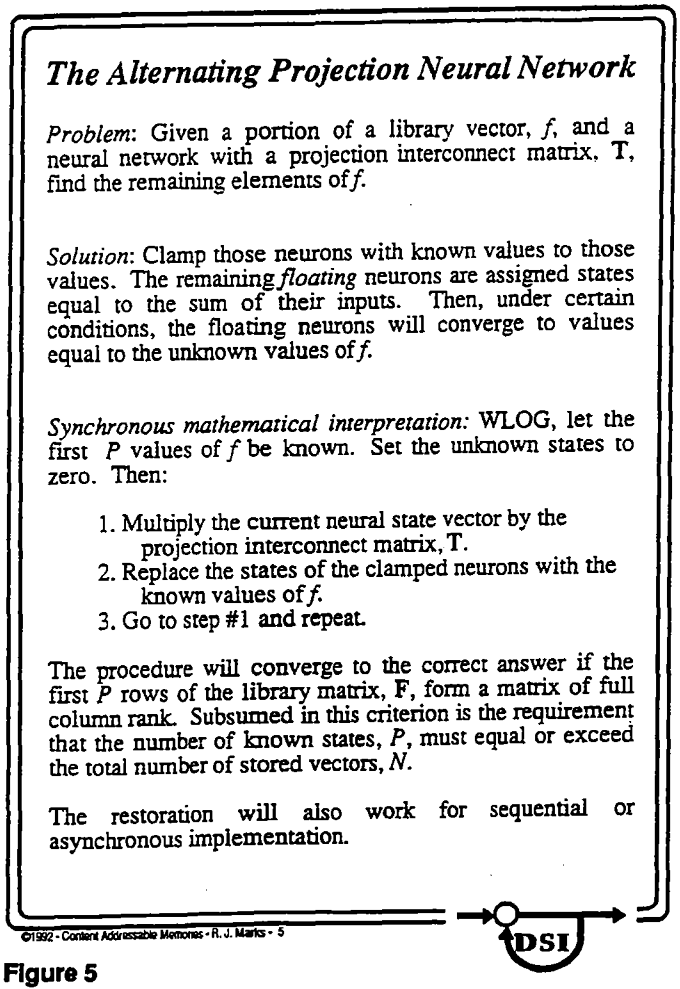
©1992 - Content Addressable Memories - R. J. Marks - 4

Figure 4

This is an illustration of the fundamental theorem of POCS (projection onto convex sets). Alternatingly projecting converges to a point common to the intersection. The final fixed point is a function of the initialization unless, of course, there is only one point of intersection. The other theorems of POCS are:

1. Alternating projection between two non-intersecting points converges to a limit cycle between the sets where each is closest to the other in the mean square sense.
2. If three or more convex sets do not intersect, there is little positive that results from POCS. The limit cycle can be dependant on set ordering and initialization.

Notes Notes Notes



The Alternating Projection Neural Network

Problem: Given a portion of a library vector, *f*, and a neural network with a projection interconnect matrix, **T**, find the remaining elements of *f*.

Solution: Clamp those neurons with known values to those values. The remaining *floating* neurons are assigned states equal to the sum of their inputs. Then, under certain conditions, the floating neurons will converge to values equal to the unknown values of *f*.

Synchronous mathematical interpretation: WLOG, let the first *P* values of *f* be known. Set the unknown states to zero. Then:

1. Multiply the current neural state vector by the projection interconnect matrix, **T**.
2. Replace the states of the clamped neurons with the known values of *f*.
3. Go to step #1 and repeat.

The procedure will converge to the correct answer if the first *P* rows of the library matrix, **F**, form a matrix of full column rank. Subsumed in this criterion is the requirement that the number of known states, *P*, must equal or exceed the total number of stored vectors, *N*.

The restoration will also work for sequential or asynchronous implementation.

DSI

Figure 5

The library matrix is $\mathbf{F} = [f_1 \ f_2 \ ... f_N]$ and the APNN neural network matrix is $\mathbf{T} = \mathbf{F}\,(\mathbf{F}^T\,\mathbf{F})^{-1}\,\mathbf{F}^T$. The vector $\mathbf{T}\,g$ projects the vector g onto the subspace spanned by the library vectors.

Partition Notation:

$$f = [f_P : f_Q]^T$$

Operational Flowgraph:

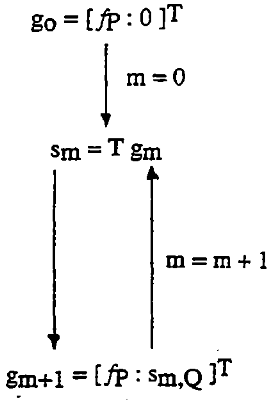


Figure 6

Geometrical Interpretation

The **T** matrix projects onto the subspace spanned by the library vectors. Clamping the known neurons projects onto the linear variety of all vectors with these known values. These two sets intersect at the point of the desired library vector, *f*.

For our previous example of $\mathbf{F} = f_1 = [1 \;\; 1]^T$, starting with the initialization $g_0 = [\, f^P : 0 \,]^T = [\, 1 : 0 \,]^T$, convergence would be as follows:

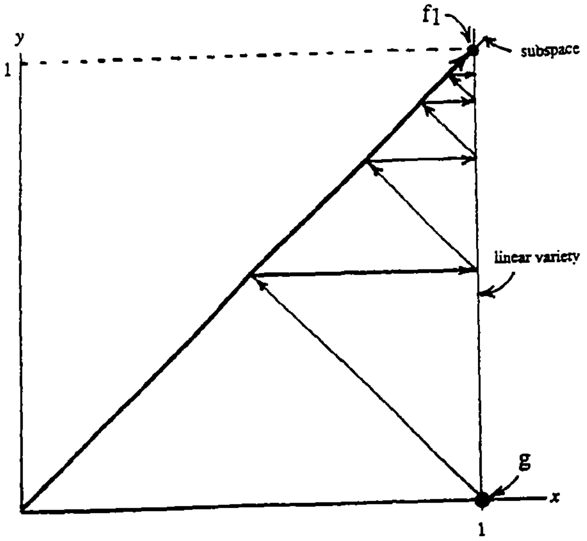
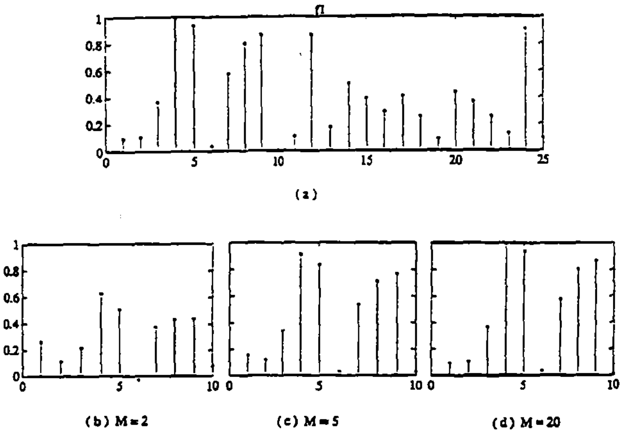


Figure 7

Homogeneous APNN Example:

L = # of neurons = 25
N = # (stochastically chosen) library vectors = 4
P = # clamped neurons = (last) 15



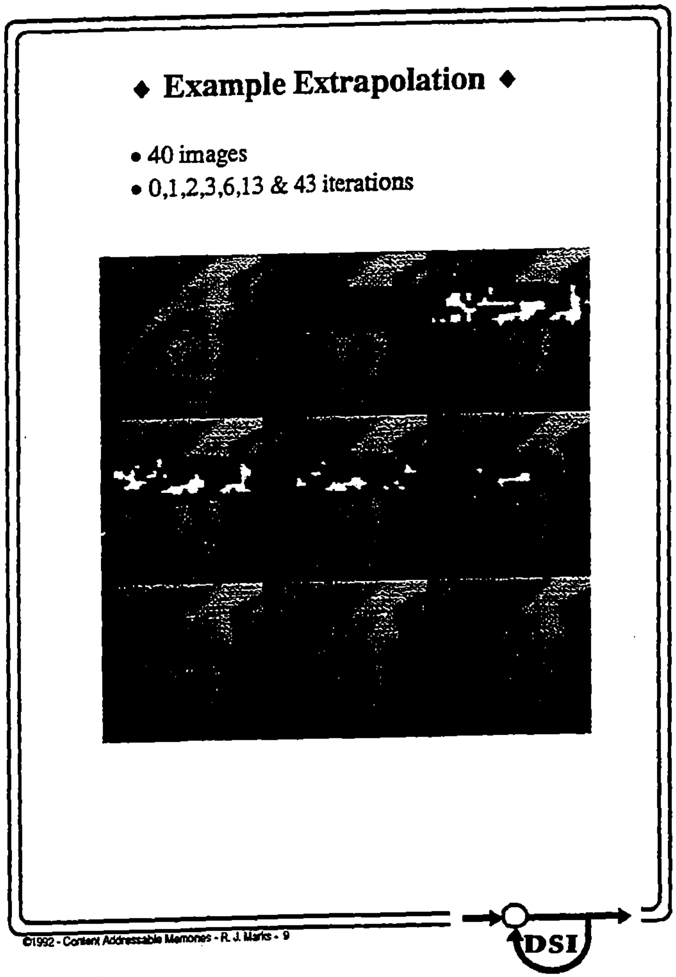
(a)

(b) M=2

(c) M=5

(d) M=20

Figure 8



◆ Example Extrapolation ◆

- 40 images
- 0,1,2,3,6,13 & 43 iterations

©1992 - Content Addressable Memories - R. J. Marks - 9

Figure 9

Result is indistinguishable from the original.

Notes Notes Notes

♦ Generalization ♦

This image was not used as the training data.

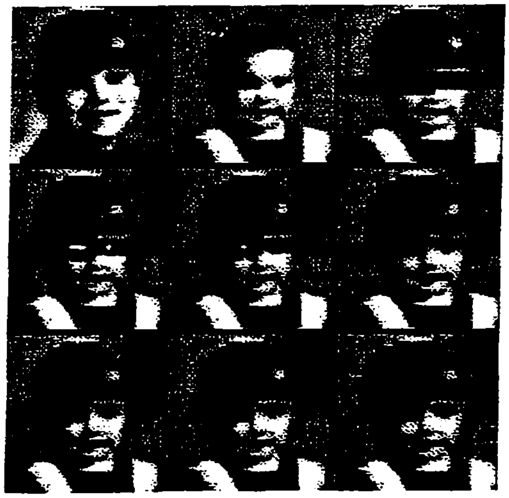
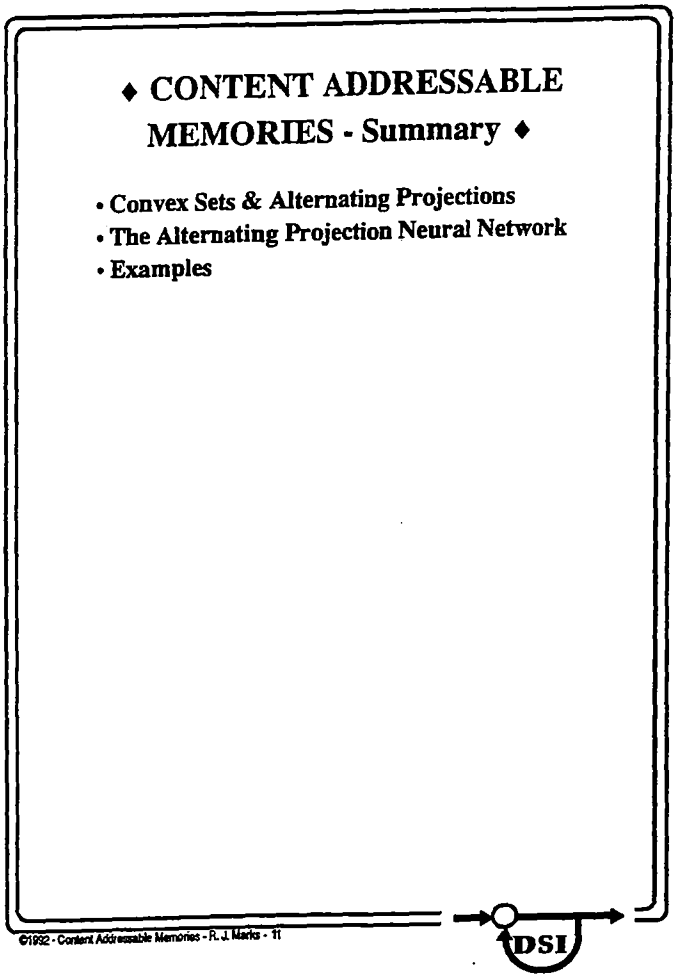


Figure 10

Remarkably, the eyes were generated, even though the neural network was not trained on this image.



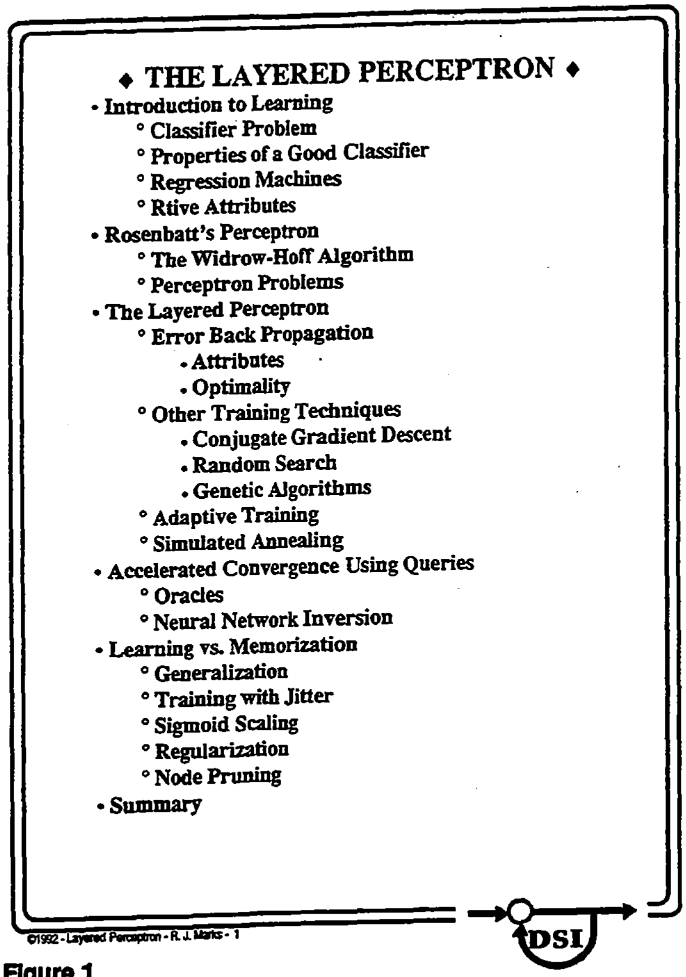
♦ CONTENT ADDRESSABLE MEMORIES - Summary ♦

- Convex Sets & Alternating Projections
- The Alternating Projection Neural Network
- Examples

Figure 11

Notes Notes Notes

The Layered Perceptron



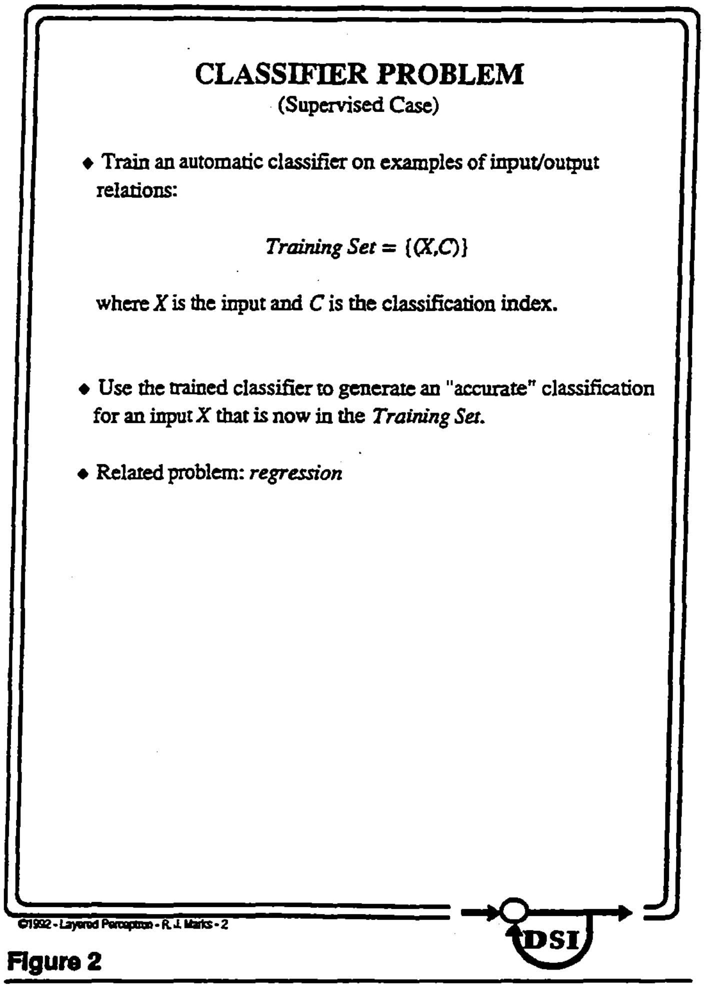
◆ THE LAYERED PERCEPTRON ◆

- Introduction to Learning
    - Classifier Problem
    - Properties of a Good Classifier
    - Regression Machines
    - Rtive Attributes
- Rosenbatt's Perceptron
    - The Widrow-Hoff Algorithm
    - Perceptron Problems
- The Layered Perceptron
    - Error Back Propagation
        - Attributes
        - Optimality
    - Other Training Techniques
        - Conjugate Gradient Descent
        - Random Search
        - Genetic Algorithms
    - Adaptive Training
    - Simulated Annealing
- Accelerated Convergence Using Queries
    - Oracles
    - Neural Network Inversion
- Learning vs. Memorization
    - Generalization
    - Training with Jitter
    - Sigmoid Scaling
    - Regularization
    - Node Pruning
- Summary

©1992 - Layered Perceptron - R. J. Marks - 1

Figure 1

Notes Notes Notes



CLASSIFIER PROBLEM

(Supervised Case)

- Train an automatic classifier on examples of input/output relations:

$$Training\ Set = \{(X,C)\}$$

where X is the input and C is the classification index.

- Use the trained classifier to generate an "accurate" classification for an input X that is now in the *Training Set.*

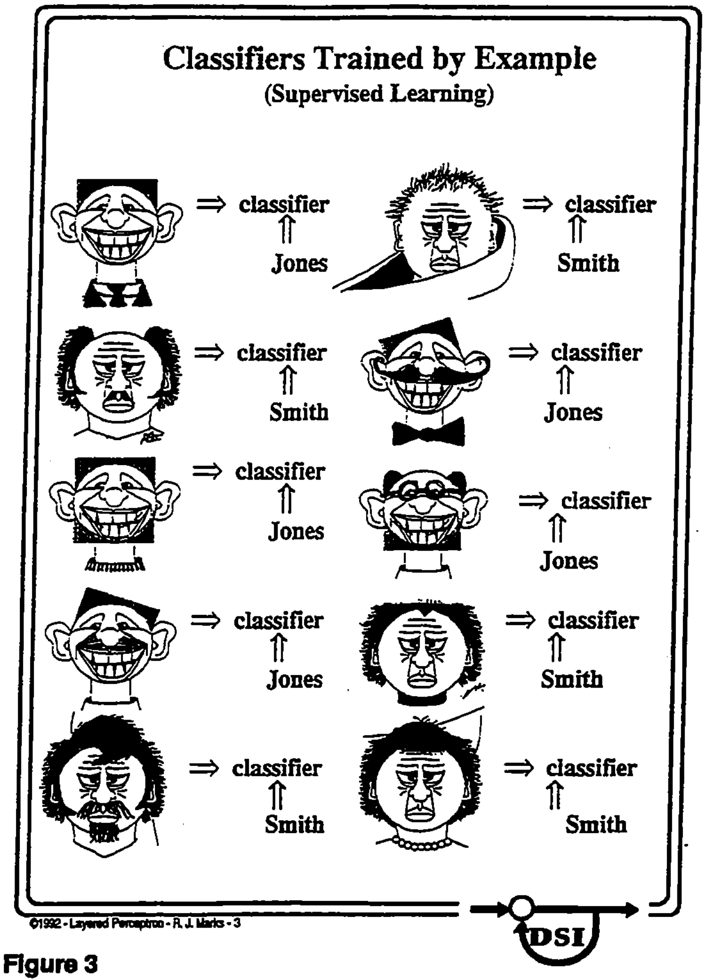
- Related problem: *regression*

©1992 - Layered Perceptron - R. J. Marks - 2

Figure 2

A set of training data pairs is given. The input vectors, X, are associated with corresponding output vectors, C. The problem of the classifier or regression machine is to learn from this data so that, when subjected to test input data which it has not yet seen, it can give a good estimate of the corresponding output (*e.g.* class).

The layered perceptron is trained with training data. For the load forecasting problem, for example, input training data might consist of a number of temperatures and the output is the forecasted load. Data from the previous year, for example, can be used. Once trained, the layered perceptron, presented with the temperatures of the current day will provide, as output, a forecast of the load for the next day.

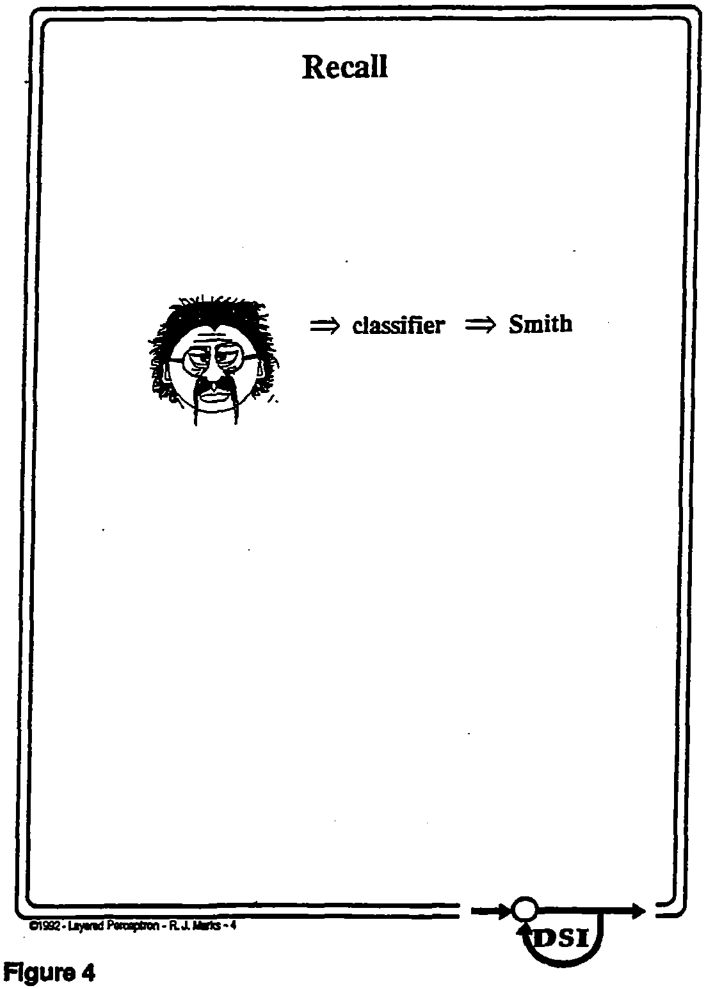


Classifiers Trained by Example

(Supervised Learning)

Figure 3

The layered perceptron is an example of a classifier or, when the output is continuous, a regression machine, which is trained by data. It is also supervised. In the example shown, for example, the classifier is told whether the input is a Smith or a Jones.



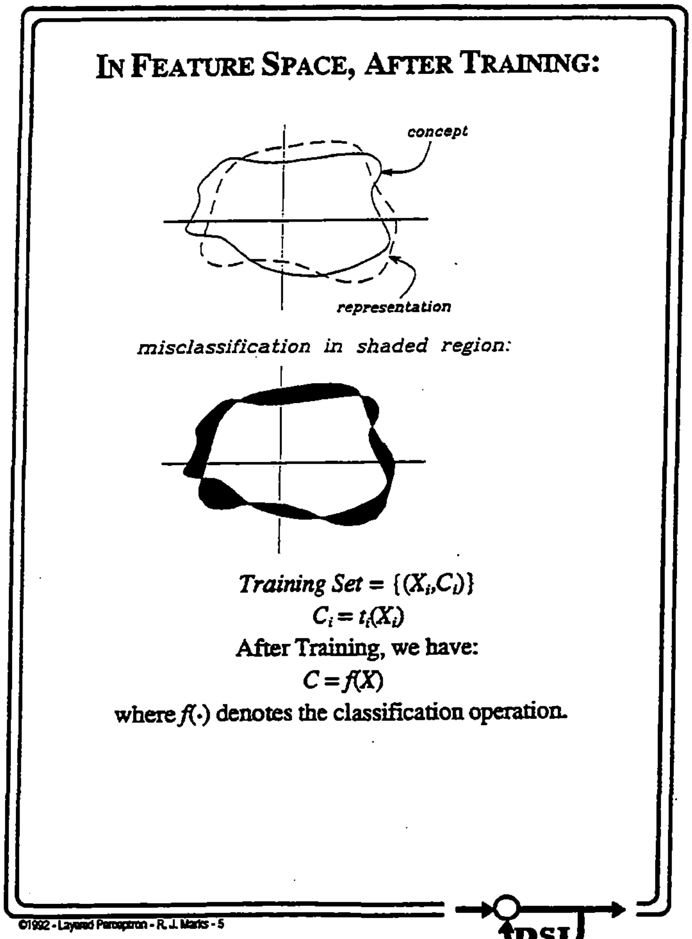
Recall

⇒ classifier ⇒ Smith

©1992 - Layered Perceptron - R. J. Marks - 4

Figure 4

Once trained, a good classifier or regression machine will properly respond to *test data*. For proper performance, the test data and the training data should be different, albeit from the same statistical source.



In Feature Space, After Training:

misclassification in shaded region:

Training Set = $\{(X_i, C_i)\}$

$C_i = t_i(X_i)$

After Training, we have:

$C = f(X)$

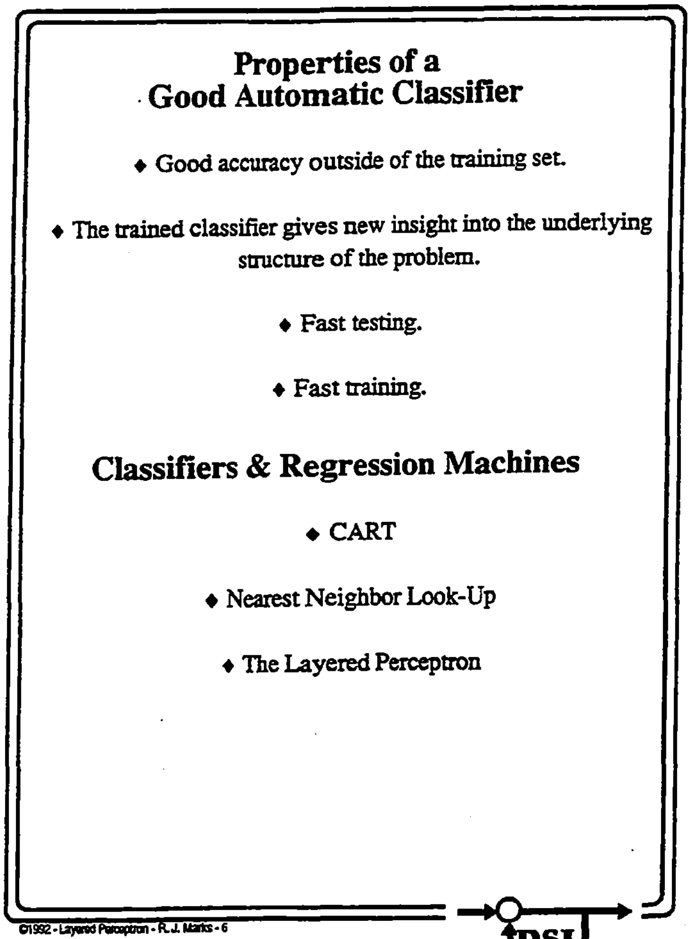
where $f(\cdot)$ denotes the classification operation.

©1992 - Layered Perceptron - R. J. Marks - 5

Figure 5

Consider the two dimensional closed curve shown here. The solid line represents the unknown *concept*. Within the curve we wish to classify the ordered pair as one. Outside, the classification is zero. Based on available training data, the classifier tries to learn the classification boundary. The estimate of the classification boundary is the *representation* shown by the dashed curve. If the training data noise is uncorrupted by uncertainty, we would expect the representation boundary to approach the concept boundary as the cardinality of the training data set increases. For a finite size training set, the resulting probability of error is equal to the probability of false classification. This is equal to the shaded area.

The concept, shown by the solid line, is to be learned. The broken line denotes the learned representation. The probability of error is equal to the probability a point is chosen is the shaded area. If the training data is chosen randomly, then a decrease in the probability of error also requires a decrease in the probability of learning something new.



Properties of a Good Automatic Classifier

- Good accuracy outside of the training set.
- The trained classifier gives new insight into the underlying structure of the problem.
- Fast testing.
- Fast training.

Classifiers & Regression Machines

- CART
- Nearest Neighbor Look-Up
- The Layered Perceptron

©1992 - Layered Perceptron - R. J. Marks - 6

Figure 6

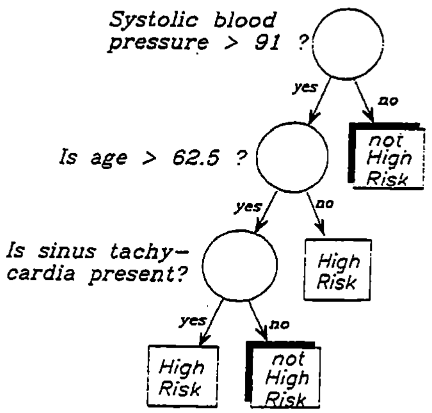
Do layered perceptrons perform better than other classifiers and regression machines? By comparison with some other high performance classifiers and regression machines, the current answer is yes. Possibly there is an underlying limit of performance placed on all classifiers and regression machines that cutting edge algorithms are approaching. If so, then secondary performance attributes such as training speed and implementation ease must be addressed as primary.

Other artificial neural networks have fallen from favor in an application sense because, quite simply, they are not competitive with other more conventional approaches. The same question must be posed in regard to the layered perceptron. Does the layered perceptron preform better than other classifiers or regression machines programmed from examples using supervised learning? Although abstract analysis of this question may be possible in some cases, it must ultimately be answered in regard to actual data. Comparisons of the layered perceptron have been performed with *classification and regression trees* (CART) and *nearest* *neighbor lookup* for such problems as speech, power security assessment and load forecasting and, in each case, have shown the layered perceptron to perform better in terms of classification or regression accuracy. Both of these competing algorithms can be implemented using parallel processing.

Notes Notes Notes

CART

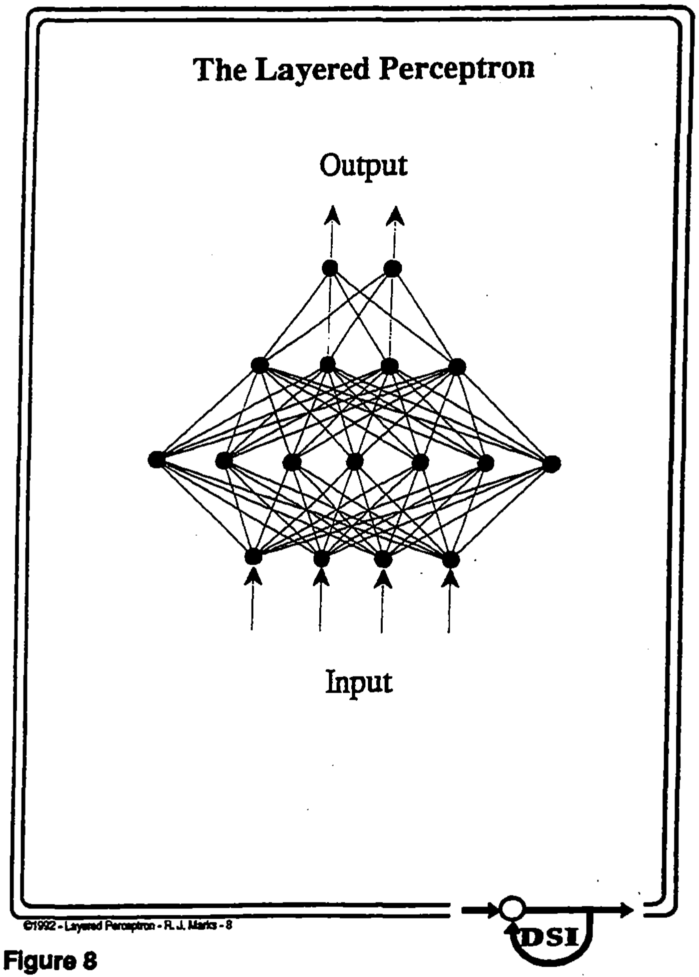
Example Decision Tree:



©1992 - Layered Perceptron - R. J. Marks - 7

Figure 7

CART (Classification and regression trees) are decision trees that are trained by example. In its fundamental form, the feature space is initially divided into planes that were perpendicular to the axes. It is this form of CART used here. In a higher order form of CART, these planes can be oriented at angles. The higher order form of CART has given preliminary results that are nearly indistinguishable in performance to the layered perceptron. There also exist other high power paradigms, such as *projection pursuit* to which the layered perceptron performance must ultimately be compared.



The Layered Perceptron

©1992 - Layered Perceptron - R. J. Marks - 8

Figure 8

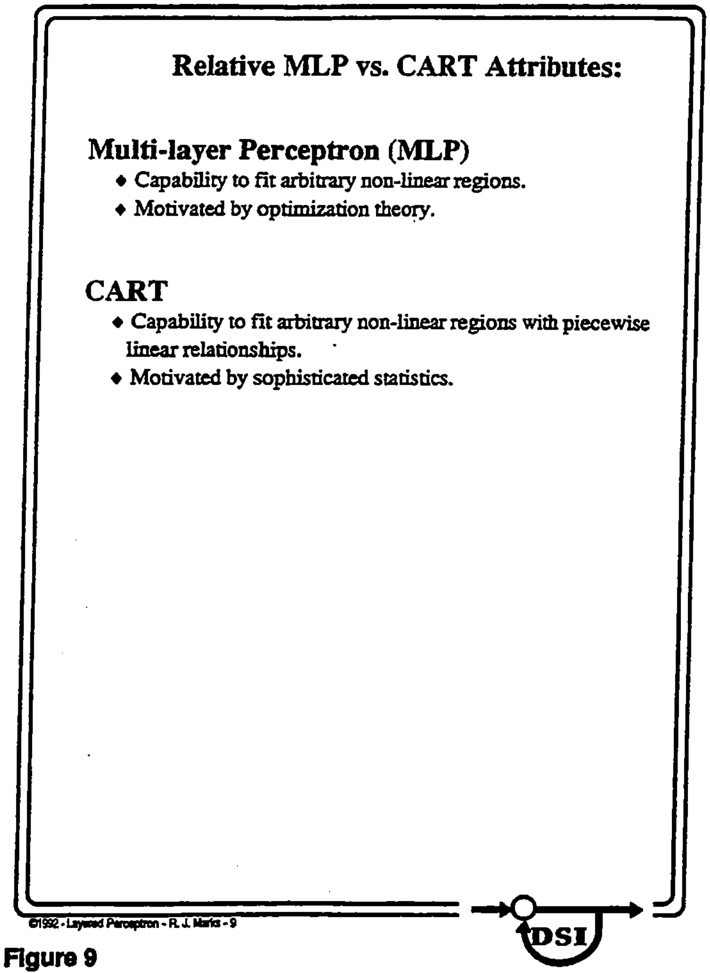
Currently, the artificial neural network most commonly used is the layered perceptron. Although convention varies, the interconnects from the input to the hidden neurons along with the hidden neurons constitute a layer. The hidden to output interconnects with the output neurons constitute a second layer. Thus, the perceptron shown here has three layers. In our treatment, we do not consider the input nodes to be neurons.

Layered perceptrons are trained by numerical data, in contrast, for example, to expert systems that are trained by rules. The layered perceptron operates in two modes; training and test. In the training mode, a set of representative *training data* is used to adjust the weights of the neural interconnects. Once these weights have been determined, the neural network is said to be trained. In the test mode, the trained neural network is activated by *test data*. The response of the layered perceptron should then be representative of the data by which it was trained. Typically, the test and training data are different sets. As we will dis-

Notes Notes Notes

cuss in the section on learning, training a machine to respond properly to the same data on which it is trained is not learning, but is rather memorization.

A layered perceptron can be used as either a classifier or a regression machine. As a classifier, the layered perceptron categorizes the input into two or more categories. In power system security assessment, for example, the trained perceptron will categorize the power either secure or insecure in accordance to the current system states. For regression applications, the output or outputs of the layered perceptron take on continuous values. Power load forecasting is an example of a regression application. Here, the output of the neural network corresponds to the forecasted load.



Relative MLP vs. CART Attributes:

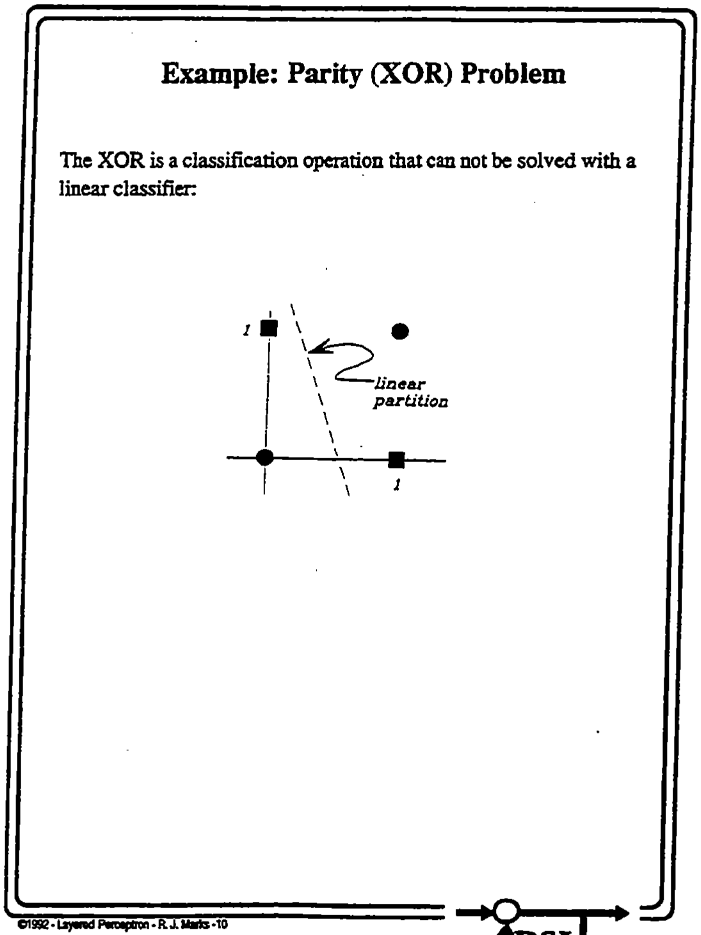
Multi-layer Perceptron (MLP)

- ◆ Capability to fit arbitrary non-linear regions.
- ◆ Motivated by optimization theory.

CART

- ◆ Capability to fit arbitrary non-linear regions with piecewise linear relationships.
- ◆ Motivated by sophisticated statistics.

Figure 9



Example: Parity (XOR) Problem

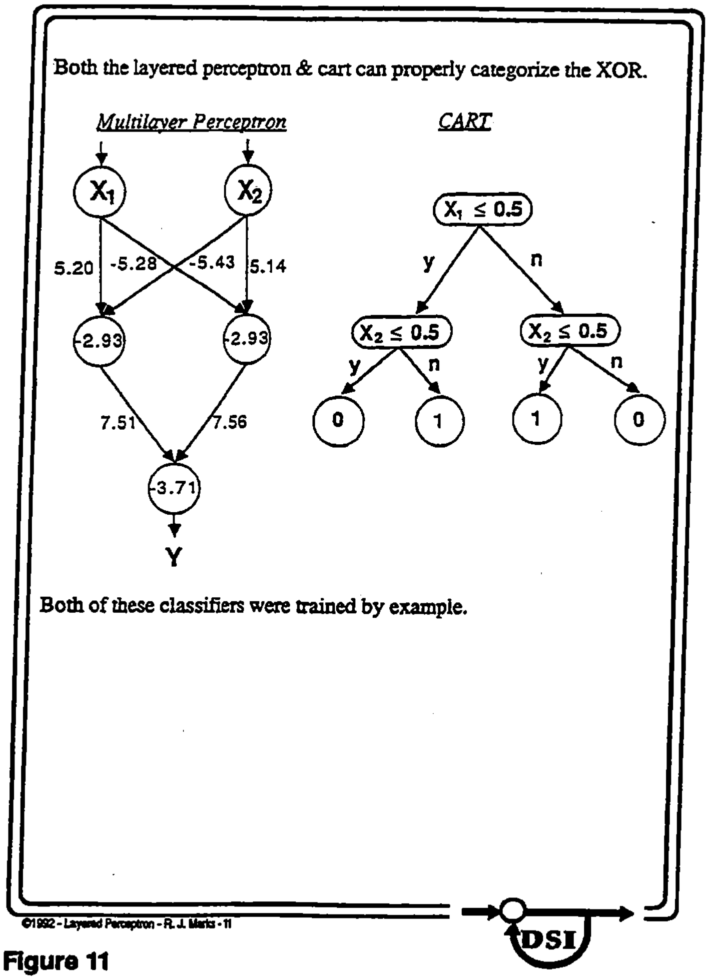
The XOR is a classification operation that can not be solved with a linear classifier:

©1992 - Layered Perceptron - R. J. Marks - 10

DSI

Figure 10

Linear classifiers cannot even handle a simple toy problem like the XOR problem illustrated here. This was pointed out in the book ***Perceptrons*** by Minsky & Papert as a quite negative attribute of the linear perceptron.



Both the layered perceptron & cart can properly categorize the XOR.

Both of these classifiers were trained by example.

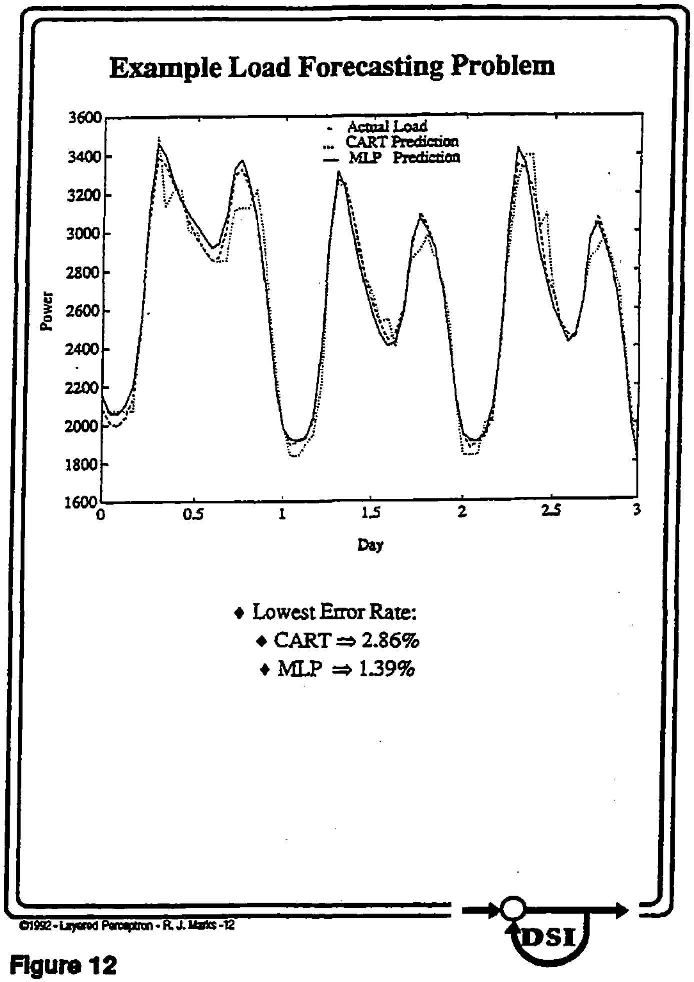
©1992 - Layered Perceptron - R. J. Marks - 11

Figure 11

Both the layered perceptron and the CART classifier were trained by example. Both give the correct answers. Note, however, that CART gives a quite intuitive decision process.

Notes Notes Notes

Notes Notes Notes



Example Load Forecasting Problem

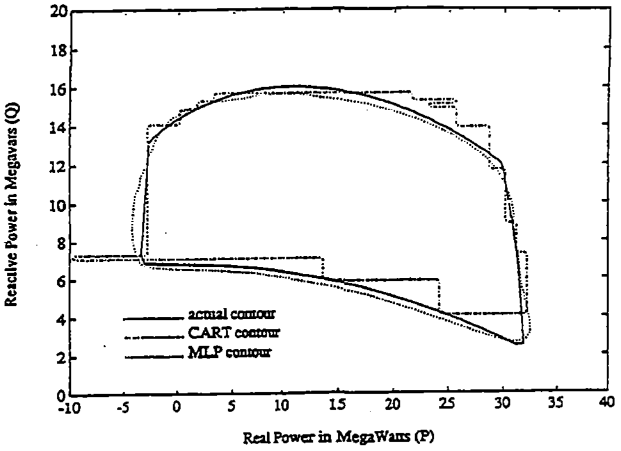
- Lowest Error Rate:
  - CART ⇒ 2.86%
  - MLP ⇒ 1.39%

©1992 - Layered Perceptron - R. J. Marks - 12

Figure 12

In power load forecasting current and forecasted temperature and current load demand is used to forecast the future power load demand. For this problem, the worst perceptron performance was an error of 1.78%. CART produced an error of 1.68%. These and other forecasting results will be discussed in depth later in the course.

Example Security Assessment Comparison



Lowest Error Rate:

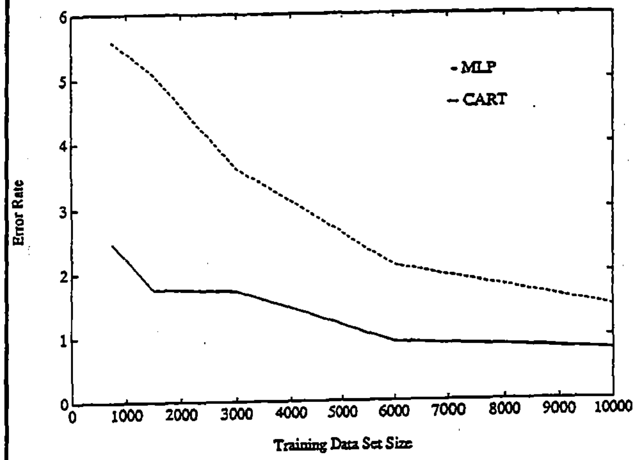
- CART ⇒ 1.46%
- MLP ⇒ 0.78%

The layered perceptron has also outperformed CART using real training data in speech and forecasting.

Figure 13

In the power security assessment problem, the state of a power system is determined to be safe or in jeopardy. Applied to this problem, the perceptron again had a lower error rate - 0.78% to 1.46%. For speaker independent vowel classification, the perceptron again had a higher correct classification rate than CART, 47.4% to 38.2%. Specifics will be addressed more at length later in the course.

The Error Rate: MLP vs. CART

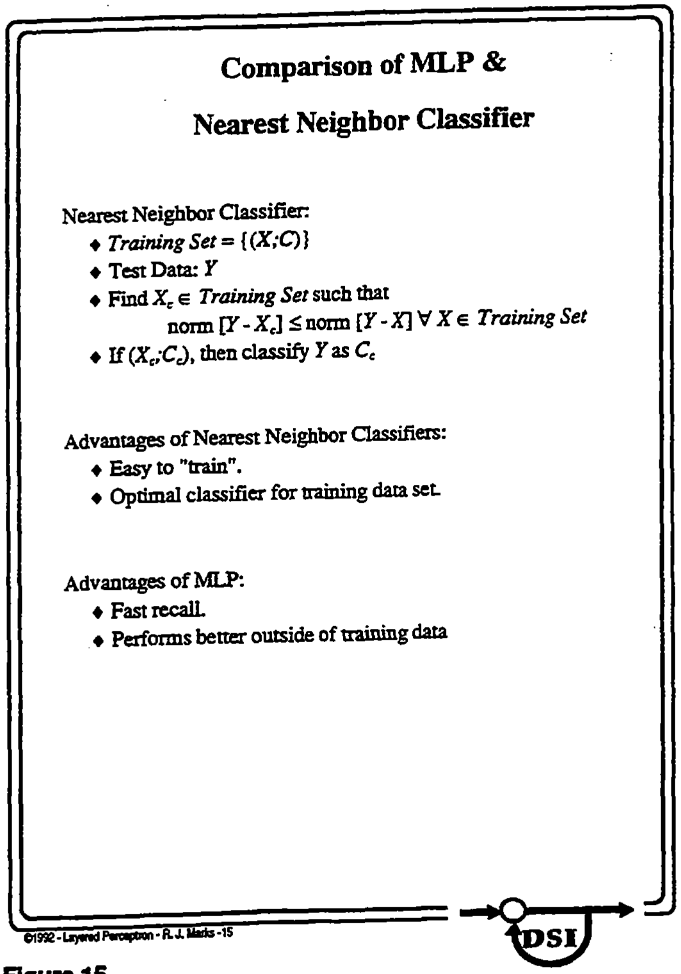


©1992 - Layered Perceptron - R. J. Marks - 14

DSI

Figure 14

The layered perceptron consistently outperformed CART in test data error as a function of data cardinality.



Comparison of MLP & Nearest Neighbor Classifier

Nearest Neighbor Classifier:

- *Training Set* = $\{(X;C)\}$
- Test Data: Y
- Find $X_c \in$ *Training Set* such that
  norm $[Y - X_c] \leq$ norm $[Y - X]\ \forall\ X \in$ *Training Set*
- If $(X_c;C_c)$, then classify Y as C_c

Advantages of Nearest Neighbor Classifiers:

- Easy to "train".
- Optimal classifier for training data set.

Advantages of MLP:

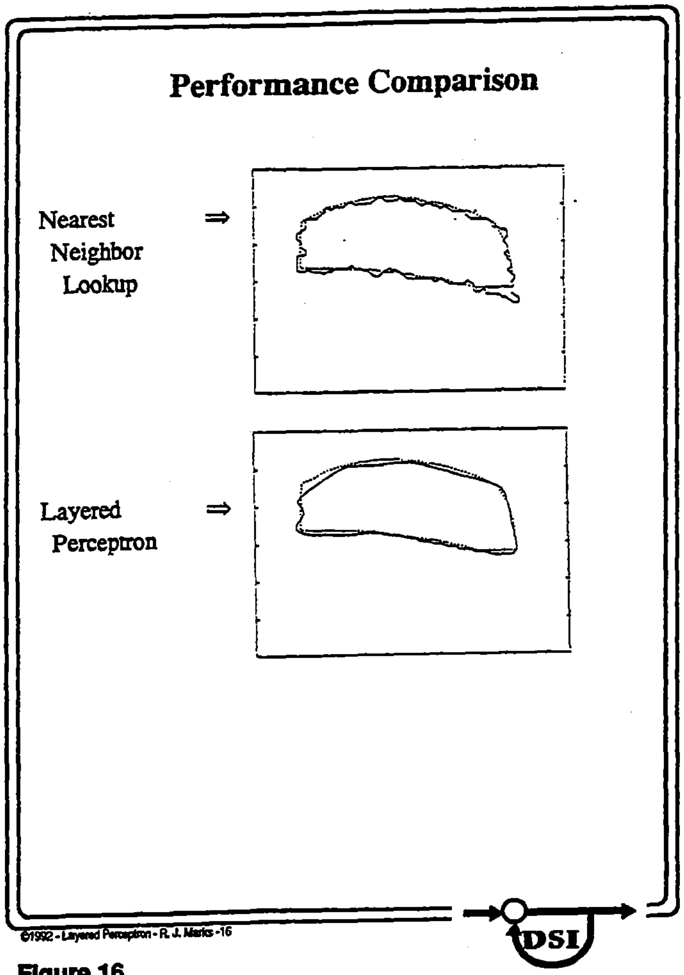
- Fast recall.
- Performs better outside of training data

©1992 - Layered Perceptron - R. J. Marks -15

DSI

Figure 15

Notes Notes Notes



Performance Comparison

Nearest Neighbor Lookup ⇒

Layered Perceptron ⇒

©1992 - Layered Perceptron - R. J. Marks - 16

Figure 16

In comparison with nearest neighbor lookup, the layered perceptron was shown to interpolate much more smoothly and with greater accuracy for the problem of power security assessment.

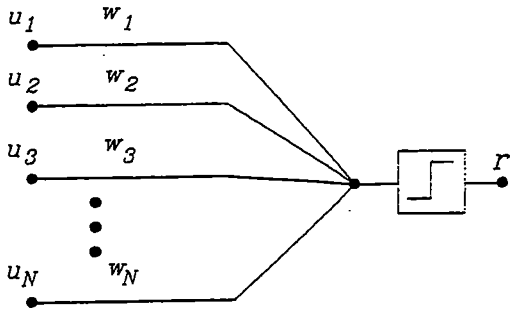
Rosenblatt's Perceptron

(1957)

$$\mathbf{w} = [\ w_1\ w_2\ w_3\ \dots\ w_N\ ]^T$$

$$\mathbf{u} = [\ u_1\ u_2\ u_3\ \dots\ u_N\ ]^T,\ u_1 = 1$$

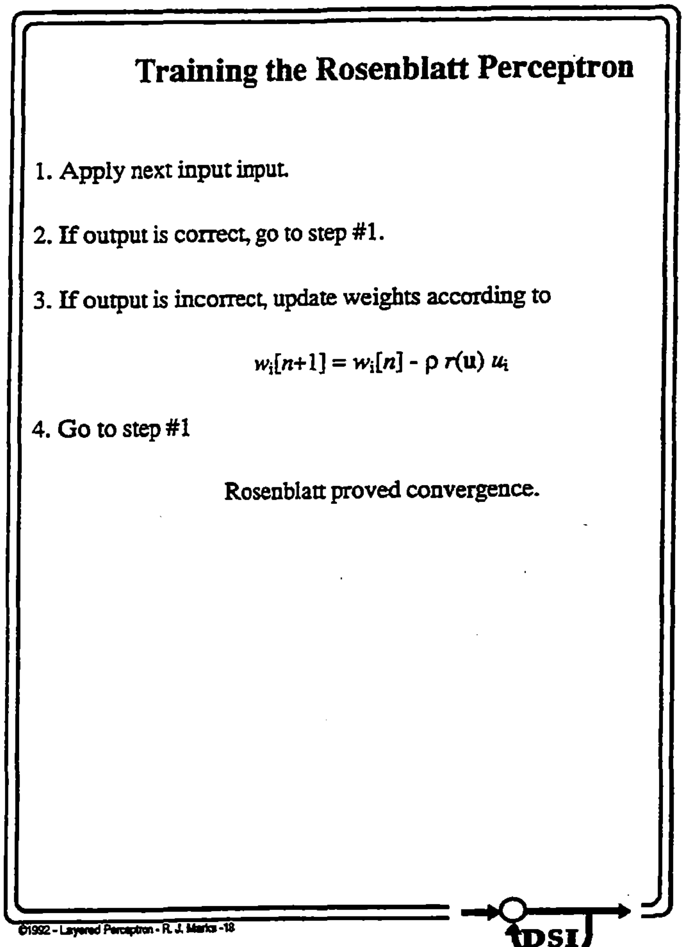
$$r(\mathbf{u}) = \mathrm{sgn}[\ \mathbf{w}^T \mathbf{u}\ ] = \mathrm{sgn}[\ w_1 u_1 + w_2 u_2 + \dots + w_N u_N\ ]$$



DSI

Figure 17

Rosenblatt's perceptron is a linear classifier. The sum of products of the inputs times the neural weights is thresholded to decide between one of two classes the neural network was trained to recognize.



Training the Rosenblatt Perceptron

1. Apply next input input.

2. If output is correct, go to step #1.

3. If output is incorrect, update weights according to

$$w_i[n+1] = w_i[n] - \rho \, r(\mathrm{u}) \, u_i$$

4. Go to step #1

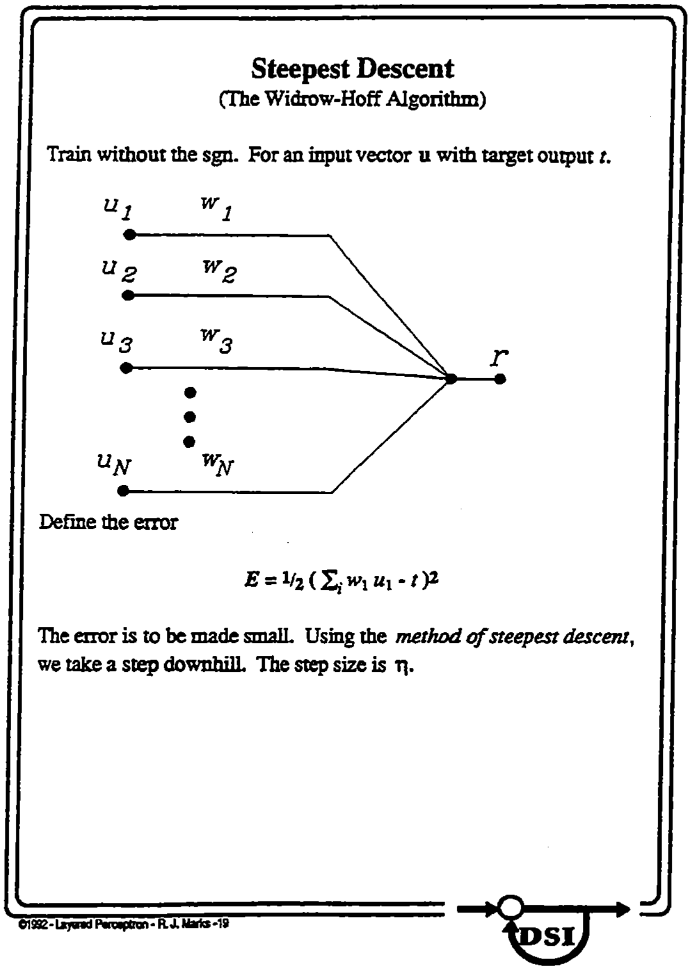
Rosenblatt proved convergence.

©1992 - Layered Perceptron - R. J. Marks -18

DSI

Figure 18

"If it's not broke, don't fix it". If the training data gives you the correct result, don't change the weights. If the perceptron does give you the wrong result, the weights are moved towards the direction that will give the desired result.



Steepest Descent

(The Widrow-Hoff Algorithm)

Train without the sgn. For an input vector u with target output t.

Define the error

$$E = 1/_2 \left(\sum_i w_1 u_1 - t \right)^2$$

The error is to be made small. Using the *method of steepest descent*, we take a step downhill. The step size is η.

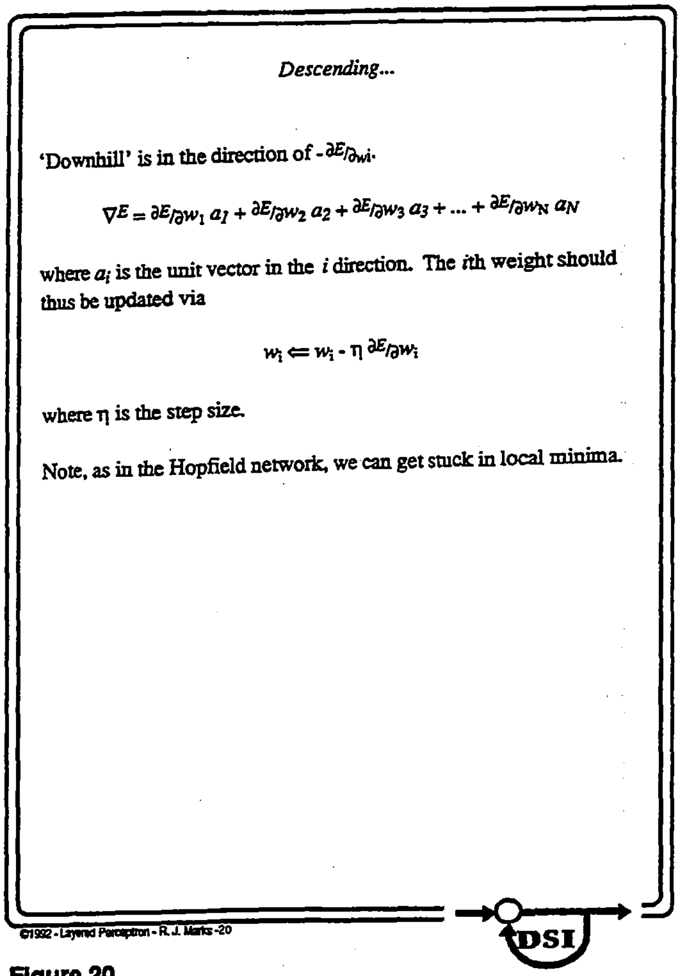
©1992 - Layered Perceptron - R.J. Marks -19

DSI

Figure 19

The training procedure for both the linear and layered perceptrons is based on the Widrow-Hoff algorithm, or steepest descent. The error of the neural net is a metric of the distance between what you have and what you want. For a given data set, the error forms a surface in weight space. At the current point in the weight space, we compute the steepest slope and take a step in that direction thereby changing our location in weight space. The process is repeated until an acceptably low error is obtained.

The perceptron trained withe steepest descent is referred to as ADALINE for *adaptive linear neuron.*



Descending...

'Downhill' is in the direction of $-{}^{\partial E}/_{\partial w_i}$.

$$\nabla E = {}^{\partial E}/_{\partial w_1}\, a_1 + {}^{\partial E}/_{\partial w_2}\, a_2 + {}^{\partial E}/_{\partial w_3}\, a_3 + \ldots + {}^{\partial E}/_{\partial w_N}\, a_N$$

where a_i is the unit vector in the i direction. The ith weight should thus be updated via

$$w_i \Leftarrow w_i - \eta\, {}^{\partial E}/_{\partial w_i}$$

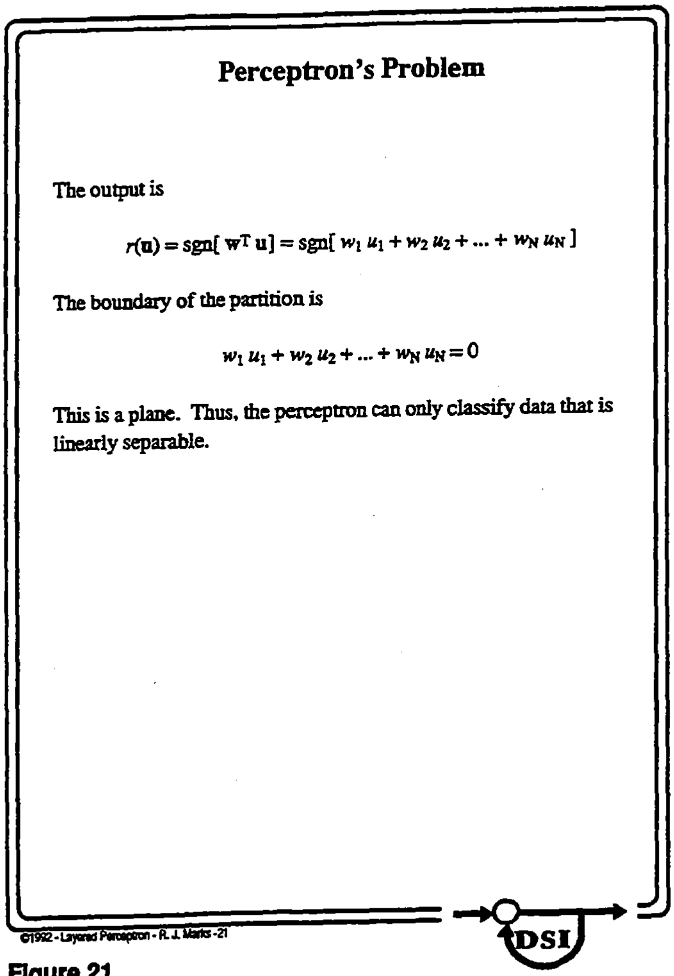
where η is the step size.

Note, as in the Hopfield network, we can get stuck in local minima.

©1992 - Layered Perceptron - R. J. Marks -20

Figure 20

These are the equations describing steepest descent. The step size, η, is a parameter of the search for the minimum. In more sophisticated applications, the step size adapts.



Perceptron's Problem

The output is

$$r(\mathbf{u}) = \text{sgn}[\ \mathbf{w}^T \mathbf{u}\ ] = \text{sgn}[\ w_1 u_1 + w_2 u_2 + ... + w_N u_N\ ]$$

The boundary of the partition is

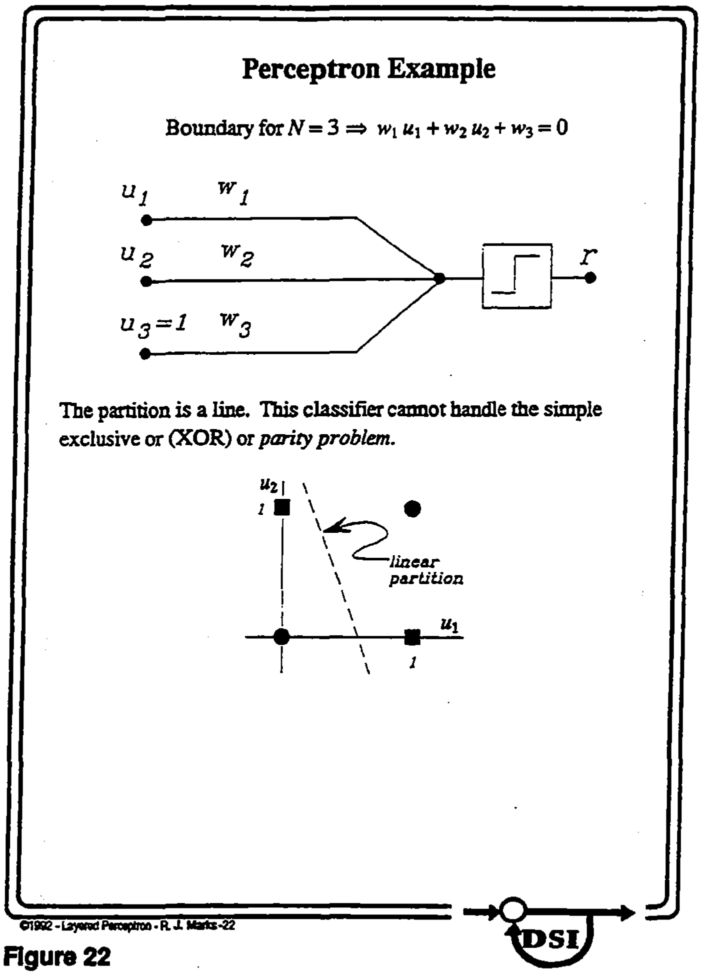
$$w_1 u_1 + w_2 u_2 + ... + w_N u_N = 0$$

This is a plane. Thus, the perceptron can only classify data that is linearly separable.

©1992 - Layered Perceptron - R. J. Marks -21

Figure 21

As these equations show, the perceptron is a linear classifier. It can only classify data that is separated by a plane.



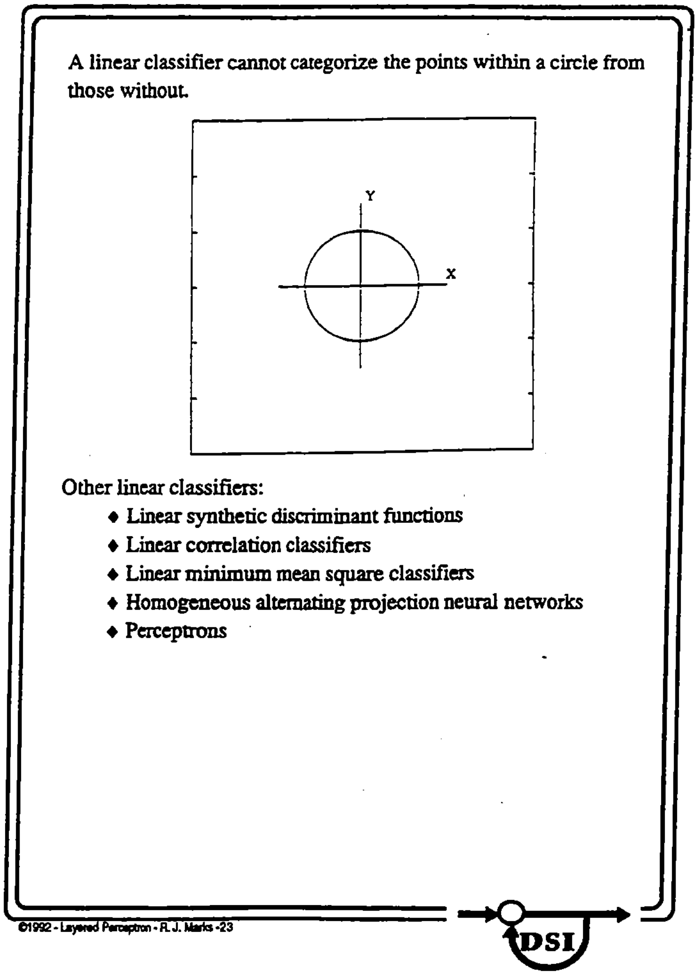
Perceptron Example

Boundary for $N = 3 \Rightarrow w_1 u_1 + w_2 u_2 + w_3 = 0$

The partition is a line. This classifier cannot handle the simple exclusive or (XOR) or *parity problem.*

Figure 22

A Rosenblatt perceptron cannot perform a simple XOR.



A linear classifier cannot categorize the points within a circle from those without.

Other linear classifiers:

- Linear synthetic discriminant functions
- Linear correlation classifiers
- Linear minimum mean square classifiers
- Homogeneous alternating projection neural networks
- Perceptrons

©1992 - Layered Perceptron - R. J. Marks -23

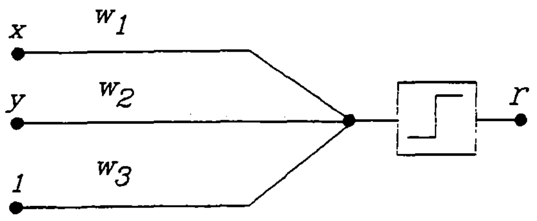
Figure 23

There are a number of linear classifiers. All of them are constrained to classifying linearly separable data and, therefore, cannot perform quite simple problems.

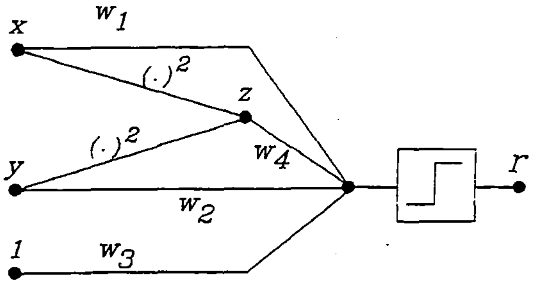
A solution: introduce nonlinearities into the perceptron.

Example:

- Before



- After, $z^2 = x^2 + y^2$



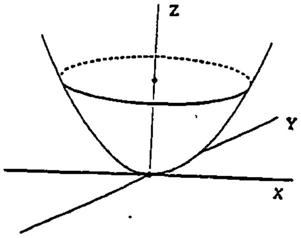
DSI

Figure 24

To make a linear classifier nonlinear, we simply reduce nonlinearities. The functional link, an example of which is shown here, allows for nonlinear classification. The degrees of freedom in the available surfaces become more diverse.

Notes Notes Notes

Using this augmented version of the perceptron, we can now distinguish the points inside a circle from those without.

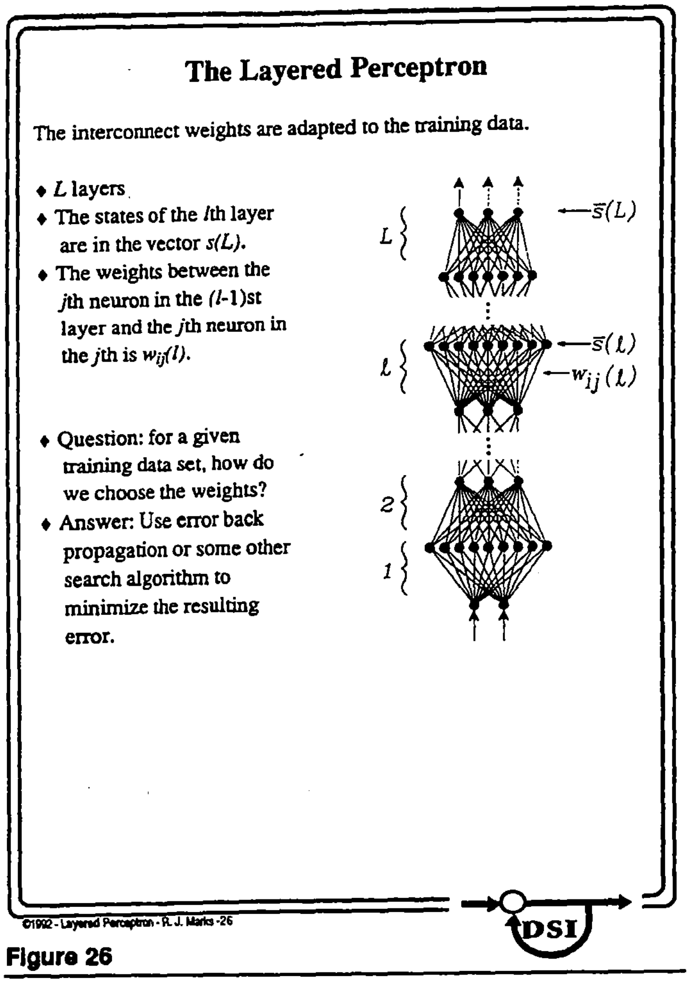


Nonlinear generalizations of the perceptron:

- ◆ Functional Link
- ◆ The Layered Perceptron
    - • Recurrent
    - • Adaptive

DSI

Figure 25



The Layered Perceptron

The interconnect weights are adapted to the training data.

- ♦ L layers
- ♦ The states of the lth layer are in the vector $s(L)$.
- ♦ The weights between the jth neuron in the $(l$-1)st layer and the jth neuron in the jth is $w_{ij}(l)$.

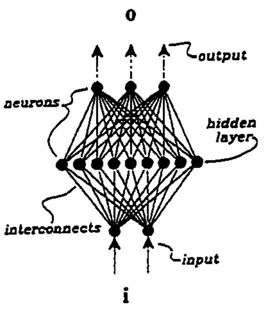
- ♦ Question: for a given training data set, how do we choose the weights?
- ♦ Answer: Use error back propagation or some other search algorithm to minimize the resulting error.

©1992 - Layered Perceptron - R. J. Marks - 26

Figure 26

The layered perceptron is another architecture wherein nonlinearities can be introduced. The result is a classifier or regression machine that can be trained by example internal to the architecture.

Characterizing The Layered Perceptron



i = input vector

o = corresponding output vector

t = desired target output

S_{Nl} = sigmoid operator

Subjects the sum of the inputs for each neuron at the *l*th level to a sigmoid nonlinearity, $s_{Nl}(\text{sum}) = [1+\exp(-\text{sum})]^{-1}$. Recall

$$ds/d(\text{sum}) = s\,(1 - s)$$

$\mathbf{W}(l)$ = the matrix of weights between the *l-1*st and *l*th levels.

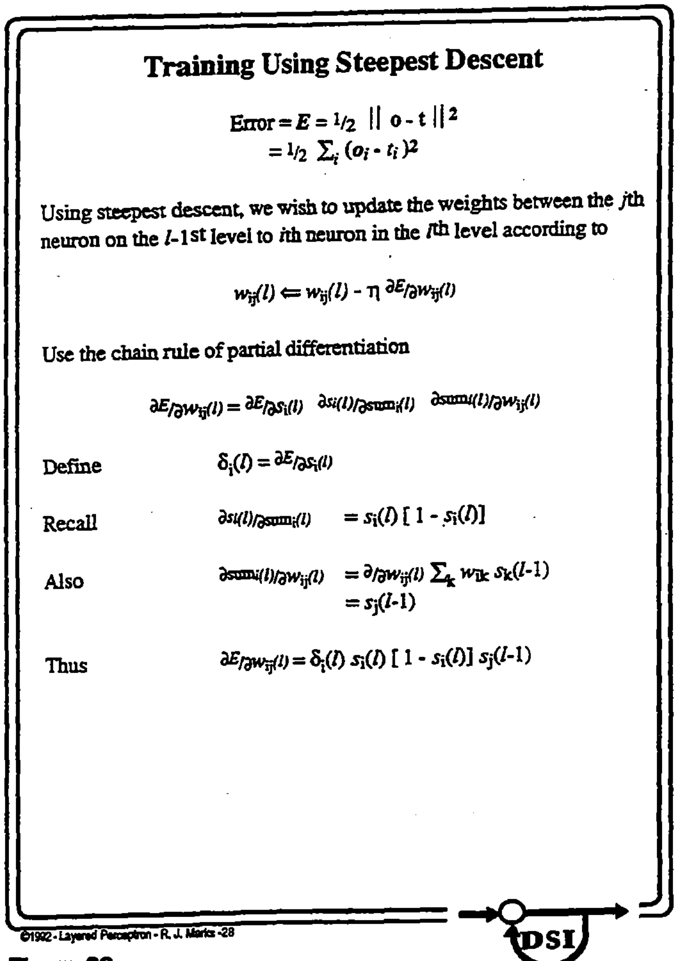
$$\mathbf{o} = S_{NL}\,\mathbf{W}(L)\,S_{N(L-1)}\,\mathbf{W}(L-1)\ldots S_{Nl}\,\mathbf{W}(l)\ldots S_{N(1)}\,\mathbf{W}(1)\,\mathbf{i}$$

©1992 - Layered Perceptron - R. J. Marks -27

Figure 27

In the forward modė, the output, **o**, of a layered perceptron is a nonlinear function of the input, *i*. The nonlinearity is in the nonlinear sigmoid operation. The choice of the sigmoid nonlinearity allows easy differentiation. This property is important in the error back propagation algorithm. The vector operator S simply performs the sigmoid operation on the sum of the inputs at a layer.

Notes Notes Notes



Training Using Steepest Descent

$$\text{Error} = E = 1/2 \; || \, o - t \, ||^2$$
$$= 1/2 \; \Sigma_i \, (o_i - t_i)^2$$

Using steepest descent, we wish to update the weights between the *j*th neuron on the *l*-1st level to *i*th neuron in the *l*th level according to

$$w_{ij}(l) \Leftarrow w_{ij}(l) - \eta \, \partial E/\partial w_{ij}(l)$$

Use the chain rule of partial differentiation

$$\partial E/\partial w_{ij}(l) = \partial E/\partial s_i(l) \;\; \partial s_i(l)/\partial \text{sum}_i(l) \;\; \partial \text{sum}_i(l)/\partial w_{ij}(l)$$

Define $\delta_i(l) = \partial E/\partial s_i(l)$

Recall $\partial s_i(l)/\partial \text{sum}_i(l) = s_i(l)\,[\,1 - s_i(l)]$

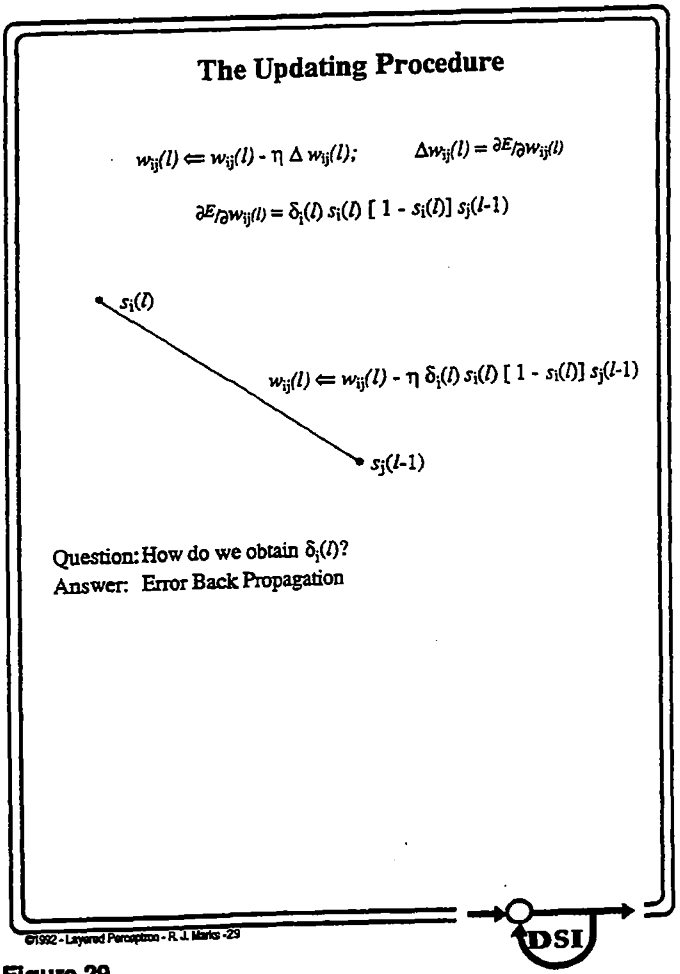
Also $\partial \text{sum}_i(l)/\partial w_{ij}(l) = \partial/\partial w_{ij}(l) \, \Sigma_k \, w_{ik} \, s_k(l\text{-}1)$
$= s_j(l\text{-}1)$

Thus $\partial E/\partial w_{ij}(l) = \delta_i(l) \, s_i(l)\,[\,1 - s_i(l)]\, s_j(l\text{-}1)$

©1992 - Layered Perceptron - R. J. Marks -28

Figure 28

The math behind the error backpropagation algorithm is quite eloquent. Error backpropagation, based on the chain rule of partial derivatives, allows training of the layered perceptron totally within the neural network architecture.



The Updating Procedure

$$w_{ij}(l) \Leftarrow w_{ij}(l) - \eta \, \Delta w_{ij}(l); \qquad \Delta w_{ij}(l) = {}^{\partial E}/_{\partial w_{ij}(l)}$$

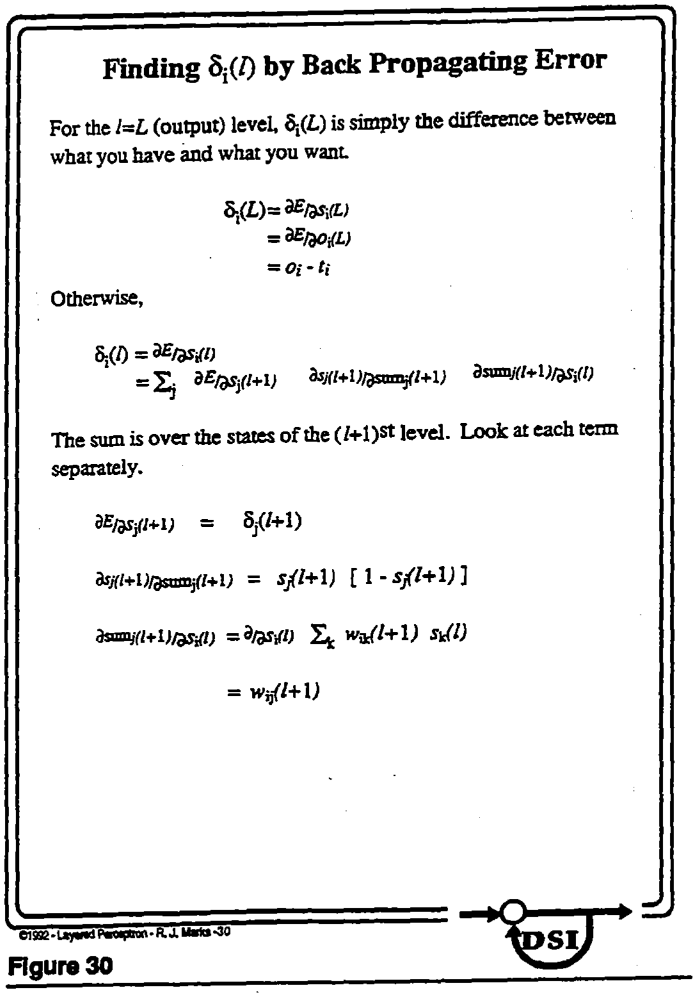
$${}^{\partial E}/_{\partial w_{ij}(l)} = \delta_i(l) \, s_i(l) \, [\, 1 - s_i(l)] \, s_j(l\text{-}1)$$

$$w_{ij}(l) \Leftarrow w_{ij}(l) - \eta \, \delta_i(l) \, s_i(l) \, [\, 1 - s_i(l)] \, s_j(l\text{-}1)$$

Question: How do we obtain $\delta_i(l)$?
Answer: Error Back Propagation

Figure 29

Two neurons are shown. The weight update between them is a function of the two neuron's states and $\delta_i(l)$.



Finding $\delta_i(l)$ by Back Propagating Error

For the l=L (output) level, $\delta_i(L)$ is simply the difference between what you have and what you want.

$$\begin{aligned}\delta_i(L) &= \partial E/\partial s_i(L)\\ &= \partial E/\partial o_i(L)\\ &= o_i - t_i\end{aligned}$$

Otherwise,

$$\begin{aligned}\delta_i(l) &= \partial E/\partial s_i(l)\\ &= \textstyle\sum_j \;\; \partial E/\partial s_j(l+1) \quad \partial s_j(l+1)/\partial \mathrm{sum}_j(l+1) \quad \partial \mathrm{sum}_j(l+1)/\partial s_i(l)\end{aligned}$$

The sum is over the states of the (l+1)st level. Look at each term separately.

$$\partial E/\partial s_j(l+1) \quad = \quad \delta_j(l+1)$$

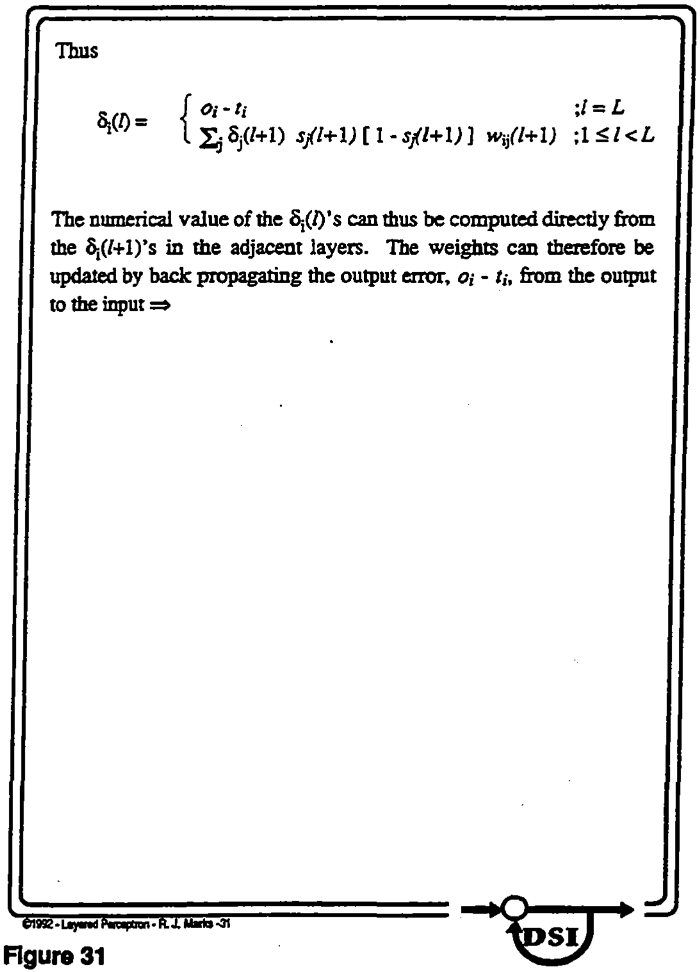
$$\partial s_j(l+1)/\partial \mathrm{sum}_j(l+1) \; = \; s_j(l+1) \;\; [\, 1 - s_j(l+1) \,]$$

$$\begin{aligned}\partial \mathrm{sum}_j(l+1)/\partial s_i(l) &= \partial/\partial s_i(l) \;\; \textstyle\sum_k \; w_{ik}(l+1) \;\; s_k(l)\\ &= w_{ij}(l+1)\end{aligned}$$

©1992 - Layered Perceptron - R. J. Marks -30

DSI

Figure 30



Thus

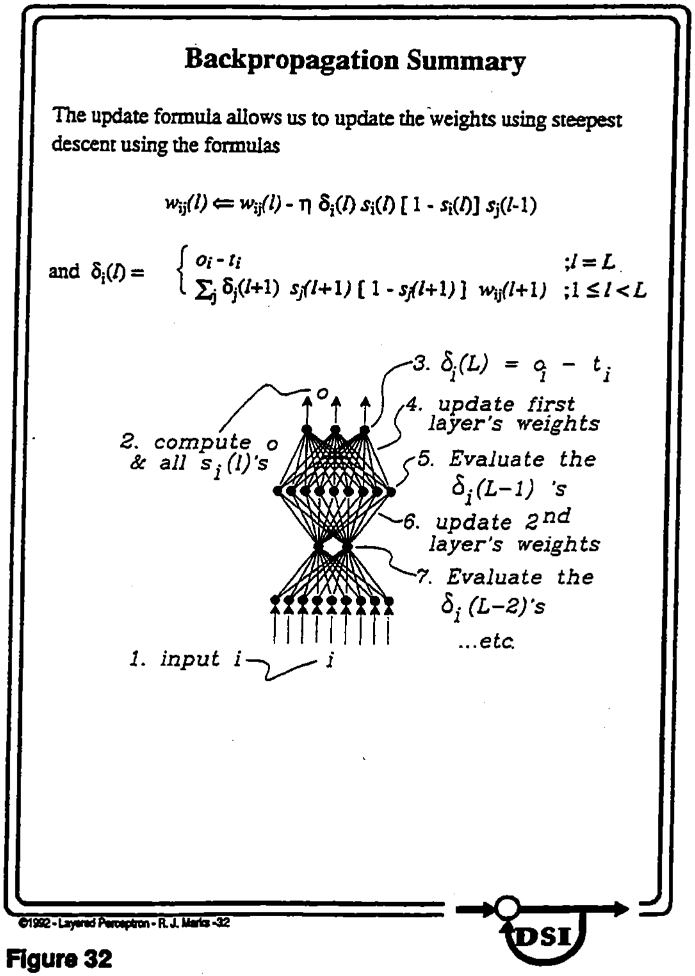
$$\delta_i(l) = \begin{cases} o_i - t_i & ; l = L \\ \sum_j \delta_j(l+1)\; s_j(l+1)\,[\,1 - s_j(l+1)\,]\; w_{ij}(l+1) & ; 1 \le l < L \end{cases}$$

The numerical value of the $\delta_i(l)$'s can thus be computed directly from the $\delta_i(l+1)$'s in the adjacent layers. The weights can therefore be updated by back propagating the output error, $o_i - t_i$, from the output to the input ⇒

©1992 - Layered Perceptron - R. J. Marks -31

Figure 31

The values of δ_i at each layer can be computed from the δ_i's at the previous layer. The δ_i at the output layer is simply the difference between what you have and what you want. Hence the name error back propagation. The training procedure is also referred to as the *generalized delta rule*.



Backpropagation Summary

The update formula allows us to update the weights using steepest descent using the formulas

$$w_{ij}(l) \Leftarrow w_{ij}(l) - \eta\ \delta_i(l)\ s_i(l)\ [\ 1 - s_i(l)]\ s_j(l\text{-}1)$$

and $\delta_i(l) = \begin{cases} o_i - t_i & ;l = L \\ \sum_j \delta_j(l+1)\ s_j(l+1)\ [\ 1 - s_j(l+1)\ ]\ w_{ij}(l+1) & ;1 \le l < L \end{cases}$

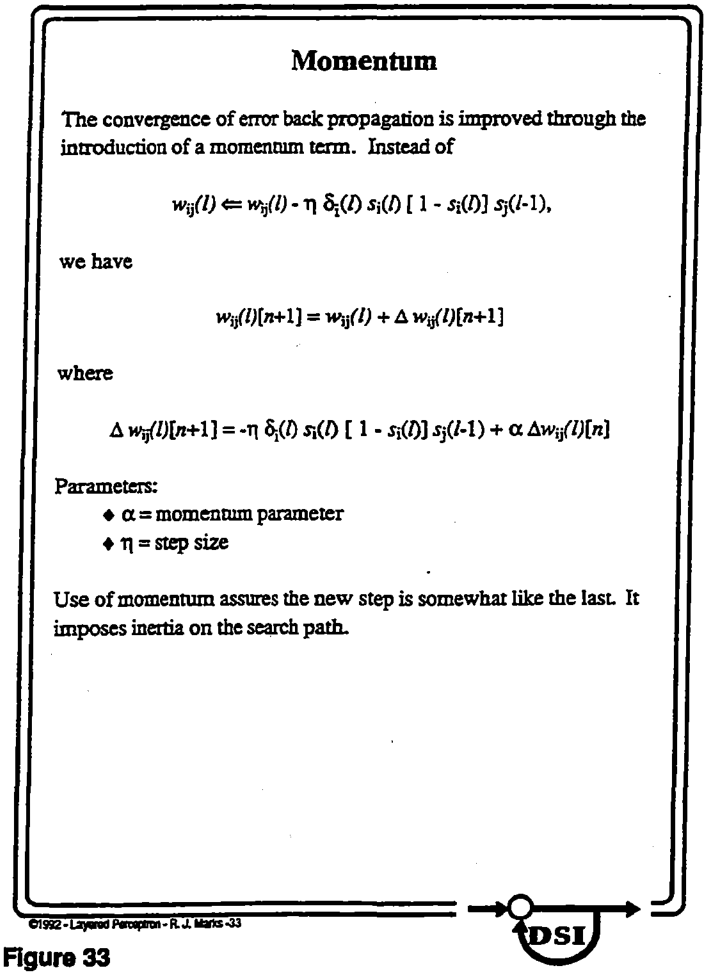
3. $\delta_i(L) = o_i - t_i$
o
4. update first layer's weights
2. compute o & all $s_i(l)$'s
5. Evaluate the $\delta_i(L-1)$'s
6. update 2nd layer's weights
7. Evaluate the $\delta_i\ (L-2)$'s
...etc.
1. input i
i

©1992 - Layered Perceptron - R. J. Marks -32

Figure 32

We consider one training data pair, (*i*,*t*). For the input *i*, compute all the neural states, including the output, *o*. The value of δ_i is computed at the output and, with the calculated states in the last two layers, the first layer of weights is updated. The value of the δ_i's are also computed at this layer. The process is repeated, and the effect of the error is used to update all the weights as it is backpropagated towards the input. The next training data pair is then used and the updating process is repeated. Iteration continues until convergence.

Notes Notes Notes



Momentum

The convergence of error back propagation is improved through the introduction of a momentum term. Instead of

$$w_{ij}(l) \Leftarrow w_{ij}(l) - \eta\, \delta_i(l)\, s_i(l)\, [\, 1 - s_i(l)]\, s_j(l\text{-}1),$$

we have

$$w_{ij}(l)[n+1] = w_{ij}(l) + \Delta\, w_{ij}(l)[n+1]$$

where

$$\Delta\, w_{ij}(l)[n+1] = -\eta\, \delta_i(l)\, s_i(l)\, [\, 1 - s_i(l)]\, s_j(l\text{-}1) + \alpha\, \Delta w_{ij}(l)[n]$$

Parameters:

- α = momentum parameter
- η = step size

Use of momentum assures the new step is somewhat like the last. It imposes inertia on the search path.

Figure 33

The performance of iterative algorithms is typically improved by the introduction of relaxation parameters. In error back propagation, this is referred to as the momentum parameter.

Notes Notes Notes

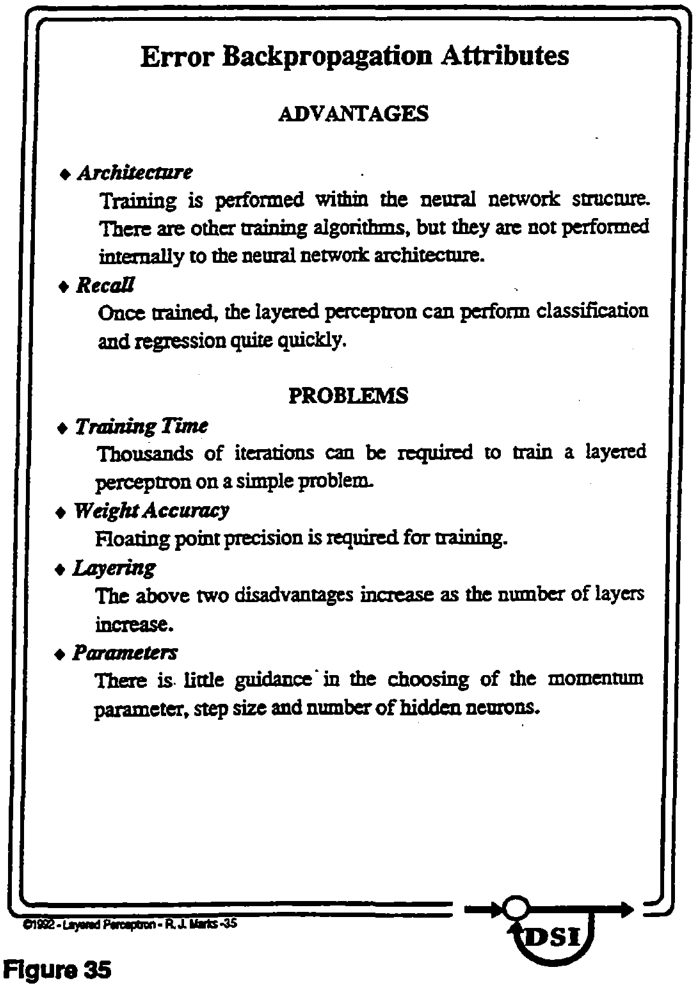
♦ Variations ♦

There are commonly used variations on the layered perceptron architecture.

1. Interconnection between nonadjacent layers.

2. Feedback interconnects between layers (recurrent neural networks).

Figure 34



Error Backpropagation Attributes

ADVANTAGES

- ◆ ***Architecture***
  Training is performed within the neural network structure. There are other training algorithms, but they are not performed internally to the neural network architecture.
- ◆ ***Recall***
  Once trained, the layered perceptron can perform classification and regression quite quickly.

PROBLEMS

- ◆ ***Training Time***
  Thousands of iterations can be required to train a layered perceptron on a simple problem.
- ◆ ***Weight Accuracy***
  Floating point precision is required for training.
- ◆ ***Layering***
  The above two disadvantages increase as the number of layers increase.
- ◆ ***Parameters***
  There is little guidance in the choosing of the momentum parameter, step size and number of hidden neurons.

©1992 - Layered Perceptron - R. J. Marks -35

DSI

Figure 35

Although back error propagation is the most widely used method to train multi-layer perceptrons, it in not the only nor necessarily the best approach. Indeed, most any algorithm that searches for a minimum can be used to train a layered perceptron. Back propagation is attractive because it can be performed within the neural network structure. The following problems are specifically associated with the back propagation algorithm. They may, however, be associated with other search paradigms also.

Training time. Thousands of iterations can be required to train a layered perceptron on even a simple problem.

Weight accuracy. Back error propagation requires high computational precision. Each iteration can result in a change in bits of only low significance. As such, training cannot be done on high speed, but low accuracy, analog electronic or optical devices. Once trained, however, a layered perceptron can be tested using low analog precision.

Layering. The required computational precision increases with the number of layers.

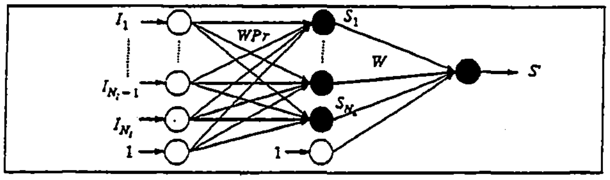
Scaling. The scaling problem can be illustrated through the *curse of dimensionality.* Specifically, for a problems of similar partition complexity, the required cardinality of the training data set grows exponentially with respect to the number of input nodes. Visualize, for example, a binary classifier with two inputs and a single output. In order to classify points within a unit square to a certain accuracy, assume that we require, say, 100 input-output data pairs.

Increase the number of inputs to three now requires classification within a unit cube. For the same precision, we now have to train on 10 planes with 100 points for each plane. The required number of data pairs increases to about 1000. Roughly, if P pairs are required in one dimension, then P^N pairs are required in N dimensions. We note, however, that correlation relationships among the input data can effect this argument. Note that this problem is not specific to the layered perceptron, but is applicable to any classifier or regression machine trained by example.

Notes Notes Notes

♦ Opimality ♦

- ♦ The layered perceptron is commonly used as a detector.
- ♦ There exists a wealth of optimal detection theory.
- ♦ How does the layered perceptron compare to optimality?



♦ Caution: Optimal detectors are typically parametric. Neural nets are not.

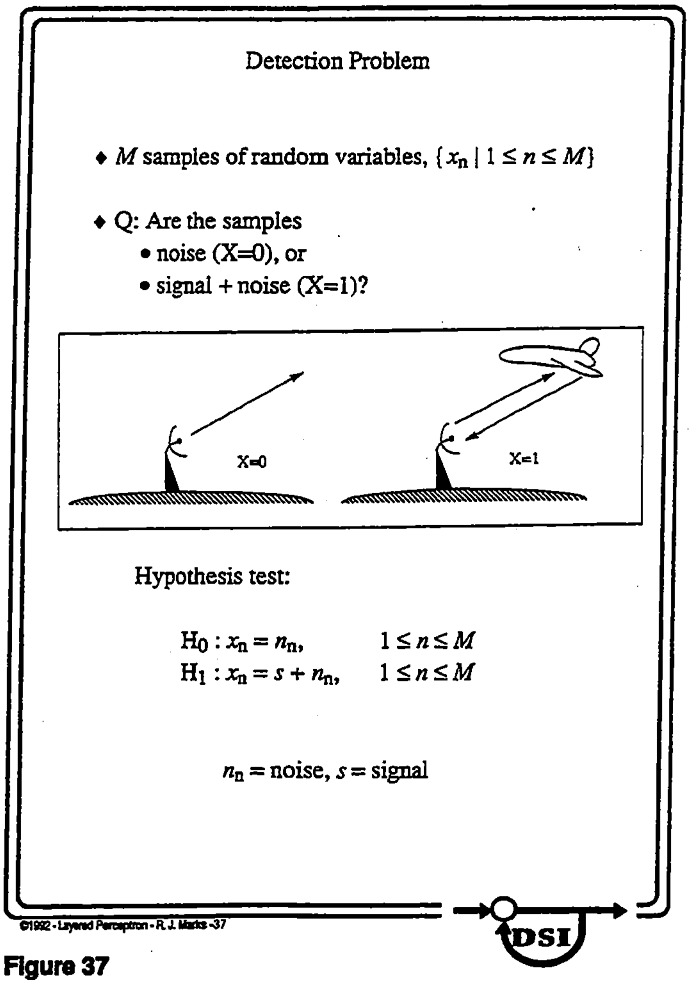
©1992 - Layered Perceptron - R. J. Marks -36

DSI

Figure 36

One important attribute of artificial neural networks is their ability to perform detection. There exists a vast literature in optimal detection theory. How does the neural network performance compare to that of the optimal detector? Quite well! This, despite the fact that the neural network does not know beforehand the parameters of the noise or signal.

Notes Notes Notes



Detection Problem

- M samples of random variables, $\{x_n \mid 1 \le n \le M\}$
- Q: Are the samples
  - noise (X=0), or
  - signal + noise (X=1)?

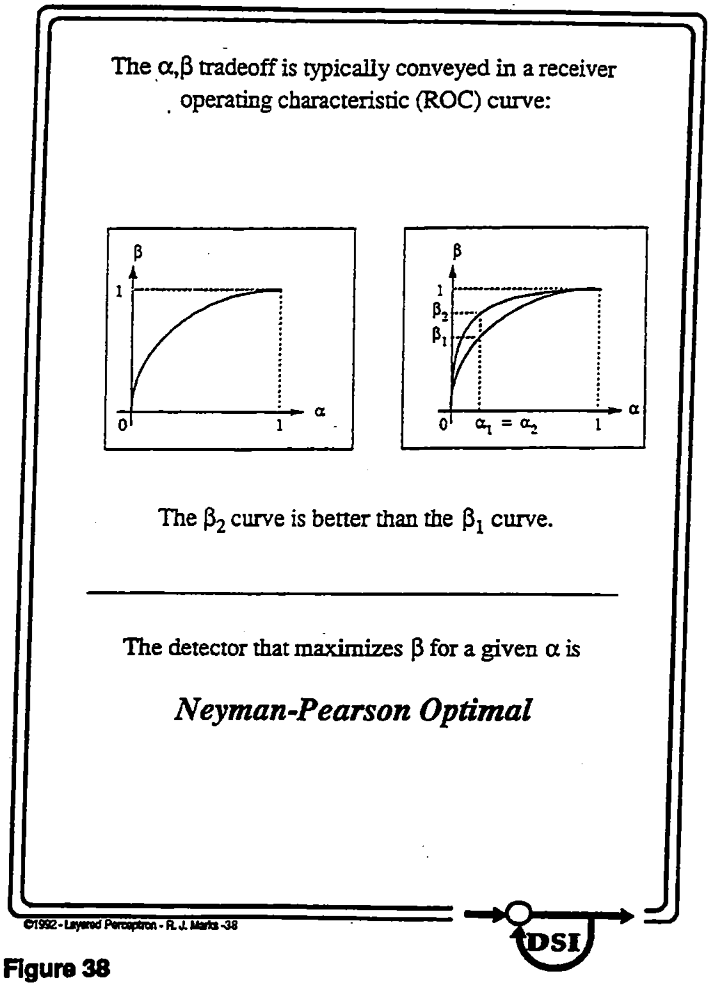
Hypothesis test:

$$\begin{array}{ll} H_0 : x_n = n_n, & 1 \le n \le M \\ H_1 : x_n = s + n_n, & 1 \le n \le M \end{array}$$

n_n = noise, s = signal

Figure 37

Elementary detection simply asks whether the received signal indicates a target is present or not. Detection theorists pose such a problem as a hypothesis test. High impedance fault detection and security assessment are example of detection problems.



The α,β tradeoff is typically conveyed in a receiver operating characteristic (ROC) curve:

The β_2 curve is better than the β_1 curve.

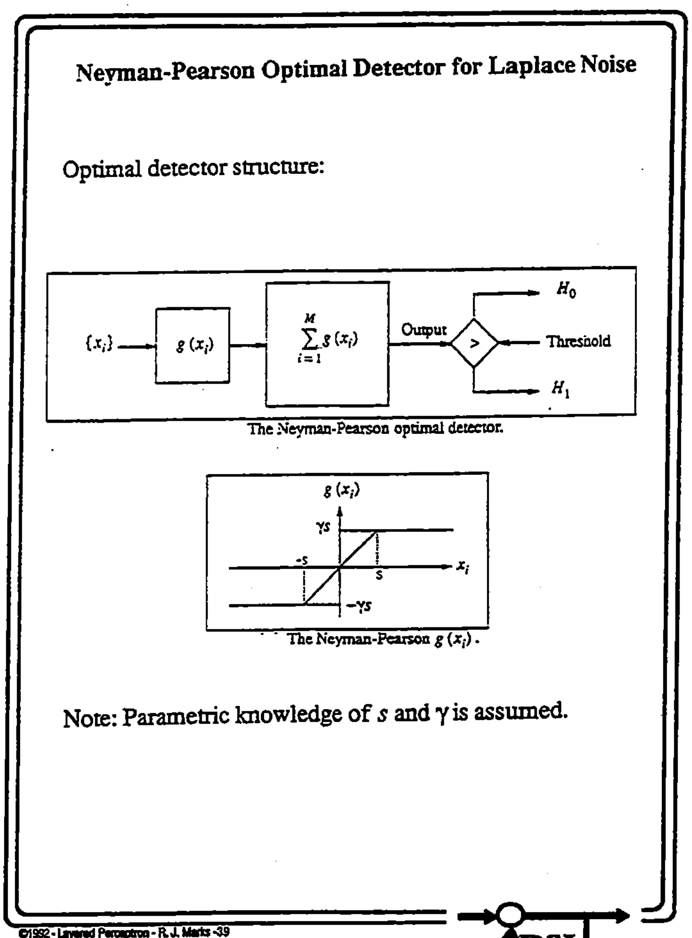
The detector that maximizes β for a given α is

Neyman-Pearson Optimal

©1992 - Layered Perceptron - R. J. Marks -38

Figure 38

There is a trade-off in detection theory between the detection probability, β, and the false alarm probability α. If your detector says the target is always present, then β = 1. The false alarm rate, though, can be poor. Conversely, if the target is always announced as absent, the false alarm probability is one, but the detection probability will be poor. The plot of β vs. α is a receiver operating curve, or ROC. For a given problem. the ROC curve which lies above all other ROC curve is referred to as Neyman-Pearson optimal. For certain detection problems, the Neyman-Pearson optimal detector is known.



Neyman-Pearson Optimal Detector for Laplace Noise

Optimal detector structure:

The Neyman-Pearson optimal detector.

The Neyman-Pearson $g(x_i)$.

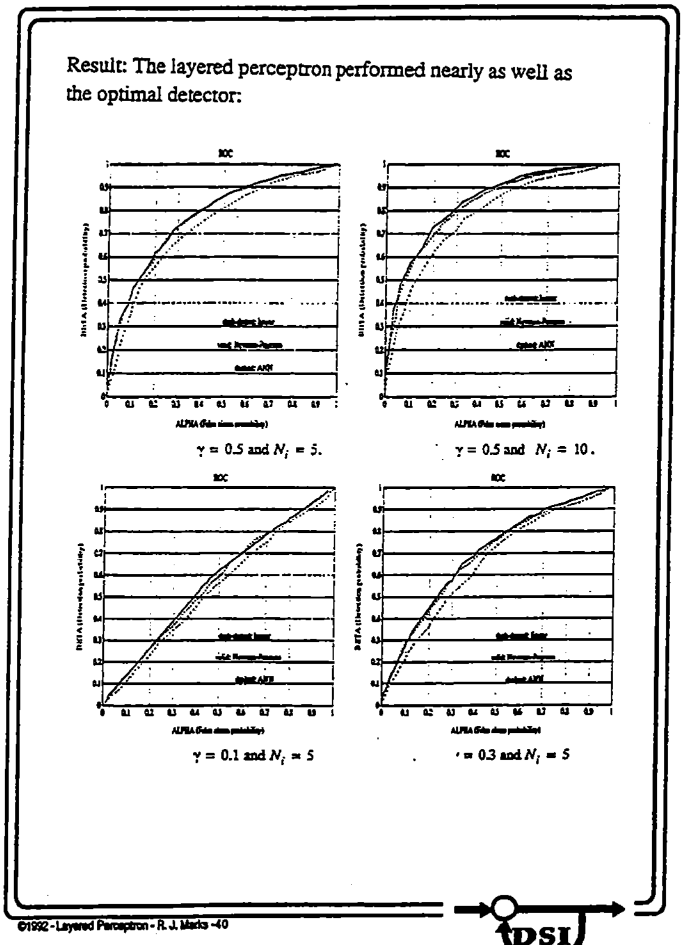
Note: Parametric knowledge of s and γ is assumed.

©1992 - Layered Perceptron - R. J. Marks -39

Figure 39

Noise with density $\gamma/2 \exp(-\gamma|n|)$ is Laplace noise with parameter γ. The Neyman-Pearson optimal detector for Laplace noise for detection of a signal s is shown. For the detection of detecting a signal in Laplace noise, there exists no better detector.

A neural network is trained for the Laplace noise problem. How does it compare to the optimal detector?



Result: The layered perceptron performed nearly as well as the optimal detector:

$\gamma = 0.5$ and $N_i = 5$.

$\gamma = 0.5$ and $N_i = 10$.

$\gamma = 0.1$ and $N_i = 5$

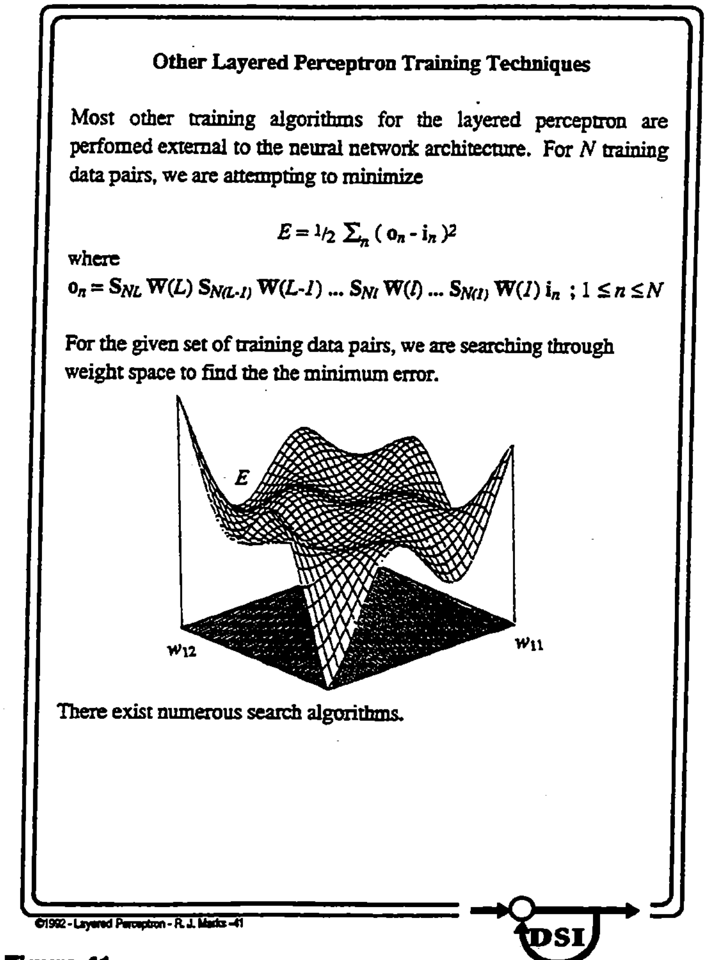
$\gamma = 0.3$ and $N_i = 5$

©1992 - Layered Perceptron - R. J. Marks -40

Figure 40

Shown are ROC curves for various Laplace parameters for various sample sizes. The ROC curves for the neural network detector are nearly as good as the optimal detector in each case.

This result demonstrates that the neural network has a remarkable ability to perform in the absence of parameterized data. Note that the optimal detector requires knowledge of the Laplace parameter and the signal strength. The neural network does not.



Other Layered Perceptron Training Techniques

Most other training algorithms for the layered perceptron are perfomed external to the neural network architecture. For N training data pairs, we are attempting to minimize

$$E = \tfrac{1}{2} \sum_n (\mathbf{o}_n - \mathbf{i}_n)^2$$

where

$$\mathbf{o}_n = \mathbf{S}_{NL}\, \mathbf{W}(L)\, \mathbf{S}_{N(L-1)}\, \mathbf{W}(L-1) \ldots \mathbf{S}_{Nl}\, \mathbf{W}(l) \ldots \mathbf{S}_{N(1)}\, \mathbf{W}(1)\, \mathbf{i}_n \ ; \ 1 \le n \le N$$

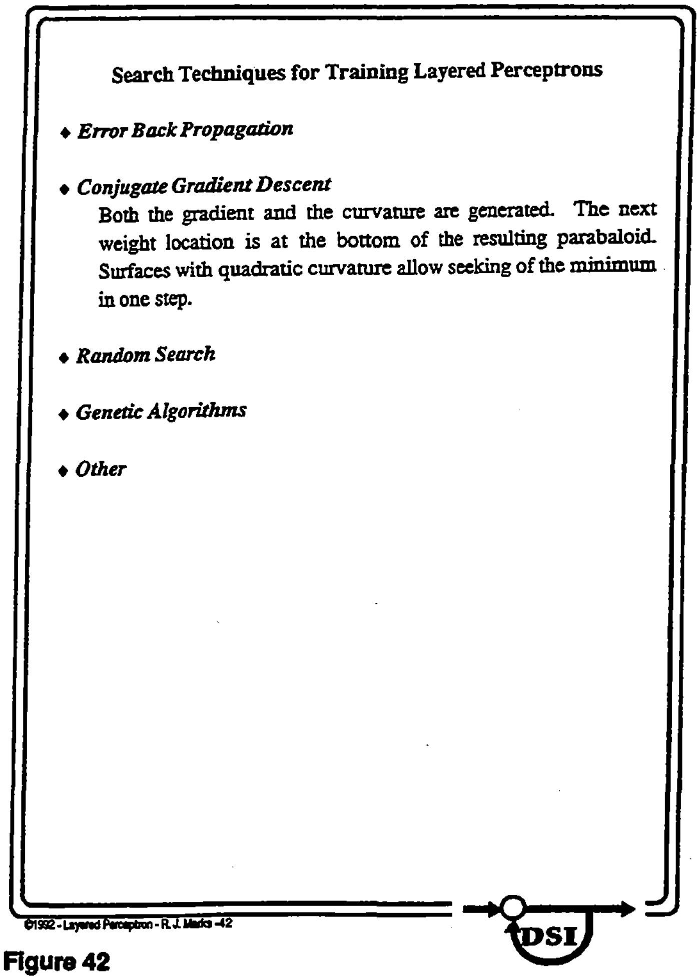
For the given set of training data pairs, we are searching through weight space to find the the minimum error.

There exist numerous search algorithms.

©1992 - Layered Perceptron - R. J. Marks -41

Figure 41

The training data and the neural network architecture dictate an error surface in weight space. Training corresponds to finding that point in weight space where the error is minimized. There exist numerous techniques to search for such a minimum.

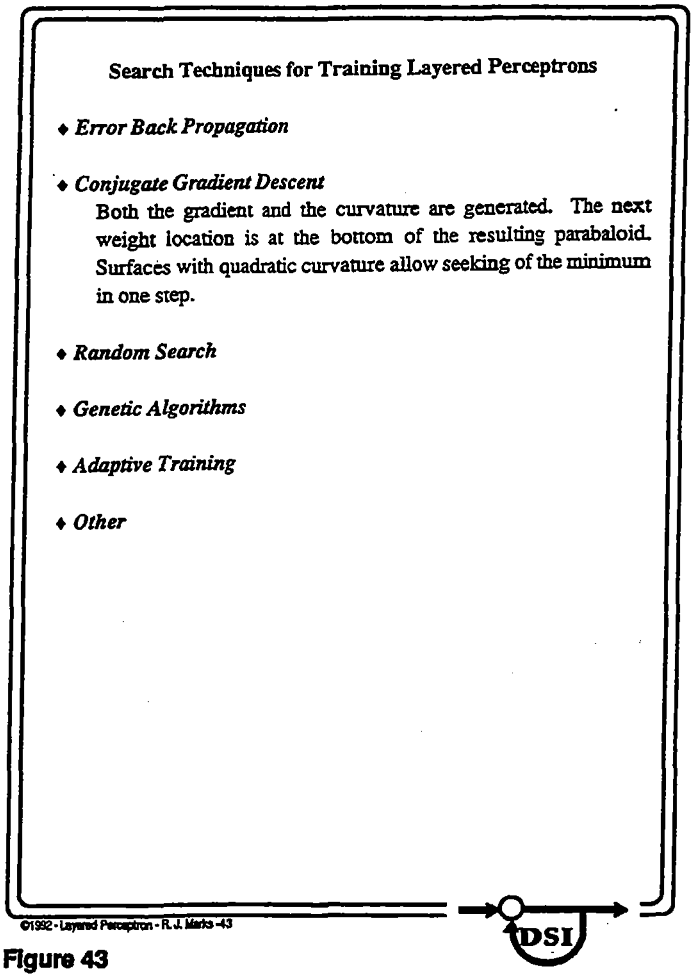


Search Techniques for Training Layered Perceptrons

- ***Error Back Propagation***
- ***Conjugate Gradient Descent***
  Both the gradient and the curvature are generated. The next weight location is at the bottom of the resulting parabaloid. Surfaces with quadratic curvature allow seeking of the minimum in one step.
- ***Random Search***
- ***Genetic Algorithms***
- ***Other***

©1992 - Layered Perceptron - R.J. Marks -42

Figure 42



Search Techniques for Training Layered Perceptrons

- ♦ *Error Back Propagation*
- ♦ *Conjugate Gradient Descent*
  Both the gradient and the curvature are generated. The next weight location is at the bottom of the resulting paraboloid. Surfaces with quadratic curvature allow seeking of the minimum in one step.
- ♦ *Random Search*
- ♦ *Genetic Algorithms*
- ♦ *Adaptive Training*
- ♦ *Other*

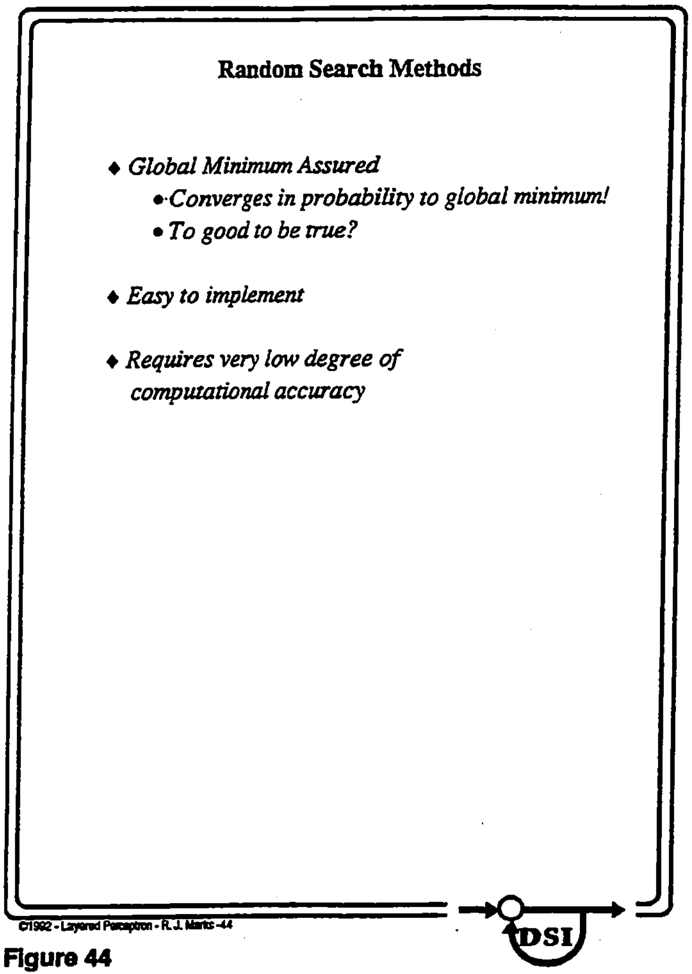
©1992 - Layered Perceptron - R.J. Marks -43

Figure 43

There are a plethora of techniques available for optimization. Error back propagation is the most commonly used. It has the advantage of being able to implemented totally within the neural network structure.

Given a neural network architecture and a training data set, however, training a neural network simply corresponds to finding weights that will give a minimum to the error function. The techniques listed here have each been investigated as a technique to train a layered perceptron.

We will elaborate briefly on random search, genetic algorithms and adaptive training.



Random Search Methods

- *Global Minimum Assured*
  - *Converges in probability to global minimum!*
  - *To good to be true?*
- *Easy to implement*
- *Requires very low degree of computational accuracy*

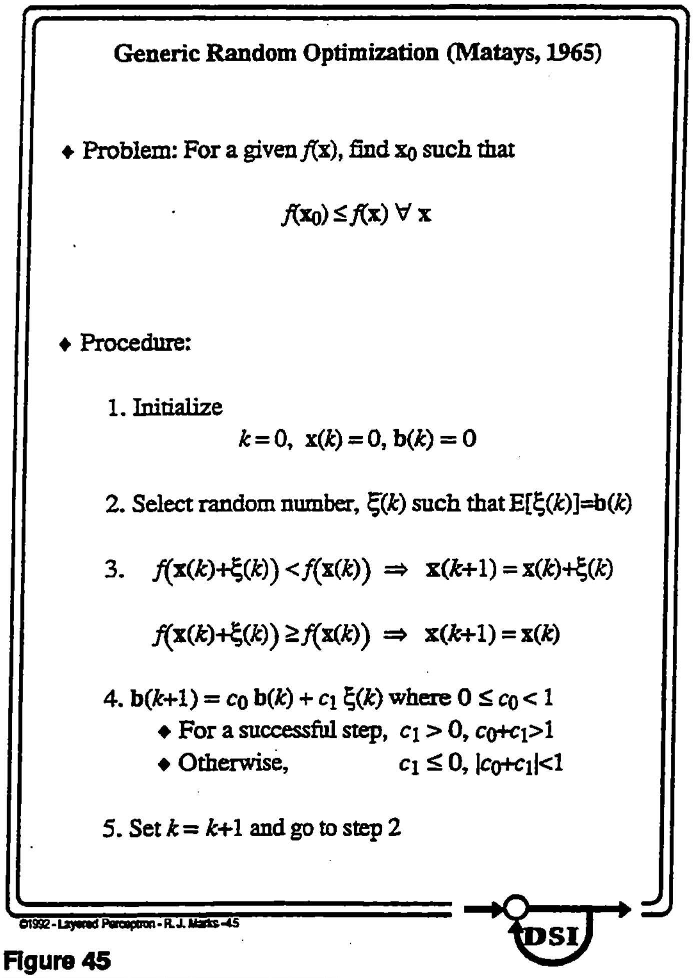
©1992 - Layered Perceptron - R. J. Marks -44

Figure 44

Good news: Random search has a fantastic property! It is guaranteed to converge to the global minimum! Error back propagation can get stuck in local minima.

Bad news: Convergence is guaranteed *in probability*, a very weak type of convergence. For example, the limit of the probability that you will win the Washington state lottery as your number of attempts goes to infinity is equal to one.

The other good news is that, in comparison with error back propagation, random search can be implemented with low accuracy computation.



Generic Random Optimization (Matays, 1965)

- Problem: For a given $f(\mathbf{x})$, find $\mathbf{x}_0$ such that

$$f(\mathbf{x}_0) \leq f(\mathbf{x}) \ \forall \ \mathbf{x}$$

- Procedure:

1. Initialize
$$k = 0, \ \mathbf{x}(k) = 0, \ \mathbf{b}(k) = 0$$

2. Select random number, $\xi(k)$ such that $E[\xi(k)]{=}\mathbf{b}(k)$

3. $f(\mathbf{x}(k){+}\xi(k)) < f(\mathbf{x}(k)) \ \Rightarrow \ \mathbf{x}(k{+}1) = \mathbf{x}(k){+}\xi(k)$

$f(\mathbf{x}(k){+}\xi(k)) \geq f(\mathbf{x}(k)) \ \Rightarrow \ \mathbf{x}(k{+}1) = \mathbf{x}(k)$

4. $\mathbf{b}(k{+}1) = c_0\, \mathbf{b}(k) + c_1\, \xi(k)$ where $0 \leq c_0 < 1$
   - For a successful step, $c_1 > 0$, $c_0{+}c_1{>}1$
   - Otherwise, $c_1 \leq 0$, $|c_0{+}c_1|{<}1$

5. Set $k = k{+}1$ and go to step 2

©1992 - Layered Perceptron - R. J. Marks -45

DSI

Figure 45

Here is the mathematics describing basic random optimization.

For steepest descent (*e.g.* error back propagation), the idea is this. We are standing on the error surface with the goal of locating the minimum. We sense the direction of going downhill and take a step. The procedure is repeated until the global minimum is, hopefully, found.

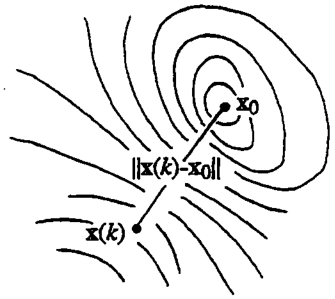
In random search, a step is taken randomly. If the new location is better than the old location, you stay there. If not, you go back to the original spot and take another step. There is a sense of momentum used, in that the statistics of your step are a function of the result of your previous successful step. Again, this approach guarantees convergence to the global minimum with probability one.

Notes Notes Notes

Convergence

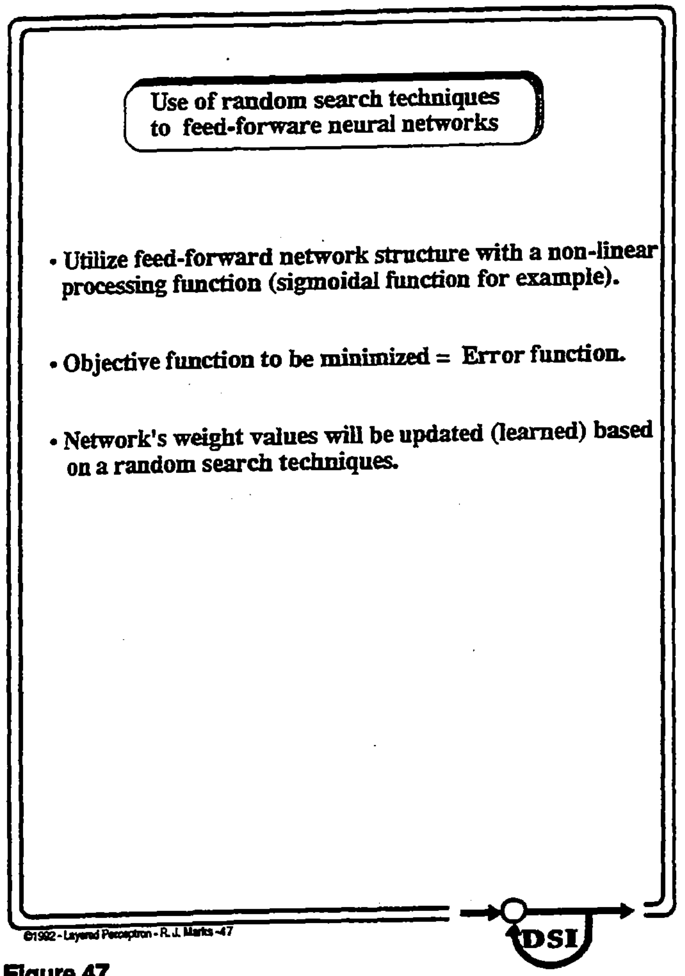
$$\forall\, \delta > 0,$$

$$\lim_{k \to 0} \text{Prob}[\|\mathbf{x}(k)\text{-}\mathbf{x}_0\| > \delta] = 0$$



DSI

Figure 46



Use of random search techniques to feed-forware neural networks

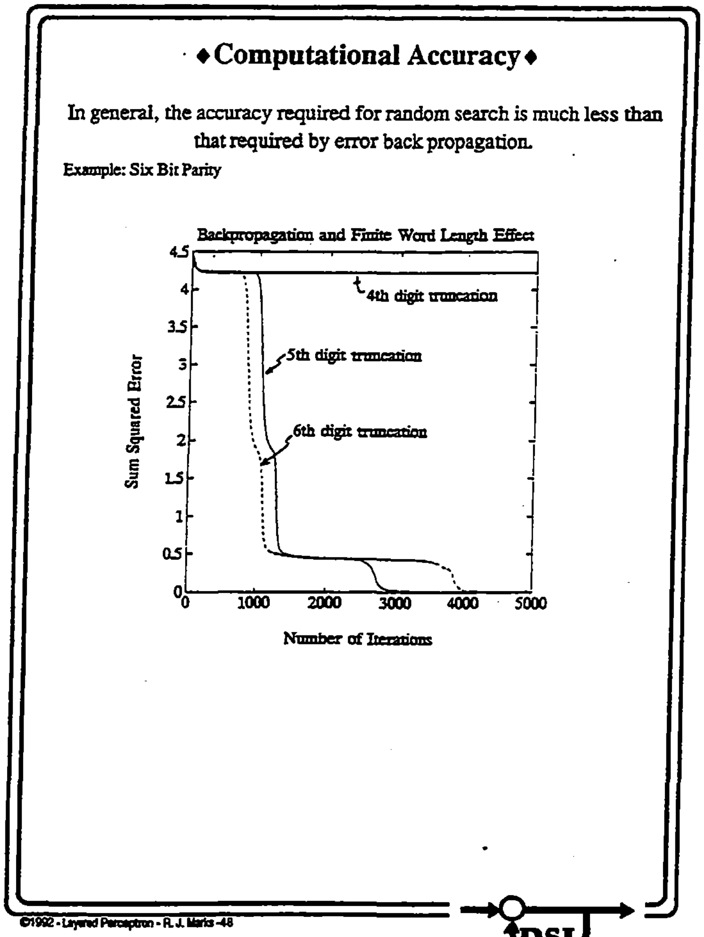
- Utilize feed-forward network structure with a non-linear processing function (sigmoidal function for example).

- Objective function to be minimized = Error function.

- Network's weight values will be updated (learned) based on a random search techniques.

©1992 - Layered Perceptron - R. J. Marks -47

Figure 47



♦Computational Accuracy♦

In general, the accuracy required for random search is much less than that required by error back propagation.

Example: Six Bit Parity

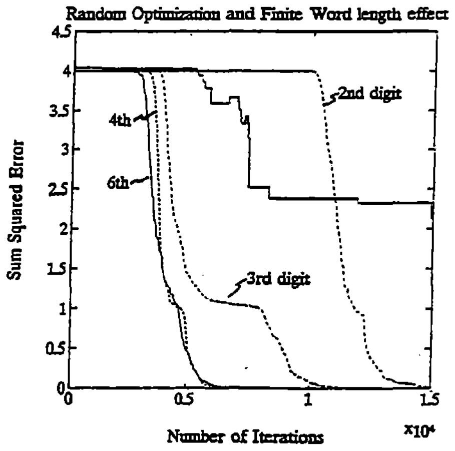
©1992 - Layered Perceptron - R. J. Marks -48

Figure 48

Error back propagation requires high computational accuracy. In most cases, for example, training can not be performed with analog accuracy. This is not the case for random search.

The six bit parity problem looks at whether the number of bits equal to one is even or not. If even, a one is indicated. Otherwise, a zero results. Shown here is the error resulting in training a layered perceptron when the weight accuracy is limited to a small number of digits. For four digits, the network does not converge.

Using random search training, convergence can occur at a relatively low computational accuracy.

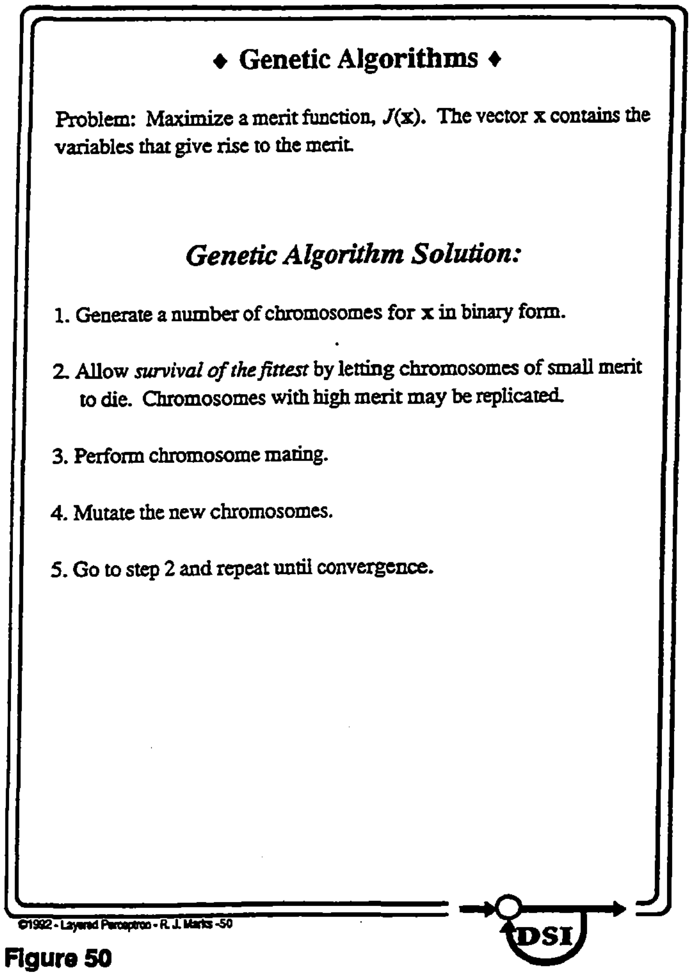


©1992 - Layered Perceptron - R. J. Marks -49

Figure 49

Training a layered perceptron to recognize a six bit parity can be achieved using a much lower computational accuracy. Convergence, illustrated here, occurs with two digit accuracy.

(The random optimization used here is a more sophisticated version than that described previously.)



◆ **Genetic Algorithms** ◆

Problem: Maximize a merit function, $J(\mathbf{x})$. The vector $\mathbf{x}$ contains the variables that give rise to the merit.

Genetic Algorithm Solution:

1. Generate a number of chromosomes for $\mathbf{x}$ in binary form.
2. Allow *survival of the fittest* by letting chromosomes of small merit to die. Chromosomes with high merit may be replicated.
3. Perform chromosome mating.
4. Mutate the new chromosomes.
5. Go to step 2 and repeat until convergence.

©1992 - Layered Perceptron - R. J. Marks -50

Figure 50

Notes Notes Notes

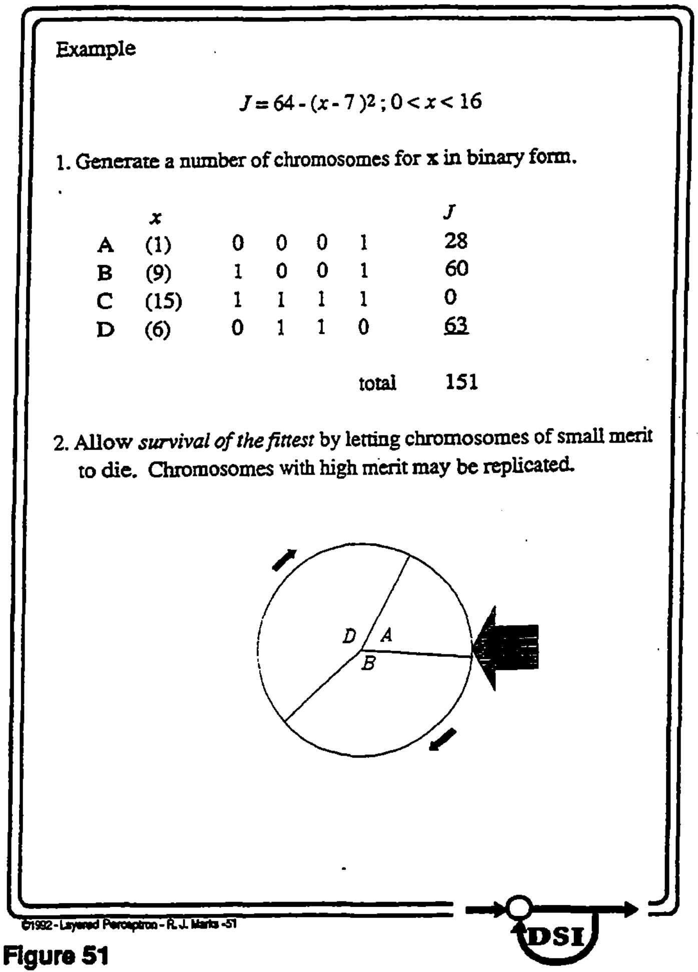
The field of Artificial Life is giving rise to numerous biologically motived computational paradigms, including genetic algorithms and evolutionary programing.

Genetic algorithms are an optimization technique used to minimize a merit function, *J(x)*. For neural networks, this could be 1/*E* where *E* is the error function. Listed here is the iterative procedure for optimization.

Genetic algorithms can be used to simultaneously determine the optimal architecture for a layered perceptron (*i.e.* the number of hidden layers and number of neurons in each layer) as well as the numerical values of the weights.

Genetic algorithms have been applied to optimal capacitor placement in power systems.

Notes Notes Notes



Example

$$J = 64 - (x - 7)^2 ; 0 < x < 16$$

1. Generate a number of chromosomes for x in binary form.

	x					J
A	(1)	0	0	0	1	28
B	(9)	1	0	0	1	60
C	(15)	1	1	1	1	0
D	(6)	0	1	1	0	63
					total	151

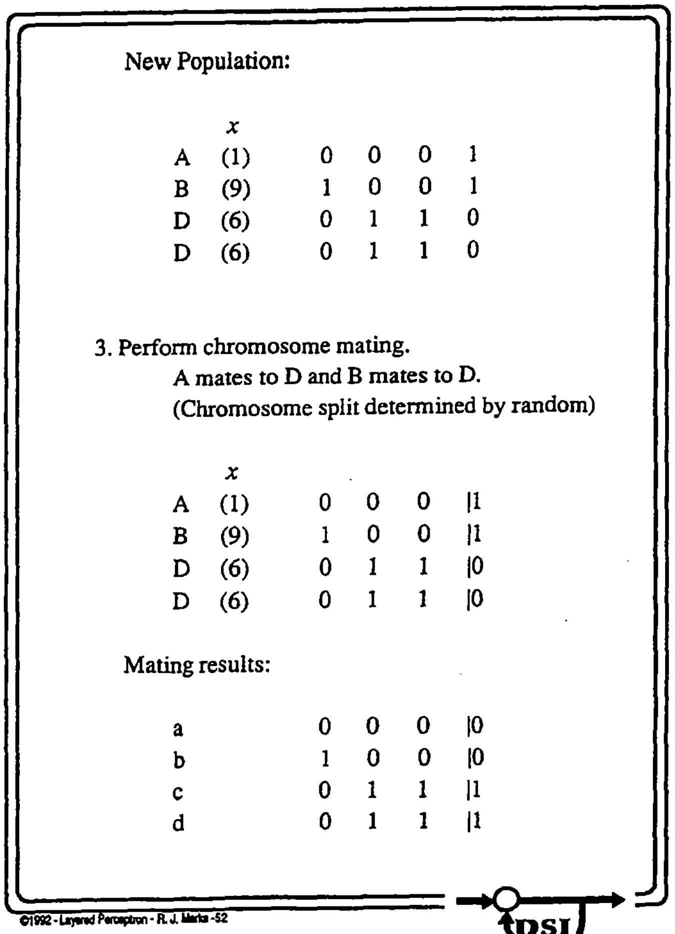
2. Allow *survival of the fittest* by letting chromosomes of small merit to die. Chromosomes with high merit may be replicated.

©1992 - Layered Perceptron - R. J. Marks -51

Figure 51

Chromosomes are strings of ones and zeros. When they are used as neural networks, the chromosomes are binary strings corresponding to the weights. The weights are placed end to end to form a very long chromosome. Here, the initial chromosomes are selected at random.

In the first step, the merit of each chromosome is evaluated. Secondly, we allow *survival of the fittest*. This process can be visualized using the wheel-of-fortune shown on the bottom. The chromosome *A* has a 28% chance of being chosen. Similarly, *B* has a 60% chance, *C* has a zero chance and *D* has a 63% chance. The wheel is spun equal to the number of chromosomes. In this case, it is spun four times.



New Population:

	x				
A	(1)	0	0	0	1
B	(9)	1	0	0	1
D	(6)	0	1	1	0
D	(6)	0	1	1	0

3. Perform chromosome mating.
A mates to D and B mates to D.
(Chromosome split determined by random)

	x				
A	(1)	0	0	0	|1
B	(9)	1	0	0	|1
D	(6)	0	1	1	|0
D	(6)	0	1	1	|0

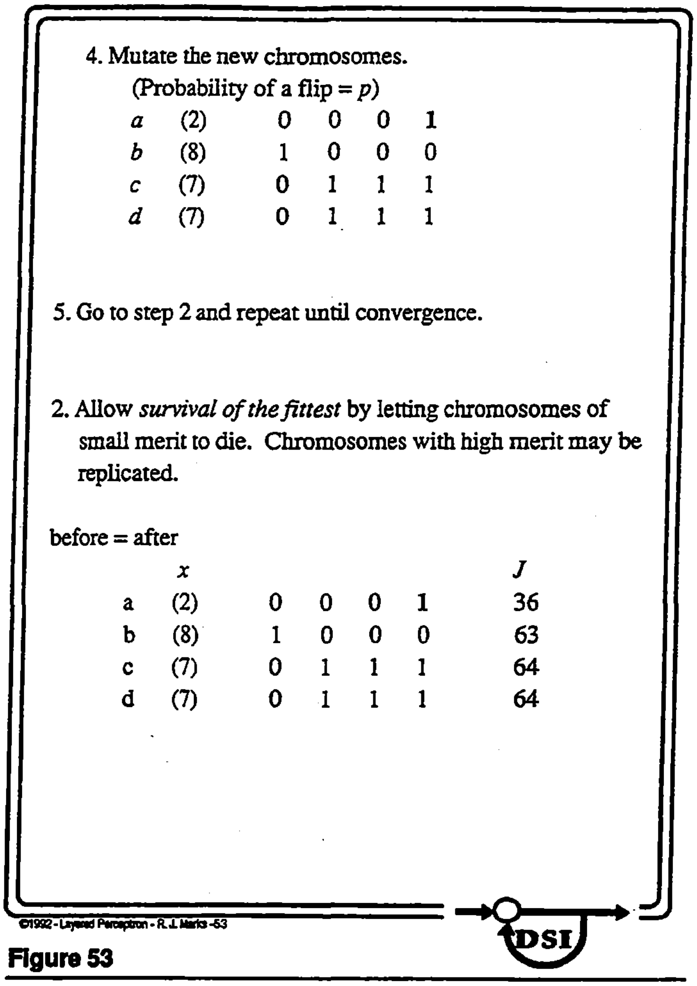
Mating results:

a	0	0	0	|0
b	1	0	0	|0
c	0	1	1	|1
d	0	1	1	|1

©1992 - Layered Perceptron - R. J. Marks -52

Figure 52

A new population results from the spin of the wheel of fortune. Next comes mating. We will mate *A* with *D* and *B* with *D*. To mate, a random point is chosen within the chromosome. The sections are then swapped to form the next generation. We even give them new names. In this case, they are (*a*, *b*, *c* and *d*).



4. Mutate the new chromosomes.

(Probability of a flip = p)

a	(2)	0	0	0	1
b	(8)	1	0	0	0
c	(7)	0	1	1	1
d	(7)	0	1	1	1

5. Go to step 2 and repeat until convergence.

2. Allow *survival of the fittest* by letting chromosomes of small merit to die. Chromosomes with high merit may be replicated.

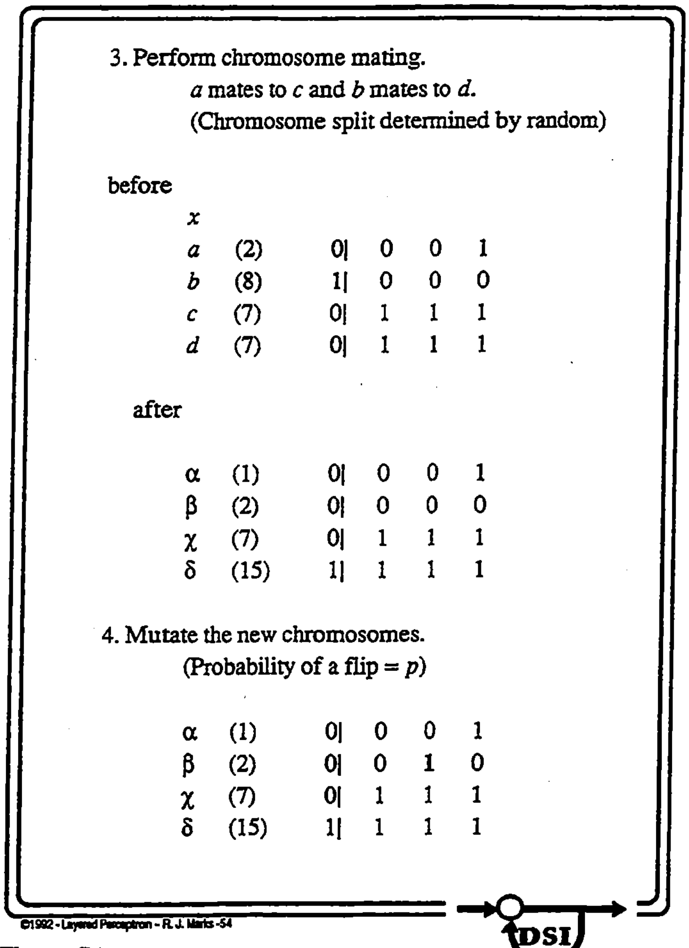
before = after

	x					J
a	(2)	0	0	0	1	36
b	(8)	1	0	0	0	63
c	(7)	0	1	1	1	64
d	(7)	0	1	1	1	64

Figure 53

Mutation: With a certain small probability, each bit has a chance of being changed (a one to a zero or a zero to a one). In this example, the zero in the upper right hand corner is changed to a one.

The process is repeated, beginning with step two (survival of the fittest). At the bottom of the page is shown the merit figure calculated after mutation.



3. Perform chromosome mating.
a mates to *c* and *b* mates to *d*.
(Chromosome split determined by random)

before

x					
a	(2)	0|	0	0	1
b	(8)	1|	0	0	0
c	(7)	0|	1	1	1
d	(7)	0|	1	1	1

after

α	(1)	0|	0	0	1
β	(2)	0|	0	0	0
χ	(7)	0|	1	1	1
δ	(15)	1|	1	1	1

4. Mutate the new chromosomes.
(Probability of a flip = p)

α	(1)	0|	0	0	1
β	(2)	0|	0	1	0
χ	(7)	0|	1	1	1
δ	(15)	1|	1	1	1

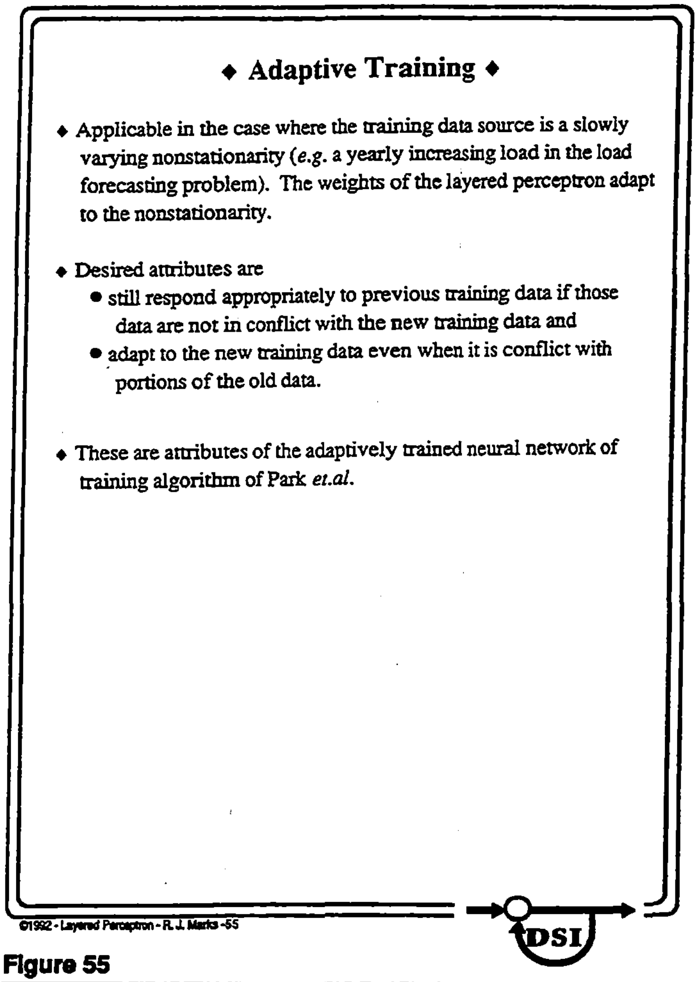
Figure 54

Shown are the results of the next iteration. The solution is clearly approaching the solution of $x = 7$. Recall that the merit function is:

$$J(x) = 64 - (x-7)^2$$

Thus, $x = 7$ is the desired result. In the last set of chromosomes, $x = 7$ is dominating the solution.

Evolutionary programming and genetic algorithms are highly promising paradigms in the field of computational intelligence in general and neural networks in particular.



◆ Adaptive Training ◆

- ◆ Applicable in the case where the training data source is a slowly varying nonstationarity (*e.g.* a yearly increasing load in the load forecasting problem). The weights of the layered perceptron adapt to the nonstationarity.
- ◆ Desired attributes are
  - ● still respond appropriately to previous training data if those data are not in conflict with the new training data and
  - ● adapt to the new training data even when it is conflict with portions of the old data.
- ◆ These are attributes of the adaptively trained neural network of training algorithm of Park *et.al.*

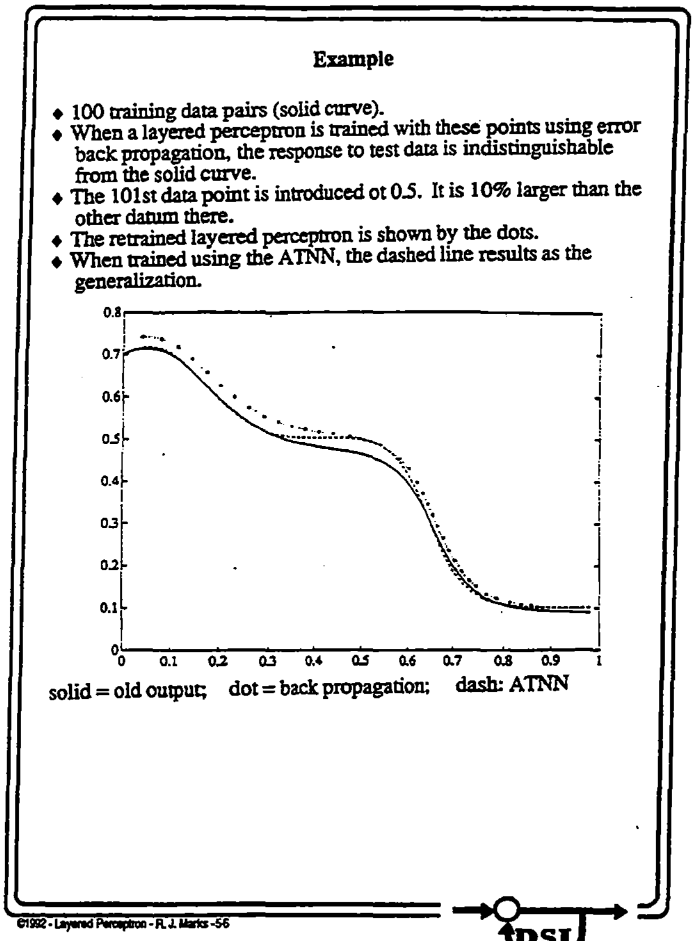
©1992 - Layered Perceptron - R. J. Marks -55

DSI

Figure 55

In the training of a layered perceptron, an assumption of stationarity of the training data is typically made. In a number of cases of interest, however, the training data is a slowly varying nonstationary process. Consider, as an example, training data for the load forecasting problem generated in a developing urban area. Training data from five years prior will be different in character to data more recently generated. In order for the layered perceptron's weights to adapt to a slowly varying nonstationarity, such a procedure should comply with the two attributes listed.

The adaptively trained neural network (ATNN) assures proper response to previous training data by seeking to minimize a weight sensitivity cost function while, at the same time, minimizing the mean square error normally ascribed to the layered perceptron.



Example

- 100 training data pairs (solid curve).
- When a layered perceptron is trained with these points using error back propagation, the response to test data is indistinguishable from the solid curve.
- The 101st data point is introduced ot 0.5. It is 10% larger than the other datum there.
- The retrained layered perceptron is shown by the dots.
- When trained using the ATNN, the dashed line results as the generalization.

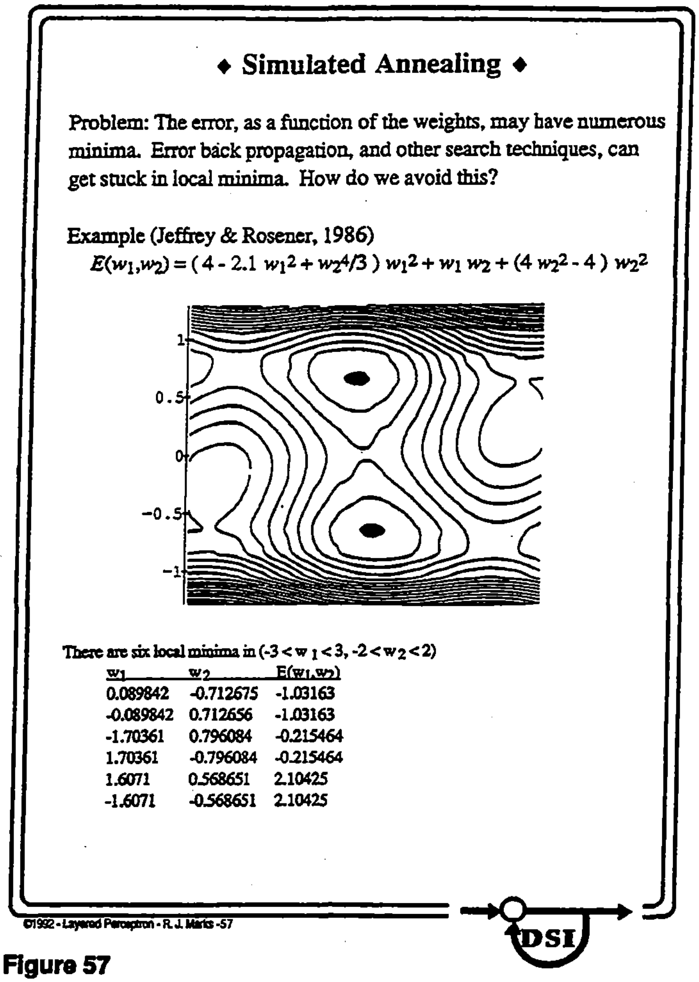
solid = old output; dot = back propagation; dash: ATNN

©1992 - Layered Perceptron - R. J. Marks -56

Figure 56

We illustrate the performance of the ATNN through an exemplar problem. Later, the procedure will be applied to the load forecasting problem.

A total of 100 training data pairs were generated using the solid curve. When a layered perceptron is trained with these points using error back propagation, the response to test data is indistinguishable from the solid curve. The 101st data point is introduced to 0.5. It is 10% larger than the other datum there. When the layered perceptron is retrained using error back propagation, the generalization is shown by the dots. When trained using the ATNN, the dashed line results as the generalization. Clearly, the dashed line has adapted to the new data point without a resulting drift of the other data. Such was not the case for error back propagation.



◆ Simulated Annealing ◆

Problem: The error, as a function of the weights, may have numerous minima. Error back propagation, and other search techniques, can get stuck in local minima. How do we avoid this?

Example (Jeffrey & Rosener, 1986)

$$E(w_1,w_2) = (4 - 2.1\, w_1^2 + w_2^4/3)\, w_1^2 + w_1 w_2 + (4\, w_2^2 - 4)\, w_2^2$$

There are six local minima in $(-3 < w_1 < 3, -2 < w_2 < 2)$

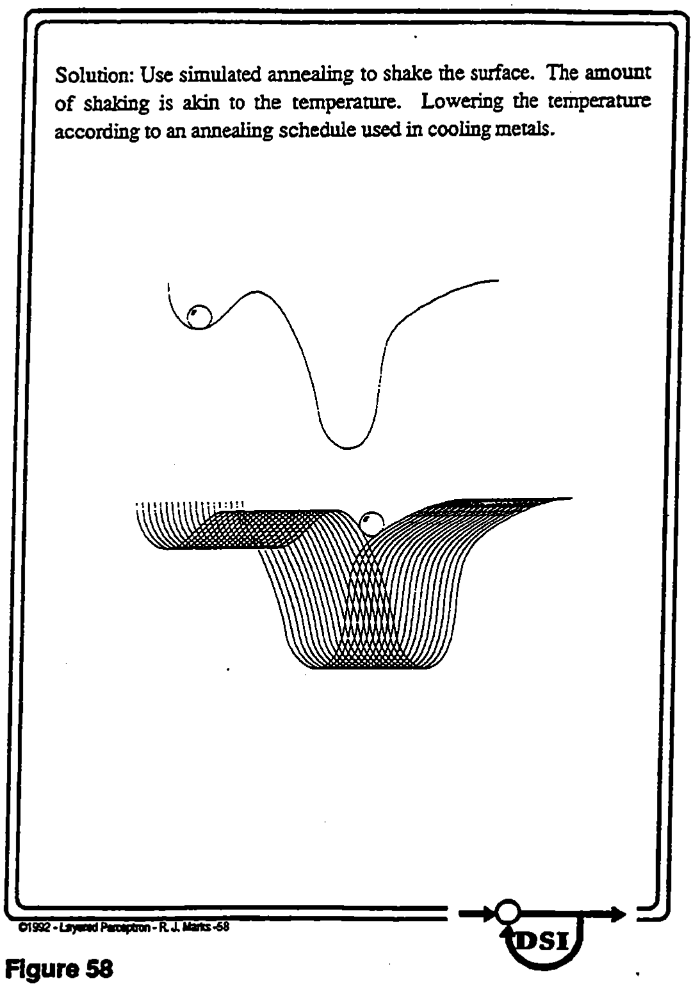
w_1	w_2	$E(w_1,w_2)$
0.089842	-0.712675	-1.03163
-0.089842	0.712656	-1.03163
-1.70361	0.796084	-0.215464
1.70361	-0.796084	-0.215464
1.6071	0.568651	2.10425
-1.6071	-0.568651	2.10425

©1992 - Layered Perceptron - R. J. Marks -57

Figure 57

For a given training data set and layered perceptron architecture, there typically exists numerous local minima in weight space. Many optimization techniques, including steepest descent, can get stuck in these local minima. Simulated annealing can be used to avoid shallow local minima and, ultimately, to find the global minimum. The contour plot shown is that of a simple deterministic function with multiple minima.

Annealing is a process used in cooling metals to assure maximum strength.

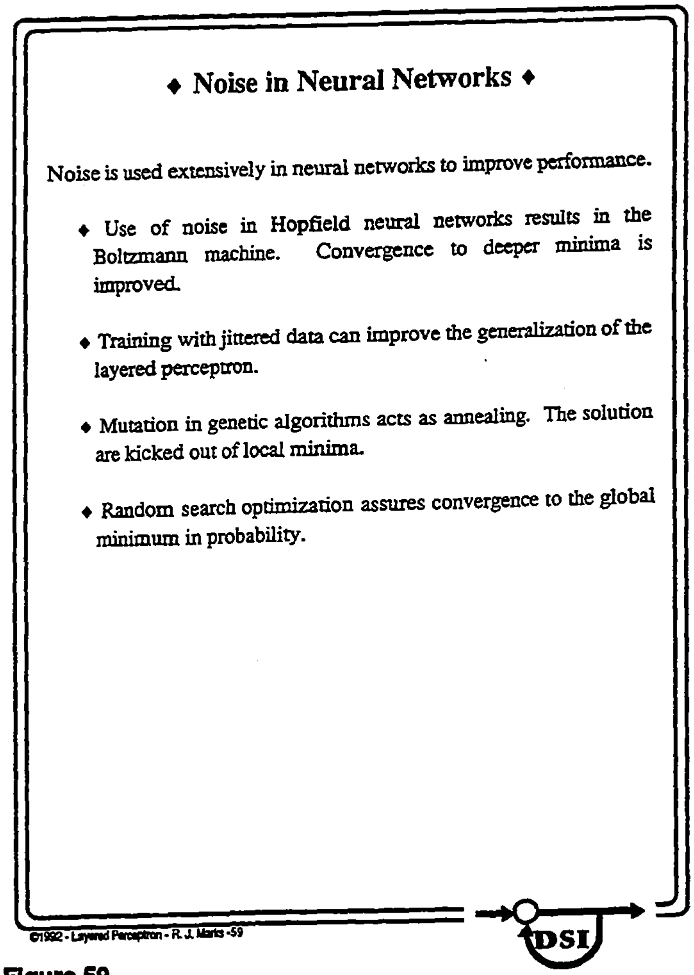


Solution: Use simulated annealing to shake the surface. The amount of shaking is akin to the temperature. Lowering the temperature according to an annealing schedule used in cooling metals.

©1992 - Layered Perceptron - R. J. Marks -58

Figure 58

Annealing is used is cooling metal to assure maximum strength. Simulated annealing is similar. By adding noise to the weights during the training process, and decreasing that noise during the training process, we are, in effect, lowering temperature. If the temperature is lowered at a sufficiently slow rate, the global minimum is reached.



◆ Noise in Neural Networks ◆

Noise is used extensively in neural networks to improve performance.

- ◆ Use of noise in Hopfield neural networks results in the Boltzmann machine. Convergence to deeper minima is improved.
- ◆ Training with jittered data can improve the generalization of the layered perceptron.
- ◆ Mutation in genetic algorithms acts as annealing. The solution are kicked out of local minima.
- ◆ Random search optimization assures convergence to the global minimum in probability.

©1992 - Layered Perceptron - R. J. Marks -59

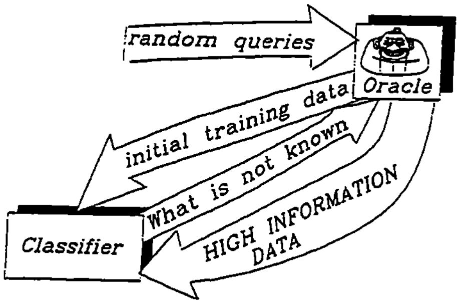
DSI

Figure 59

Interestingly, noise plays an important role in many approaches to improving the performance of neural networks. Paradoxically, accuracy is improved through the use of randomness.

Accelerated Learning Using Queries

The Basic Concept:



©1992 - Layered Perceptron - R.J. Marks -60

Figure 60

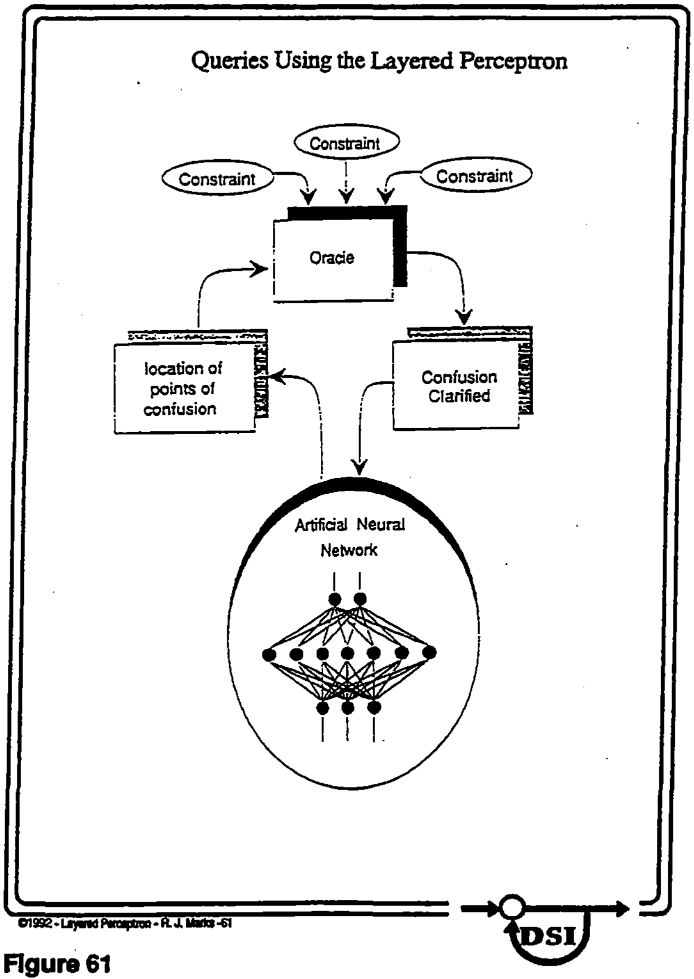
When a classifier or regression machine is trained by random example, *the more that is learned, the harder it is to learn* (*i.e.* you can't teach an old dog new tricks). This is true of the multilayered perceptron. Indeed, in the absence of data noise, additional learning takes place in a multi layered perceptron only if new data is introduced that the neural network improperly classifies. The closer the representation comes to the concept, the smaller the chance that this happens.

To illustrate, consider the classification problem of learning the location of a point a on the interval $0 < a < 1$. We choose a point at random on the unit interval. If it to the right of a, we assign it a value of one. If is to the left of a, the result is 0. It is clear that, after a number of data points have been generated at random on the unit interval, that a lies somewhere between the rightmost 0 and the left most 1. Call this subinterval C. If we generate a new data point that does not lie in the subinterval C, we have learned nothing new. If the new point lies in the subinterval C, then we revise the subinterval and make it's duration shorter. Doing so, however, decreases the chance that the next data point contains new information. That is, the probability decreases that the new data point lies in the shorter interval. Thus, in this example, the more we learn about the location of the point a, the harder it is to learn. One approach to counteract this phenomenon is with the use of oracles in query based learning.

Oracles: In supervised learning, each feature vector is assigned a classification (or regression) value or values. There is usually a cost associated with this assignment, such as the cost of performing an experiment, computational overhead or simply time. We can envision this process as a presentation to an *oracle* the feature vector. For a cost, the oracle will reveal to us the proper classification or regression value associated with that vector. Note that, if we have deep pockets to pay the oracle, there is no need to for a classifier or regression machine such as the layered perceptron. Any feature vector we desire can be taken to the oracle for proper categorization.

In many cases of interest, we have the freedom to choose the feature vectors that we present to the oracle. Ideally, we would like to present those vectors to the oracle that, in some sense, will result in training data of high information content. The motive is to effectively train the classifier or regression machine with a low training data cost. Query based training is concerned with the manner in which the training vectors that will result in high information data are chosen.

Queries Using the Layered Perceptron



Constraint

Constraint

Constraint

Oracle

location of points of confusion

Confusion Clarified

Artificial Neural Network

Figure 61

Oracles can be a computer simulation or an experiment. Independent of its form, the oracle must be paid to answer a query. A neural network can be used in a query based system if the net can be inverted.

Observation: Points close to the boundary have the highest degree of confusion. Use *conjugate pairs*.

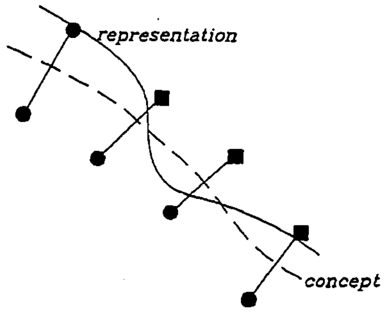
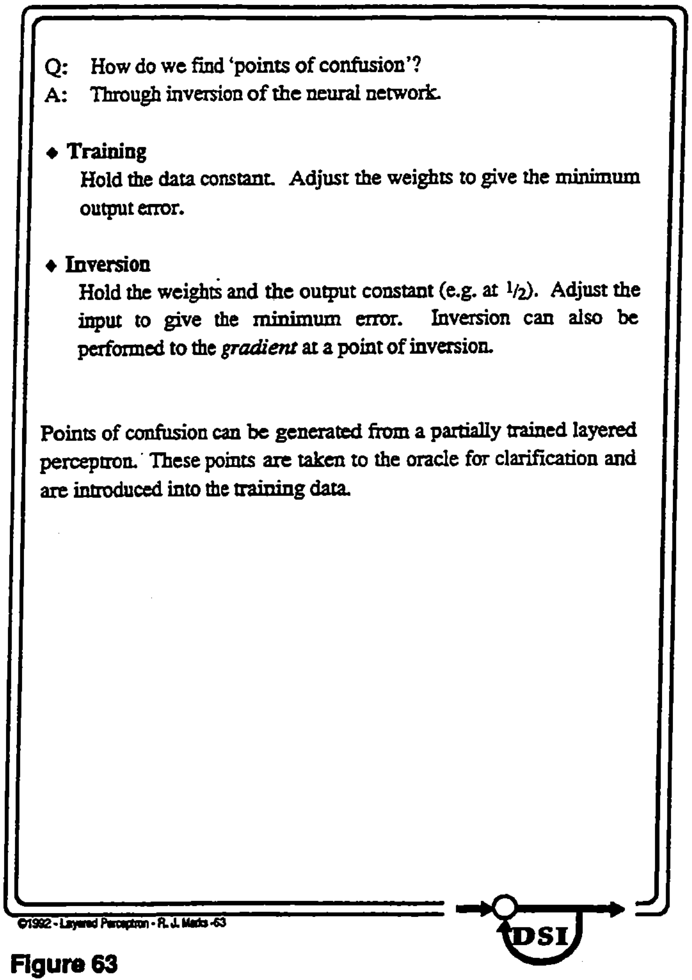


Figure 62

The binary classification problem is totally determined by the classification boundary. Indeed, here is an obvious case where the importance of data to the classification can be noted. Roughly, the closer a feature vector is to the concept classification boundary, the more information it contains.

One way to exploit this observation is through interval halving. Between each feature vector classified 0 and each classified 1, there exists a classification boundary. In many cases, taking the geometric midpoint of these two feature vectors to the oracle will result in a classification point closer to the boundary. This is assured, for example, if the underlying concept is convex.

To illustrate interval halving, let's return to the problem of finding the point a on the interval (0,1). After N randomly generated points on this interval, we would expect (in the sense of statistics), that the distance between the right most zero and the left most one is about $1/N$. Using interval halving, on the other hand, this is reduced to about 2^N. The acceleration in learning is indeed remarkable.



Q: How do we find 'points of confusion'?
A: Through inversion of the neural network.

- **Training**
  Hold the data constant. Adjust the weights to give the minimum output error.

- **Inversion**
  Hold the weights and the output constant (e.g. at $1/2$). Adjust the input to give the minimum error. Inversion can also be performed to the *gradient* at a point of inversion.

Points of confusion can be generated from a partially trained layered perceptron. These points are taken to the oracle for clarification and are introduced into the training data.

©1992 - Layered Perceptron - R. J. Marks -63

Figure 63

Another approach to query based learning is, in effect, to ask a partially trained classifier or regression machine "What is it you don't understand?". The response of the classifier or regression machine is taken to the oracle for proper categorization and the result is added to the training data set. The classifier is then further trained and the process repeated.

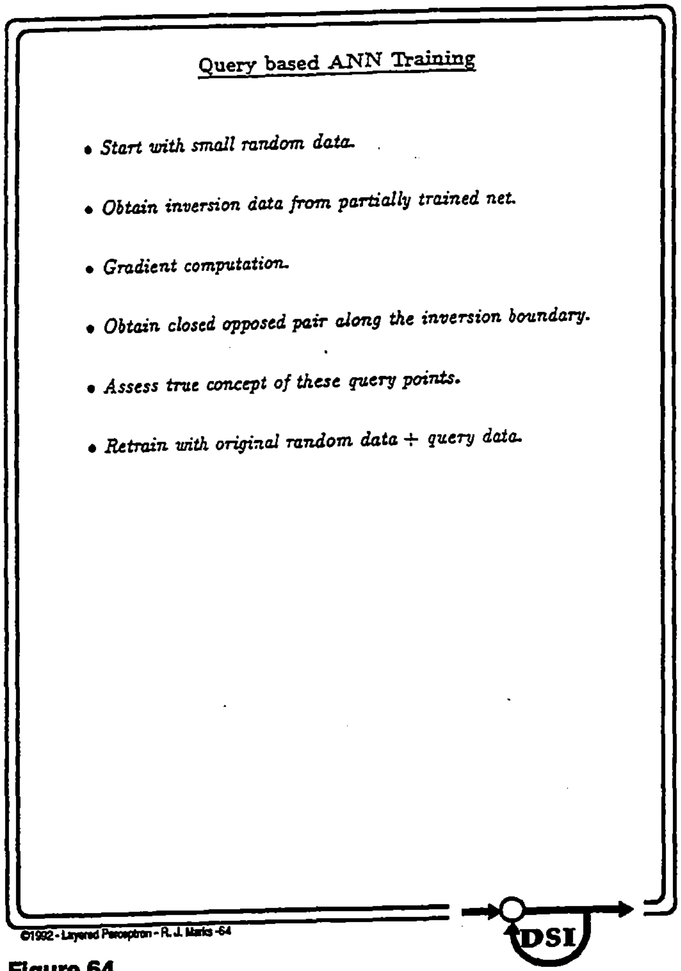
How might we apply this query approach to, say, a trained layered perceptron classifier with a single output? Assuming that the output neuron is thresholded at one half to make the classification decision, the representation boundary in feature vector space is the locus of all inputs that produce an output of one half. This locus of points corresponds to feature vectors of maximum confusion. In other words, when presented with such a vector, the neural network is uncertain to the corresponding classification. If there were a technique to find a number of these points, they could be taken to the oracle to clear the confusion. The data from the oracle could then be used for training data. The perceptron can

Notes Notes Notes

then be retrained to yield a higher accuracy. The question is, how can the locus of confusion be generated? The answer is through inversion of the neural network.

One technique for inversion of the layered perceptron has been proposed by Hwang *et.al.* The approach is basically the dual of back propagation. Instead of holding the training data constant and adjusting the weights by using steepest descent, the weights are held constant and the input is adjusted using steepest descent to give an output of one half. Clearly, a number of inputs will give the response of one half. Variations are imposed by changing the initial starting point of the input in the iteration procedure. Use of inversion in query based learning has resulted in a significant improvement in accuracy of a trained layered perceptron in comparison with a second neural network trained with a randomly selected data set of the same cardinality. In practice, data near (rather than on) the representation boundary was used to accelerate training.

Notes Notes Notes



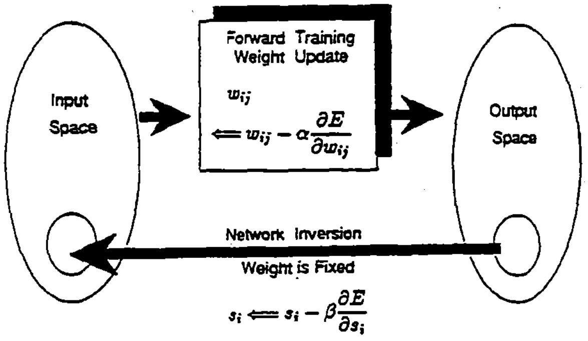
Query based ANN Training

- *Start with small random data.*
- *Obtain inversion data from partially trained net.*
- *Gradient computation.*
- *Obtain closed opposed pair along the inversion boundary.*
- *Assess true concept of these query points.*
- *Retrain with original random data + query data.*

Figure 64

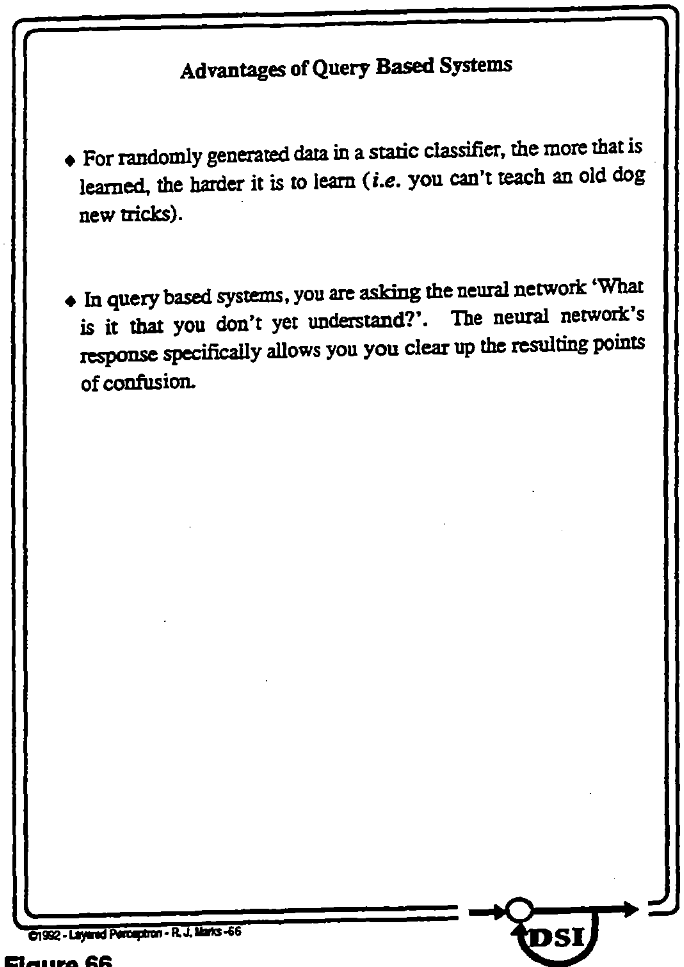
Notes Notes Notes

Query Flow


Input Space
Forward Training
Weight Update
$w_{ij} \Longleftarrow w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}}$
Output Space
Network Inversion
Weight is Fixed
$s_i \Longleftarrow s_i - \beta \frac{\partial E}{\partial s_i}$

©1992 - Layered Perceptron - R. J. Marks -65

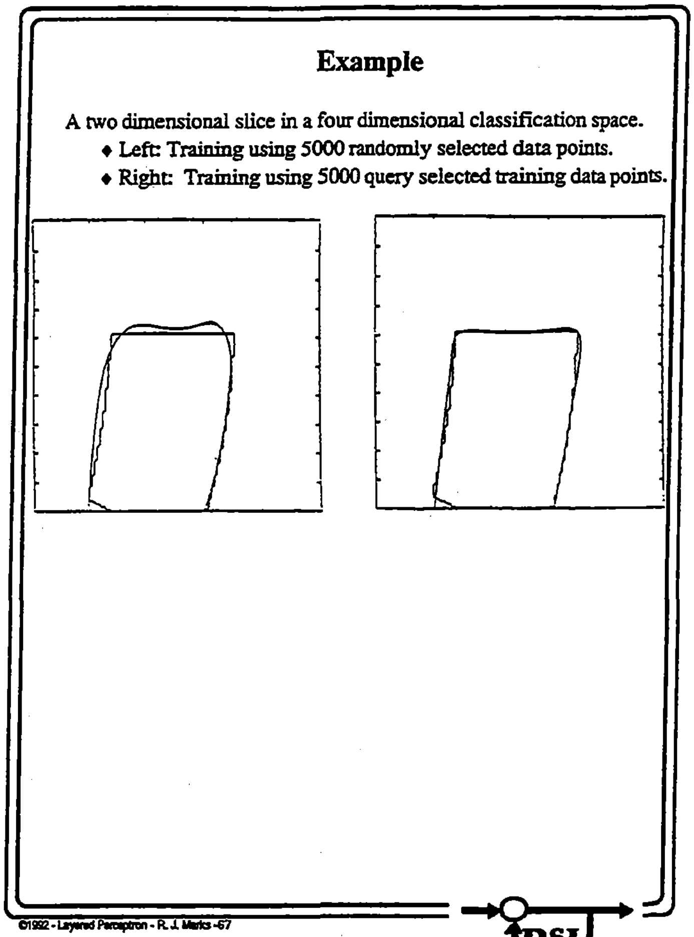
Figure 65



Advantages of Query Based Systems

- For randomly generated data in a static classifier, the more that is learned, the harder it is to learn (*i.e.* you can't teach an old dog new tricks).
- In query based systems, you are asking the neural network 'What is it that you don't yet understand?'. The neural network's response specifically allows you you clear up the resulting points of confusion.

Figure 66



Example

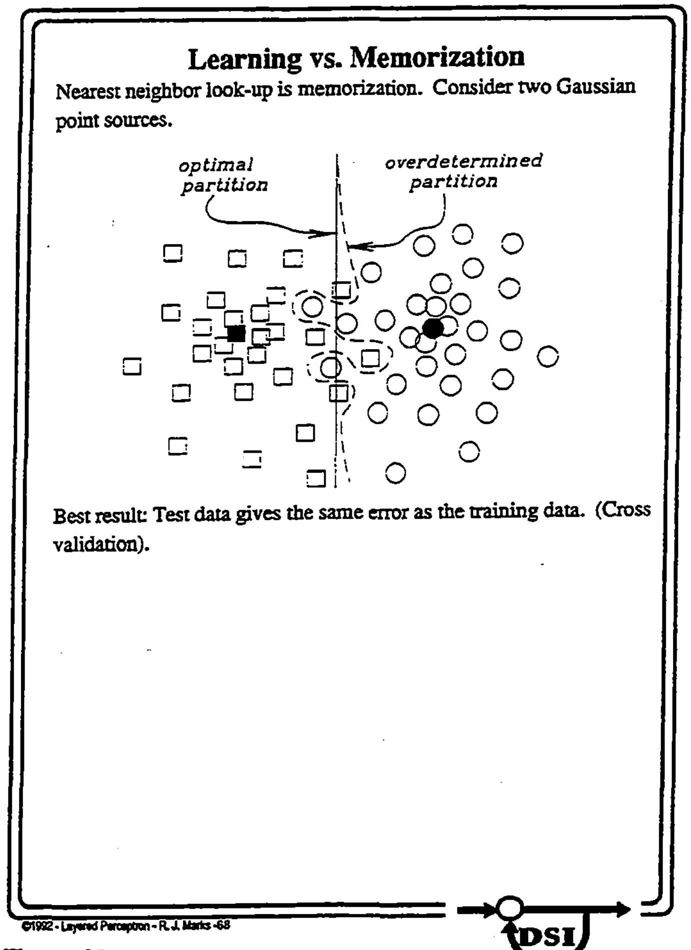
A two dimensional slice in a four dimensional classification space.

- ♦ Left: Training using 5000 randomly selected data points.
- ♦ Right: Training using 5000 query selected training data points.

©1992 - Layered Perceptron - R. J. Marks -67

Figure 67

We have applied these techniques to power security assessment. Here, an oracle for a single training vector query, can correspond to several minutes on a super computer. Details will be presented later.



Learning vs. Memorization

Nearest neighbor look-up is memorization. Consider two Gaussian point sources.

Best result: Test data gives the same error as the training data. (Cross validation).

©1992 - Layered Perceptron - R. J. Marks -68

Figure 68

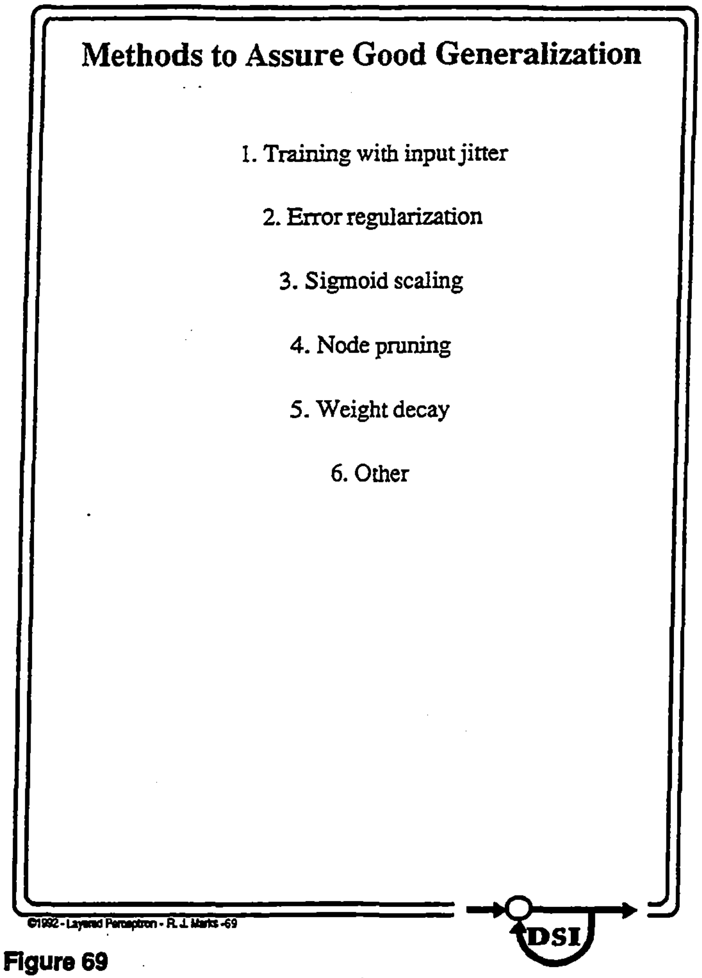
There is a difference between *training* and *memorization*. A trained classifier or regression machine can respond with confidence to a pattern which it has not seen before. The ability to properly classify data which has not been seen before is referred to as *generalization*. Memorization, on the other hand, guarantees that, when presented with a specific element in the training data set, the classifier will respond in exactly the same manner that it was trained. In the case of memorization, the response to data other than training data is not considered in the paradigm.

The ability to interpolate among the training data does not necessarily imply good generalization. We illustrate with an example from detection theory. Consider the two solid points shown here. The one on the left is a square and the one on the right is a circle. We assume the these are the centroids of 2 two-dimensional Gaussian random variables with the same variance. Given some observation point, the minimum probability of error solution results simply from determination of whether the

Notes Notes Notes

point lies to the right or the left of the perpendicular bisector between the two centroids. Consider, then, memorization from the training data shown by the hollow squares and circles. Since we require the classifier to properly categorize all points, the resulting partition boundary would follow the winding dashed line shown. Clearly, this line would become more winding with the increase of the data cardinality. This observation leads us to the conclusion that some trained classifiers should not generate a zero probability of error corresponding to the training data. This, rather, is memorization.

Are there cases where the error corresponding to training data should be zero? Yes. This is generally true when their is no noise or ambiguity in the data. How then, might we determine whether the classifier or regression machine has learned or memorized? The answer is that a properly trained classifier or regression machine should respond with the same error to training data as to test data. Note that this is a necessary though not sufficient condition. If the error from the test data is much higher than that from the training data, then, chances are, the neural system is over determined. In other words, the degrees of freedom in the classifier or regression machine is to high. For the layered perceptron, this is the number of interconnects which, of course, is related to the number of neurons in the hidden layer. If the error from the test and training data are similar, we are not guaranteed of proper training. Note, for example, that any partition line passing through the midpoint between the two centroids would result in a classifier with the same error for training and testing. Only the perpendicular bisector gives the unique minimum error solution.



Methods to Assure Good Generalization

1. Training with input jitter
2. Error regularization
3. Sigmoid scaling
4. Node pruning
5. Weight decay
6. Other

©1992 - Layered Perceptron - R. J. Marks -69

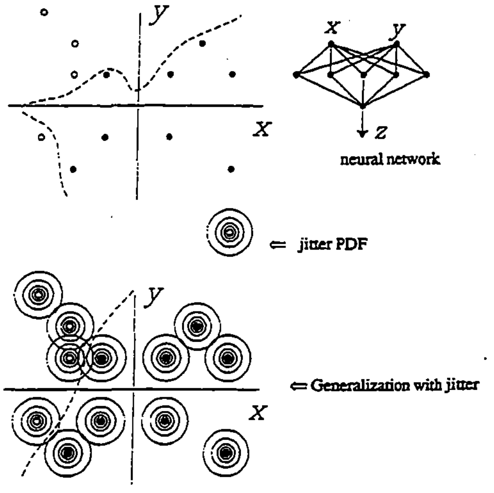
Figure 69

A well trained neural network must display good generalization. There exist numerous methods to assure good generalization. Each attempts, in essence, to match the classification capability of the neural network to that of the data. In most cases, the degrees of freedom of the neural network (*e.g.* its ability to generalize) are to great. The response of the layered perceptron must therefore be smoothed in some fashion. The techniques listed here are those most popularly used to do so.

TRAINING WITH JITTER

Instead of training with inputs (x,y), train with inputs $(x+n_1,y+n_2)$ where (n_1,n_2) is noise.

Generalization without jitter:



©1992 - Layered Perceptron - R. J. Marks -70

Figure 70

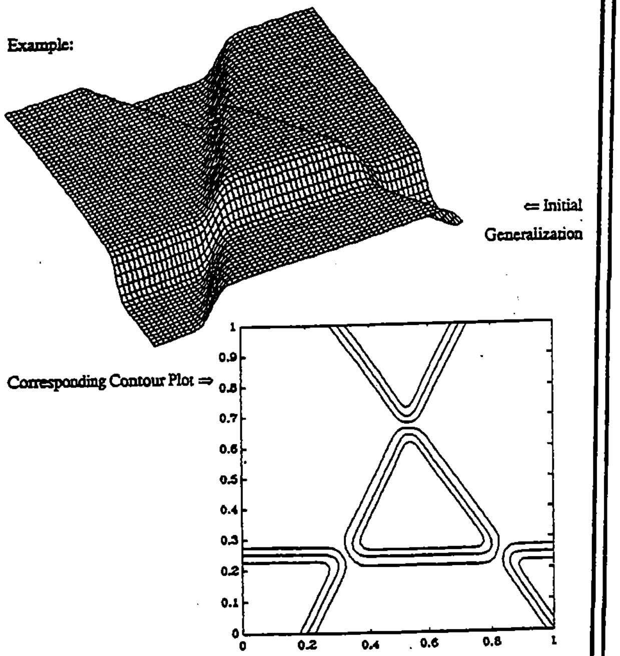
Training a neural network with to many degrees of freedom can result in poor generalization. The dashed line in the nonjittered generalization clearly results from to much freedom in generalization. Adding noise to the training data spreads the influence of each of the training datum to a larger volume. The result, as illustrated in the bottom figure, is a much smoother generalization.

The expected value of the effective target, in the case of jitter, is the (multidimensional) convolution of the original target function with the probability density function (PDF) of the jitter.

Sigmoid Scaling

After training, replace the sigmoid function, $s(x)$, with $s(x/\sigma)$.
σ is a function of the weights of the network

Example:



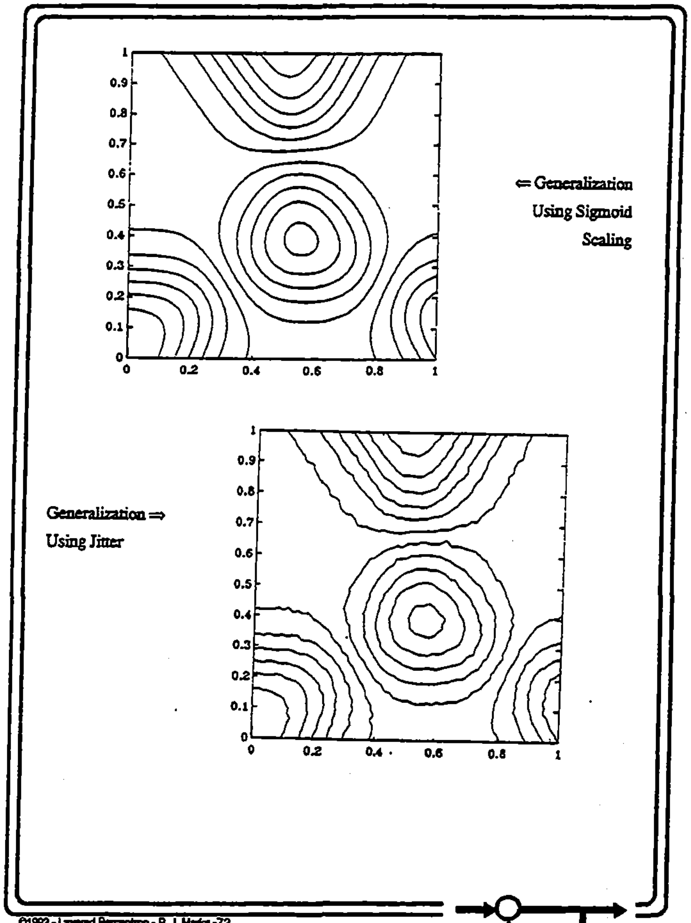
⇐ Initial Generalization

Corresponding Contour Plot ⇒

Figure 71

Better generalization can also be achieved using sigmoid scaling. The neural network is trained as usual. After training, the slopes of the sigmoids can then be scaled to acheive better generalization. Typically, the slope of the sigmoid is decreased. The amount of scaling performed is a function of the weights used.

Notes Notes Notes



⇐ Generalization Using Sigmoid Scaling

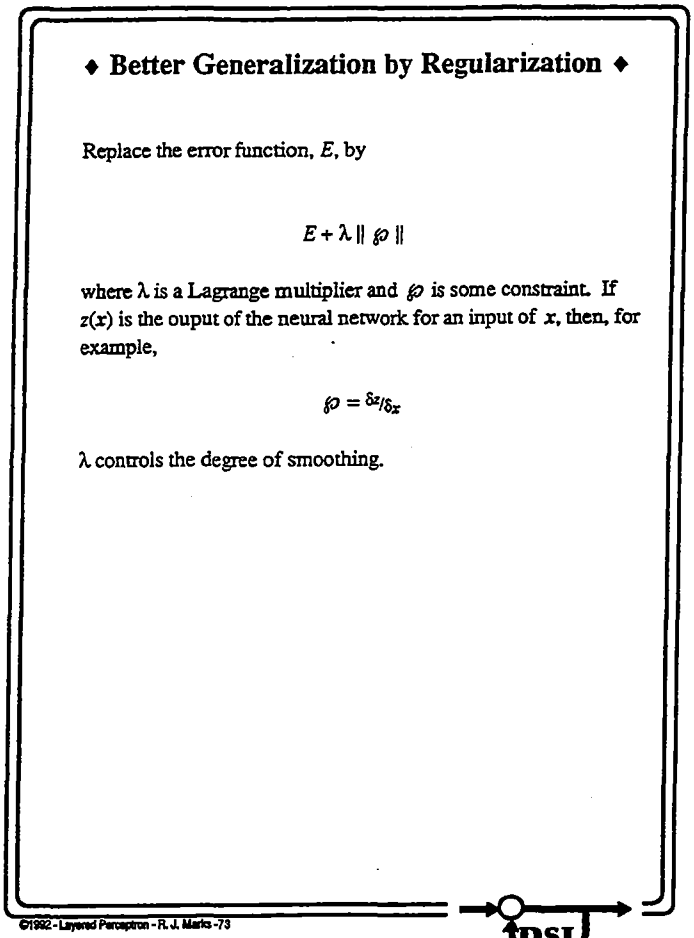
Generalization ⇒ Using Jitter

©1992 - Layered Perceptron - R. J. Marks -72

Figure 72

Shown are examples of generalization using sigmoid scaling and jitter. The effects of sigmoid scaling was chosen to give results similar to that of jitter. In terms of training time, jitter typically takes much more time. The network, in essence, must adapt to a much greater degree of data cardinality.

Question: What is the best choice of σ? This can be determined by cross validation (borate).



◆ Better Generalization by Regularization ◆

Replace the error function, E, by

$$E + \lambda \| \wp \|$$

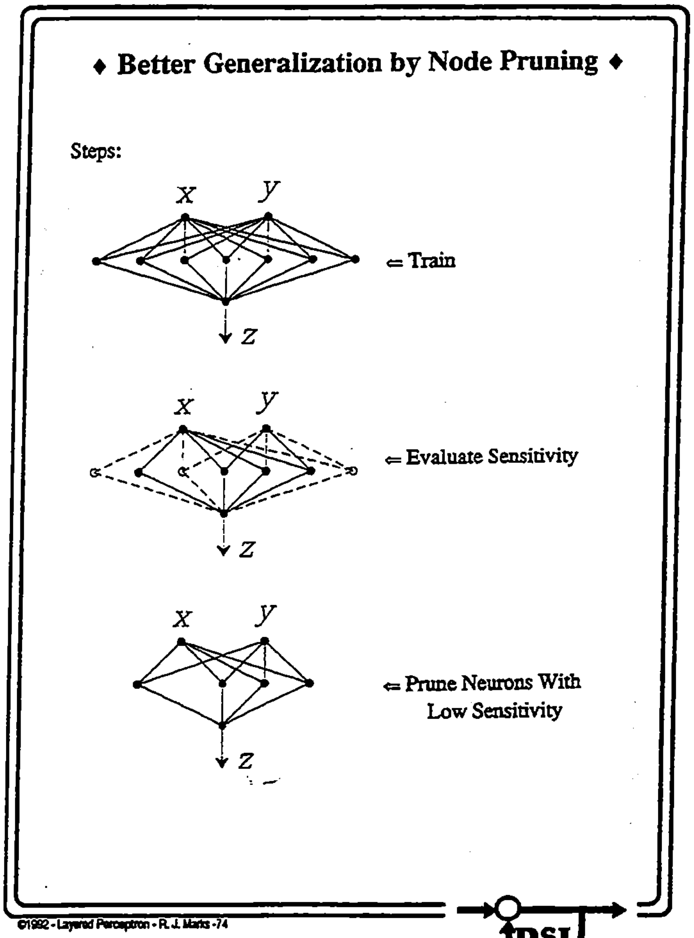
where λ is a Lagrange multiplier and $\wp$ is some constraint. If $z(x)$ is the ouput of the neural network for an input of x, then, for example,

$$\wp = {}^{\delta z}/_{\delta x}$$

λ controls the degree of smoothing.

Figure 73

The Lagrange multiplier, λ, corresponds to the σ encountered in training with jittered data. Using the method of regularization generally takes training outside of the structure of neural networks. Backpropagation can be performed within the neural network structure. Imposition of regularation perturbs the math of error back propagation to where it can no longer be performed by updating weights as a function to the states and parameters of the two neurons which they connect. For small σ, training with jitter gives results similar to that of regularization with the differential constraint.



♦ Better Generalization by Node Pruning ♦

Steps:

⇐ Train

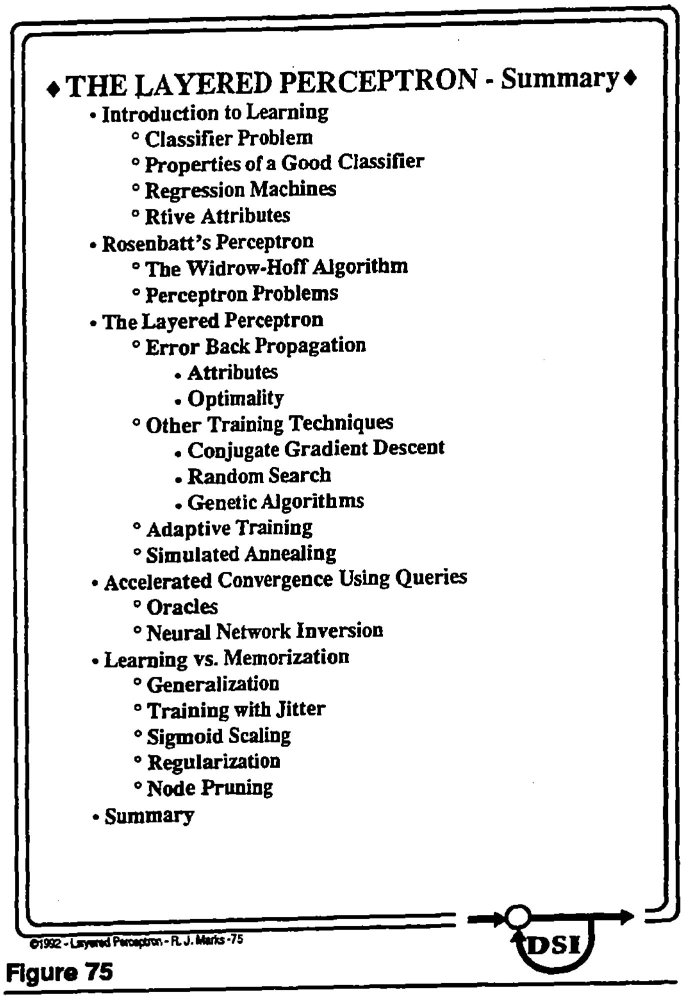
⇐ Evaluate Sensitivity

⇐ Prune Neurons With Low Sensitivity

©1992 - Layered Perceptron - R. J. Marks - 74

Figure 74

Node pruning attempts to match the degrees of freedom of the neural network with that of the data. The neural network is first trained. The sensitivity of the hidden nodes is then computed. The sensitivity is the rate of change (partial derivative) of the output with respect to the hidden neuron state. If this sensitivity (slope) is small, then large swings of the neuron state has little effect on the output. Thus, the node is not necessary and can be deleted, or pruned.



♦THE LAYERED PERCEPTRON - Summary♦

- Introduction to Learning
  - ° Classifier Problem
  - ° Properties of a Good Classifier
  - ° Regression Machines
  - ° Rtive Attributes
- Rosenbatt's Perceptron
  - ° The Widrow-Hoff Algorithm
  - ° Perceptron Problems
- The Layered Perceptron
  - ° Error Back Propagation
    - Attributes
    - Optimality
  - ° Other Training Techniques
    - Conjugate Gradient Descent
    - Random Search
    - Genetic Algorithms
  - ° Adaptive Training
  - ° Simulated Annealing
- Accelerated Convergence Using Queries
  - ° Oracles
  - ° Neural Network Inversion
- Learning vs. Memorization
  - ° Generalization
  - ° Training with Jitter
  - ° Sigmoid Scaling
  - ° Regularization
  - ° Node Pruning
- Summary

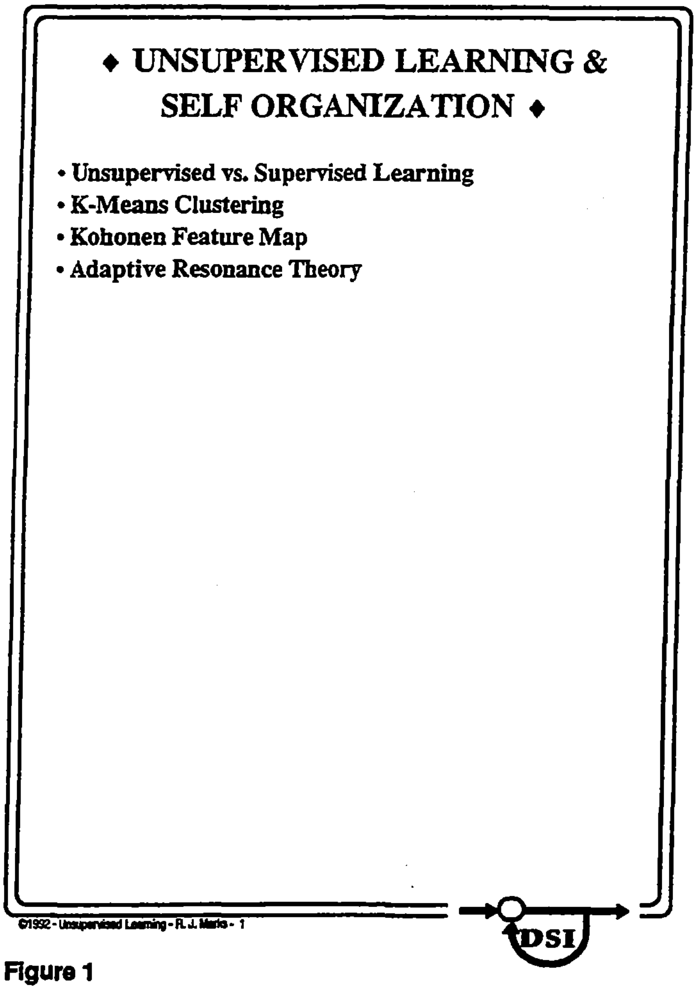
©1992 - Layered Perceptron - R. J. Marks -75

DSI

Figure 75

Notes Notes Notes

Unsupervised Learning
& Self Organization



♦ UNSUPERVISED LEARNING & SELF ORGANIZATION ♦

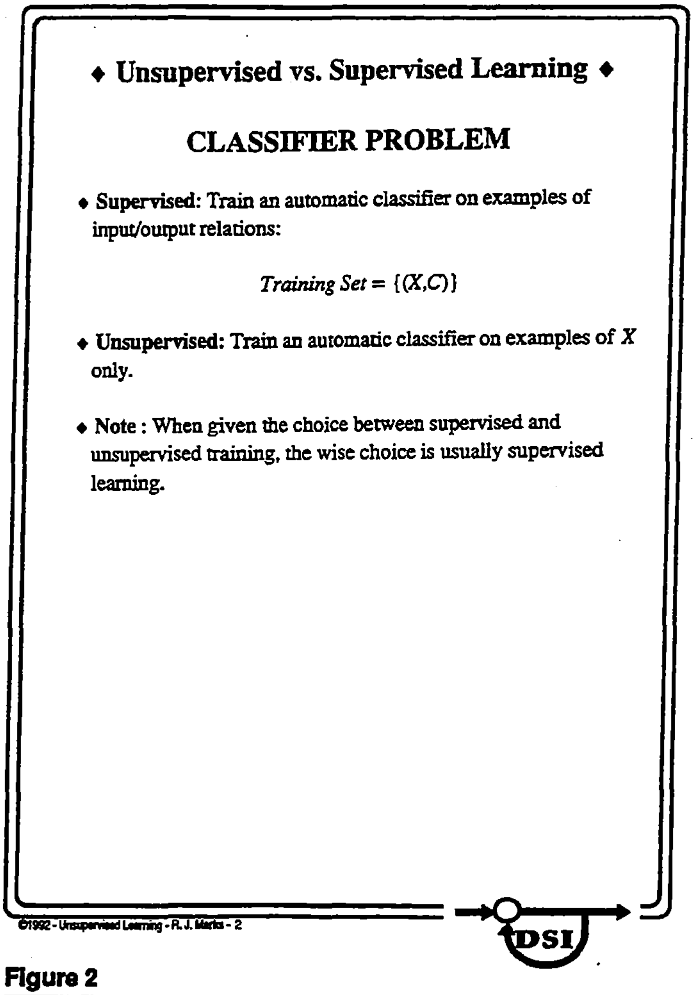
- Unsupervised vs. Supervised Learning
- K-Means Clustering
- Kohonen Feature Map
- Adaptive Resonance Theory

Figure 1

Notes Notes Notes

The layered perceptron learns through supervised training. One is given an input and a corresponding target. The neural network adjusts to match the inputs with their targets. For unsupervised training, there are no targets provided. The neural network must decide, without supervision, how the inputs are to be grouped. Such a procedure is referred to as *clustering*.

Notes Notes Notes



◆ Unsupervised vs. Supervised Learning ◆

CLASSIFIER PROBLEM

- ◆ **Supervised:** Train an automatic classifier on examples of input/output relations:

$$Training\ Set = \{(X,C)\}$$

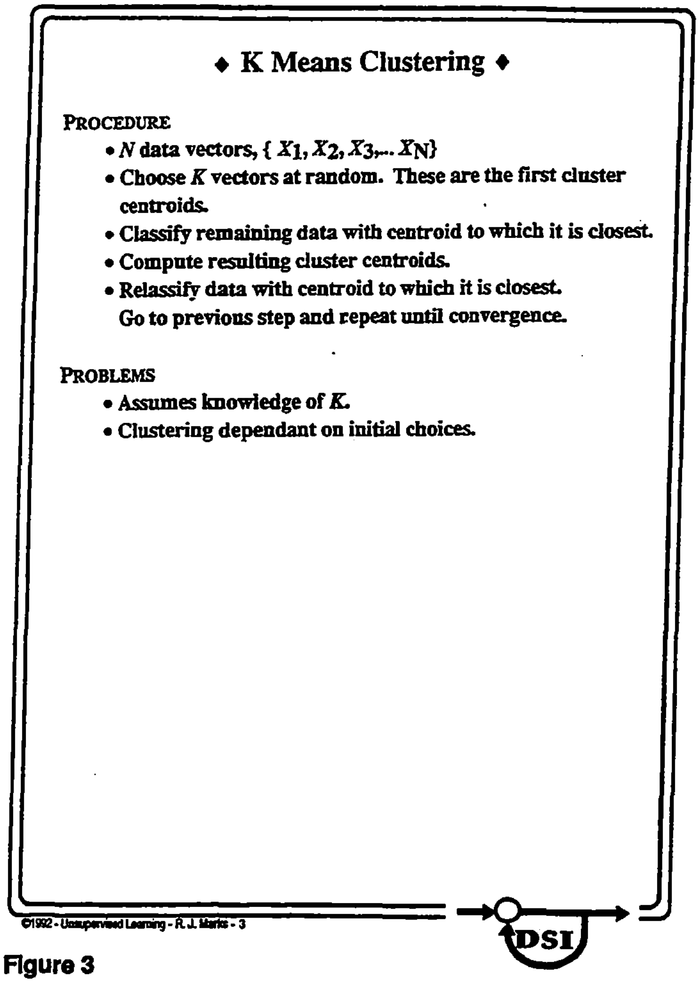
- ◆ **Unsupervised:** Train an automatic classifier on examples of X only.

- ◆ Note : When given the choice between supervised and unsupervised training, the wise choice is usually supervised learning.

©1992 - Unsupervised Learning - R. J. Marks - 2

Figure 2

The categorization is not available to us in unsupervised learning. The classifier must decide, when presented the data, to which class each belongs. Since more information is available in supervised learning, it should be chosen when an option exists.



◆ K Means Clustering ◆

PROCEDURE

- N data vectors, $\{ X_1, X_2, X_3,... X_N\}$
- Choose K vectors at random. These are the first cluster centroids.
- Classify remaining data with centroid to which it is closest.
- Compute resulting cluster centroids.
- Relassify data with centroid to which it is closest. Go to previous step and repeat until convergence.

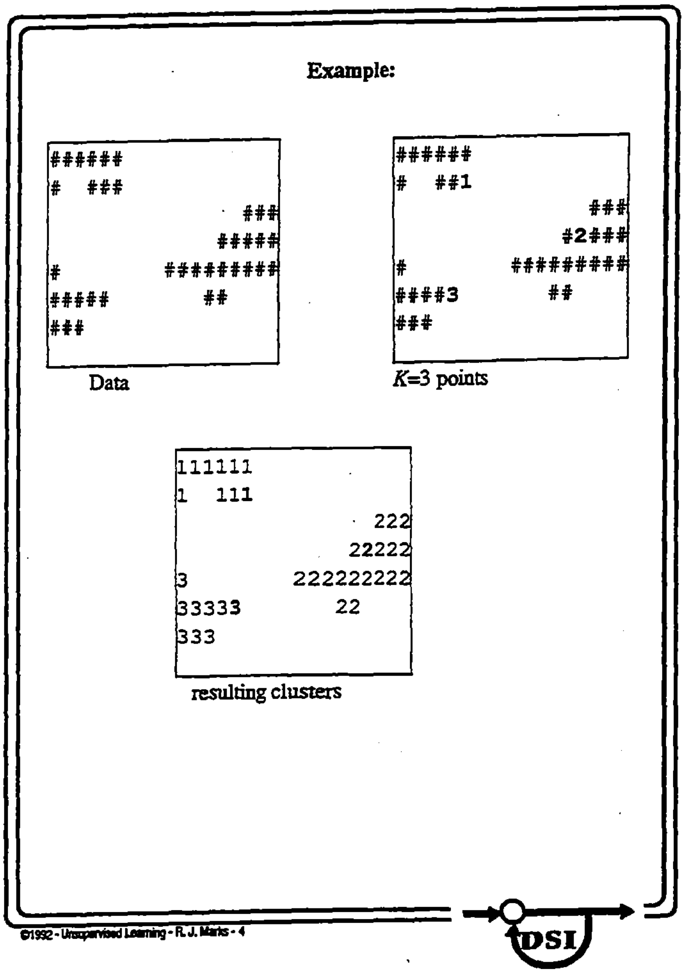
PROBLEMS

- Assumes knowledge of K.
- Clustering dependant on initial choices.

©1992 - Unsupervised Learning - R. J. Marks - 3

Figure 3

K-means clustering is a very simple unsupervised learning procedure. An assumption of *K* centroids is made. The data is categorized in accordance to these centroids. The cluster centroids are recomputed and the procedure is repeated until convergence.



Example:

```
######
#  ###
                ###
              #####
#         #########
#####        ##
###
```

Data

```
######
#  ##1
                ###
              #2###
#         #########
####3        ##
###
```

K=3 points

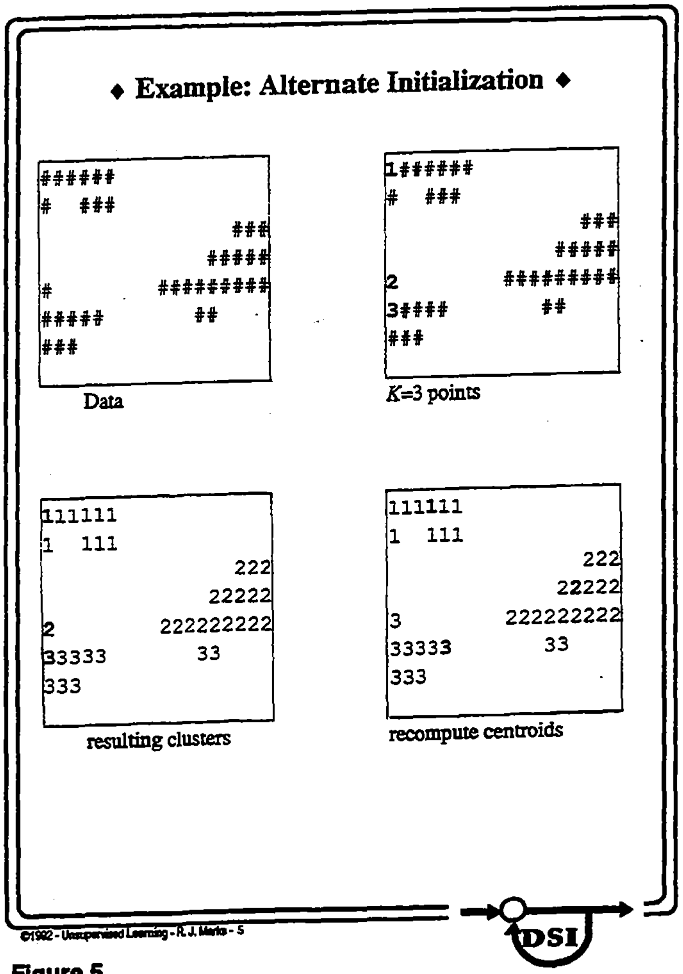
```
111111
1  111
                222
              22222
3         222222222
33333        22
333
```

resulting clusters

©1992 - Unsupervised Learning - R. J. Marks - 4

Figure 4

A set of two dimensional data is shown. Three data points are chosen as the centroid and the resulting clusters evolve.



◆ Example: Alternate Initialization ◆

```
######
#  ###
               ###
             #####
#        #########
#####       ##
###
```

Data

```
1######
#  ###
               ###
             #####
2        #########
3####       ##
###
```

K=3 points

```
111111
1  111
               222
             22222
2         222222222
33333        33
333
```

resulting clusters

```
111111
1  111
               222
             22222
3         222222222
33333        33
333
```

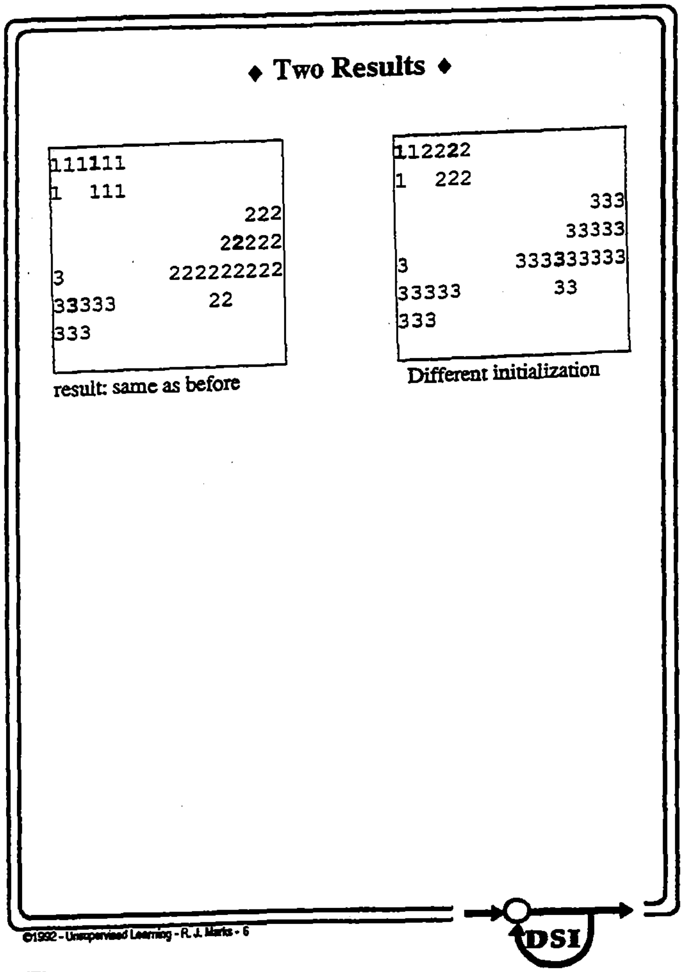
recompute centroids

©1992 - Unsupervised Learning - R. J. Marks - 5

Figure 5

This alternate initialization, as shown on the next page, converges to the same result as before.

Notes Notes Notes



♦ Two Results ♦

```
111111
1  111
               222
             22222
3        222222222
33333       22
333
```

result: same as before

```
112222
1  222
               333
             33333
3        333333333
33333        33
333
```

Different initialization

©1992 - Unsupervised Learning - R. J. Marks - 6

Figure 6

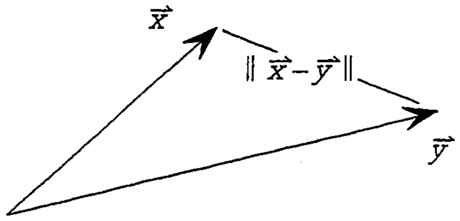
The initialization on the right converges to the clustering shown and is significantly different from the previous results.

♦ Minimum Distance and Correlation ♦

The distance between two vectors, *x* and *y* is

$$\| x - y \|^2 = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_N - y_N)^2$$

$$= (x - y)^T (x - y) = \| x \|^2 + \| y \|^2 - 2x^T y$$



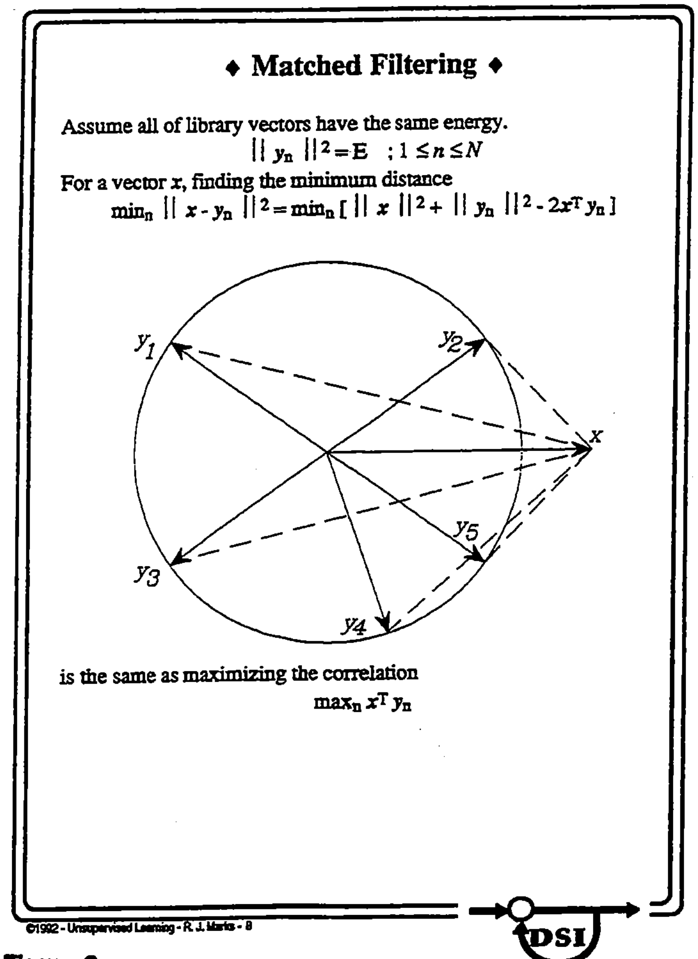
where the *correlation* between *x* and *y* is

$$x^T y = x_1 y_1 + x_2 y_2 + \ldots + x_3 y_3$$

©1992 - Unsupervised Learning - R. J. Marks - 7

Figure 7

In many classification problems, an observed vector is compared to a library of vectors. In order to determine which of the library vectors the observation is, the mean square distance between the vectors is computed.



◆ Matched Filtering ◆

Assume all of library vectors have the same energy.

$$\| y_n \|^2 = E \quad ; 1 \le n \le N$$

For a vector x, finding the minimum distance

$$\min_n \| x - y_n \|^2 = \min_n [\, \| x \|^2 + \| y_n \|^2 - 2x^T y_n \,]$$

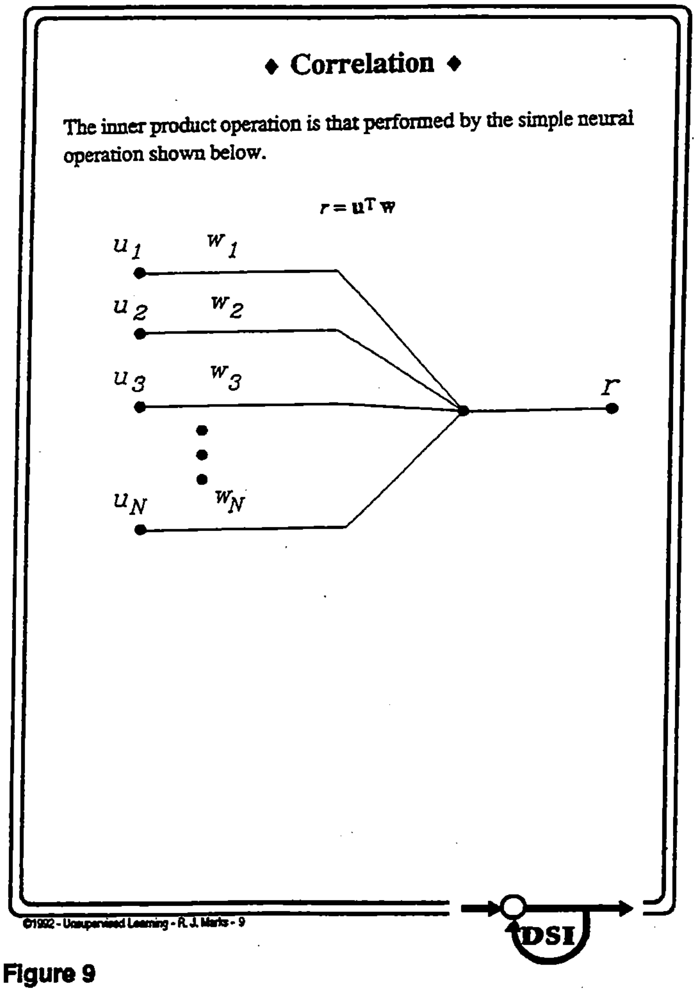
is the same as maximizing the correlation

$$\max_n x^T y_n$$

©1992 - Unsupervised Learning - R. J. Marks - 8

Figure 8

If all library vectors have the same norm, they all lie on the surface of a hypersphere. Since $\|y_n\|^2$ is always constant, minimizing the norm is the same as maximizing the correlation (inner product). This operation is referred to as *matched filtering*.



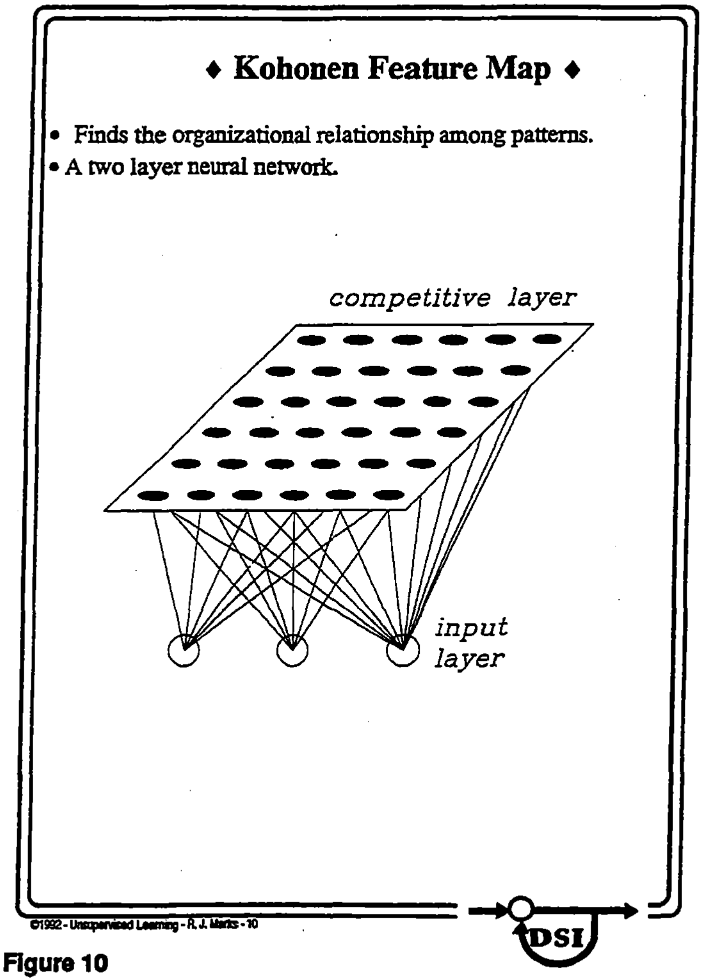
♦ Correlation ♦

The inner product operation is that performed by the simple neural operation shown below.

$$r = \mathbf{u}^T \mathbf{w}$$

Figure 9

The architecture shówn is that of a simple perceptron. The use of the inner product in a matched filter role is used in the Hopfield associative memory, the perceptron, Kohonen's feature map and ART.



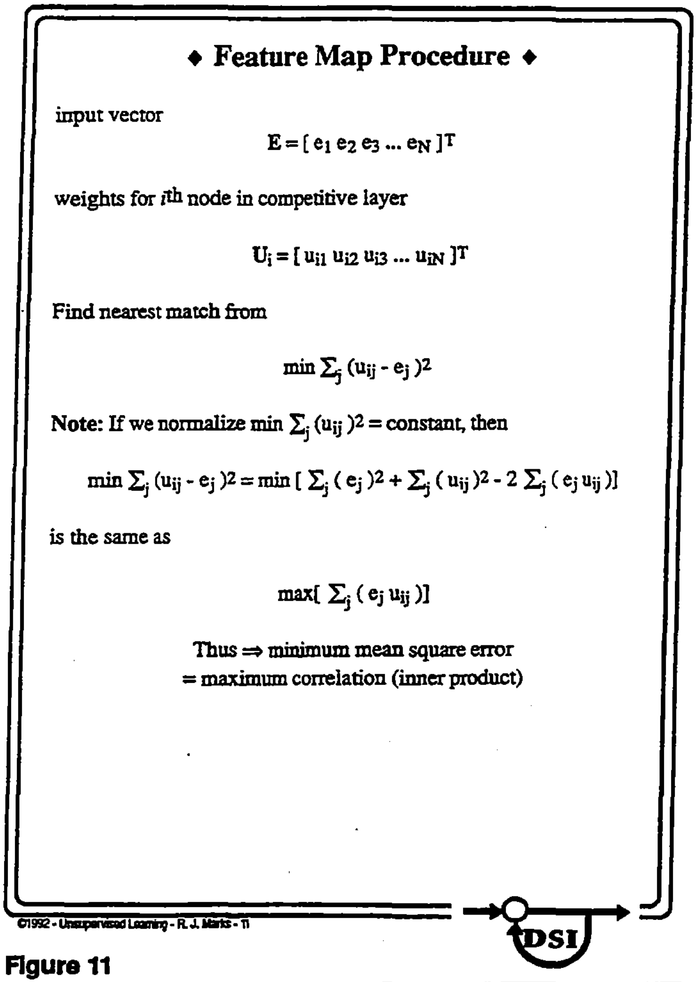
♦ Kohonen Feature Map ♦

- Finds the organizational relationship among patterns.
- A two layer neural network.

©1992 - Unsupervised Learning - R. J. Marks - 10

Figure 10

Each node in the competitive layer corresponds to the centroid of a cluster. The interconnects to a neuron in the hidden layer contain the information about the location of this centroid.



◆ Feature Map Procedure ◆

input vector

$$E = [\, e_1\ e_2\ e_3 \ldots e_N \,]^T$$

weights for i^{th} node in competitive layer

$$U_i = [\, u_{i1}\ u_{i2}\ u_{i3} \ldots u_{iN} \,]^T$$

Find nearest match from

$$\min \textstyle\sum_j (u_{ij} - e_j)^2$$

Note: If we normalize min $\sum_j (u_{ij})^2$ = constant, then

$$\min \textstyle\sum_j (u_{ij} - e_j)^2 = \min [\, \sum_j (e_j)^2 + \sum_j (u_{ij})^2 - 2 \sum_j (e_j u_{ij})]$$

is the same as

$$\max[\, \textstyle\sum_j (e_j u_{ij})]$$

Thus ⇒ minimum mean square error
= maximum correlation (inner product)

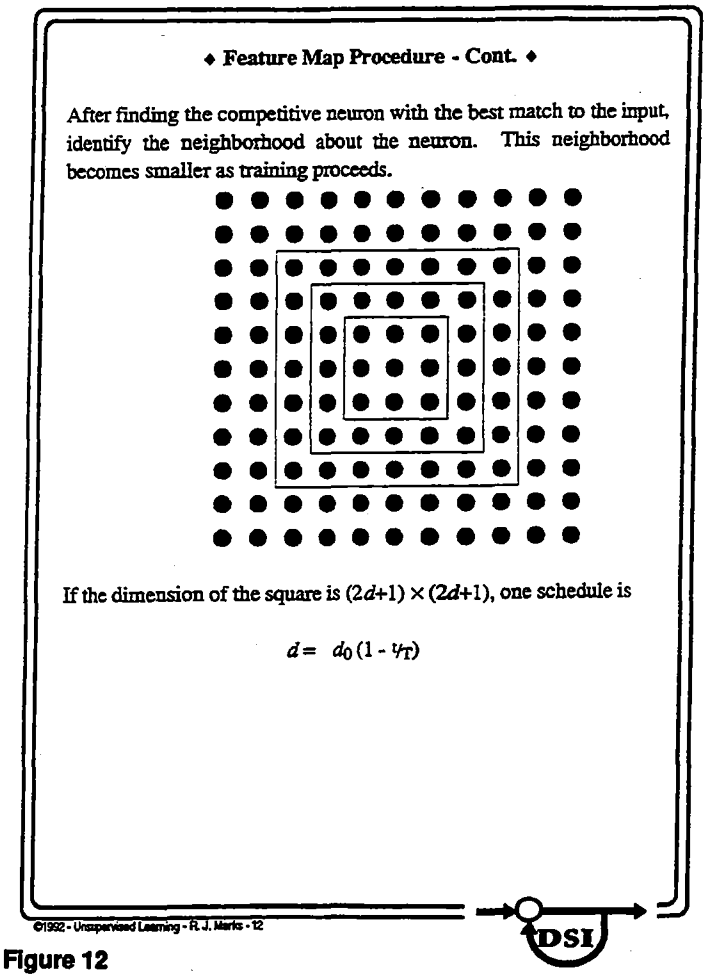
©1992 - Unsupervised Learning - R. J. Marks - 11

DSI

Figure 11

Notes Notes Notes

The first step is to find that neuron in the competitive layer that is closest to the input vector in the mean square sense. If the weights to each of the neurons in the competitive layer are normalized, this is the same as maximizing the correlation. In other words, the best match is determined by a matched filter.



◆ Feature Map Procedure - Cont. ◆

After finding the competitive neuron with the best match to the input, identify the neighborhood about the neuron. This neighborhood becomes smaller as training proceeds.

If the dimension of the square is (2d+1) × (2d+1), one schedule is

$$d = d_0 (1 - {}^{t}\!/_{T})$$

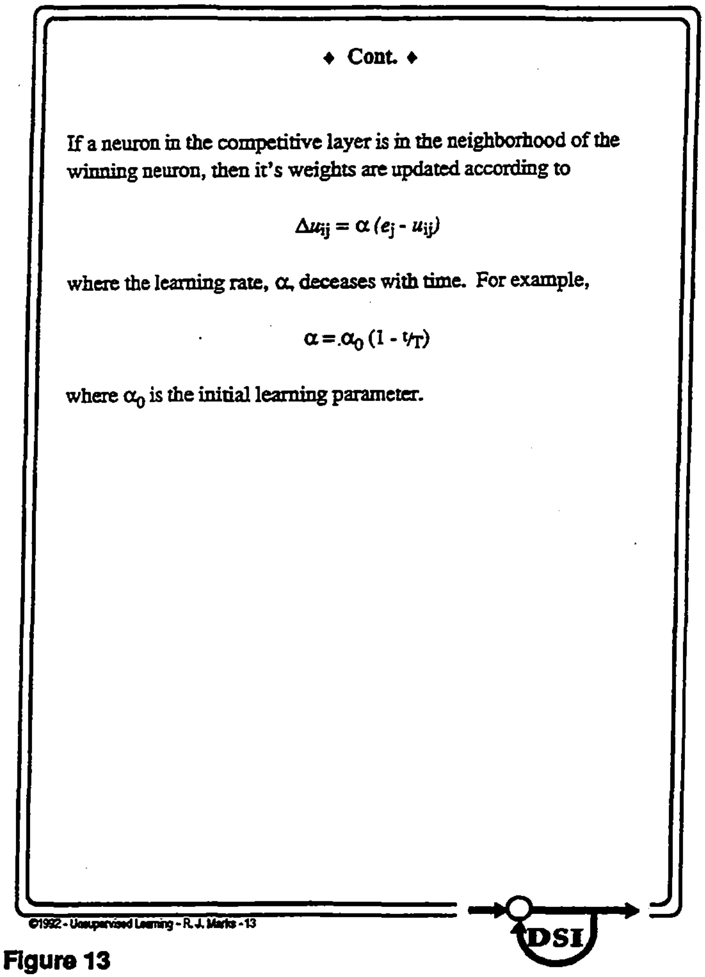
©1992 - Unsupervised Learning - R. J. Marks - 12

Figure 12

We have identified the neuron in the competitive layer that has the best match. A neighborhood around this neuron is specified. Here, we show three boxes around the neuron that has been identified. As time increases, the size of these boxes decreases. If T is the total training time, a linear schedule is

$$d = d_0 \,(1 - {}^{t}\!/_{T})$$

where d_0 is the initial size of the box.



◆ Cont. ◆

If a neuron in the competitive layer is in the neighborhood of the winning neuron, then it's weights are updated according to

$$\Delta u_{ij} = \alpha (e_j - u_{ij})$$

where the learning rate, α, deceases with time. For example,

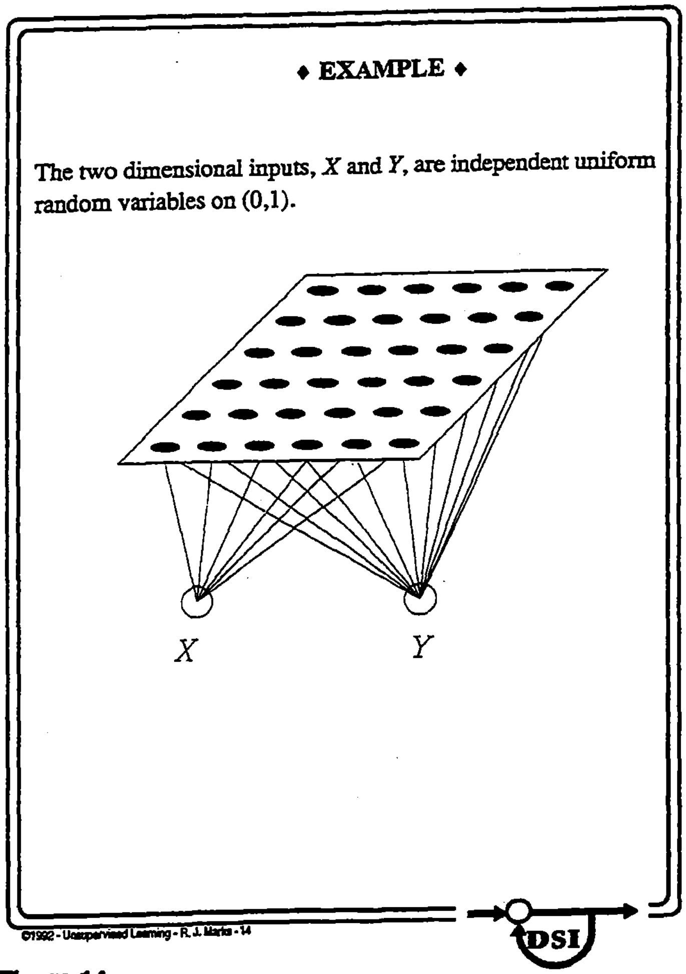
$$\alpha = .\alpha_0 (1 - t/T)$$

where α_0 is the initial learning parameter.

©1992 - Unsupervised Learning - R. J. Marks - 13

Figure 13

Once the neighborhood has been identified, the weights to those neurons are updated. The weights are updated to make them look more like the input. As time progresses, however, the weights should be more and more difficult to change. Thus, the learning rate parameter is decreased with respect to time.



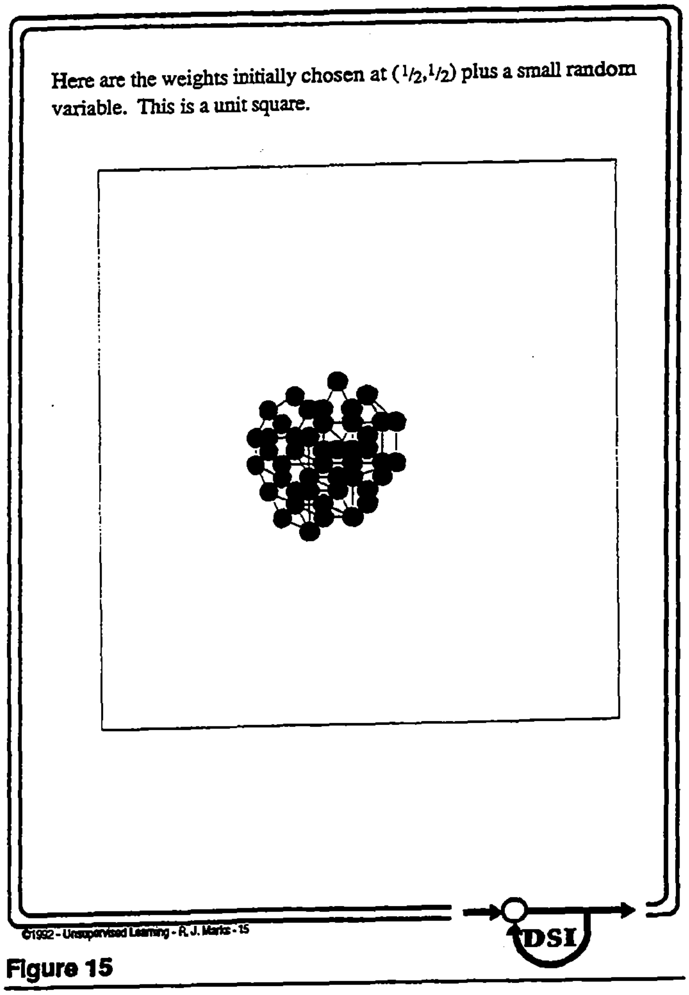
◆ EXAMPLE ◆

The two dimensional inputs, X and Y, are independent uniform random variables on (0,1).

©1992 - Unsupervised Learning - R. J. Marks - 14

Figure 14

Each of the neurons in the competitive layer has two inputs - one from X and one from Y. These two weights form an ordered pair. We expect these weights to distribute themselves in accordance of the input probability distribution.

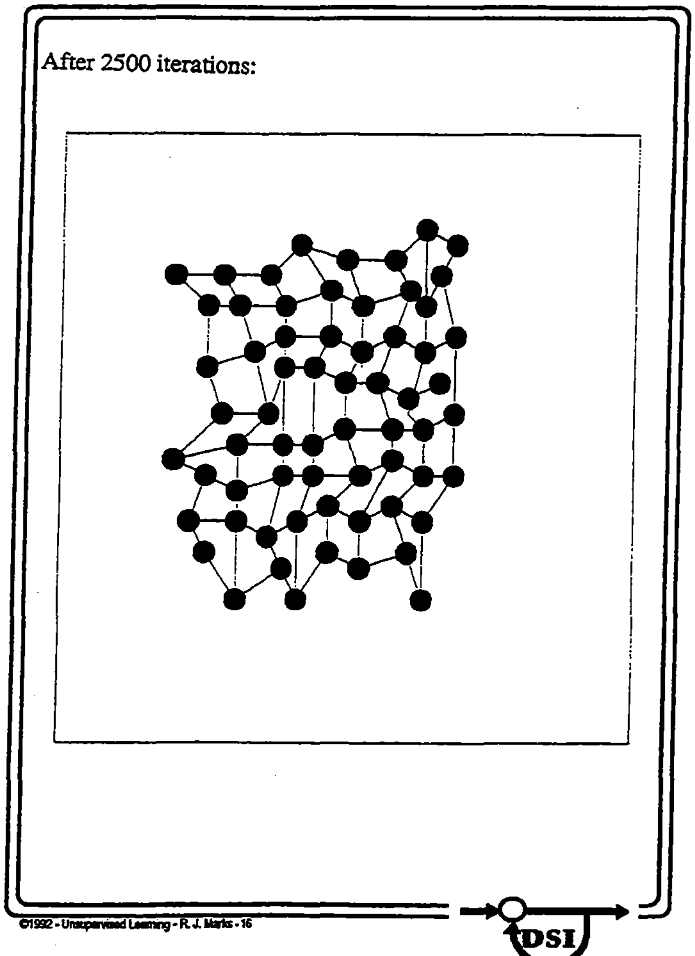


Here are the weights initially chosen at ($^1/_2$,$^1/_2$) plus a small random variable. This is a unit square.

Figure 15

The initial values of the net are chosen using a random number generator. The square is over the intervals (0,1) in both X and Y.

Notes Notes Notes



After 2500 iterations:

Figure 16

This is a result we would expect after 2500 iterations. The weights of the neurons in the competitive layer are beginning to adapt to the density of the inputs. The points are connected for display purposes only.

After over 10,000 iterations. The partition has spread almost uniformly over the square in accordance to the input distribution.

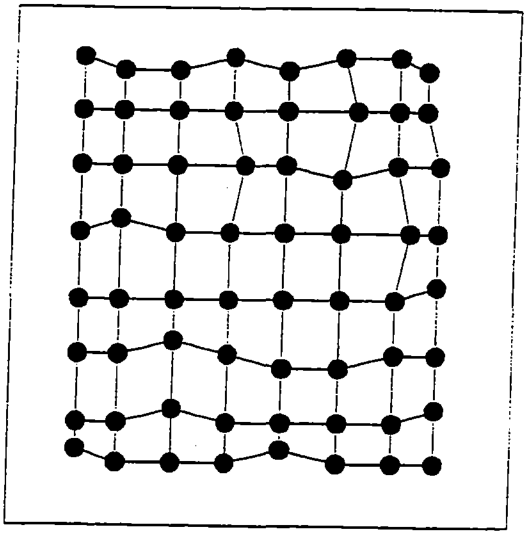
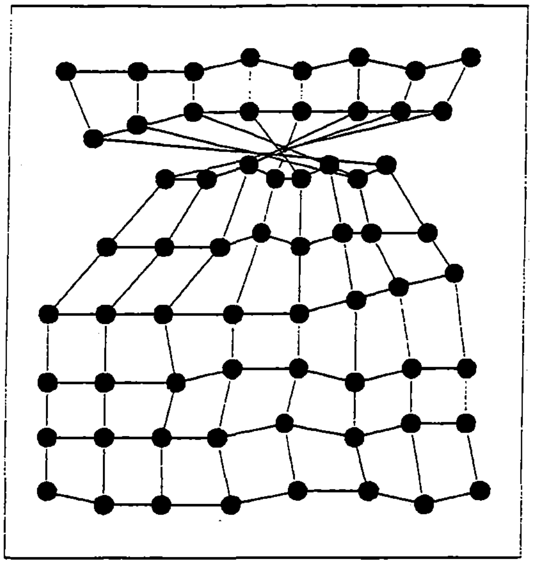


Figure 17

This is a result we would expect after 10,500 iterations. The net has nearly converged. Note the edge effects. There are no input vectors outside of the square. Thus, there are no vectors there to 'pull' the graph to the edge.

Kohonen maps can also converge inproperly, dependent on data ordering and the initial states. For example:



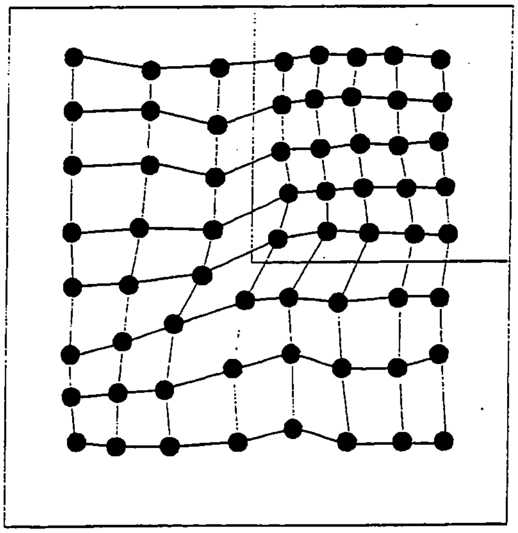
©1992 - Unsupervised Learning - R. J. Marks - 18

Figure 18

This Kohonen map has clearly converged to an improper result. In some applications, the incorrect result might not be as apparent.

◆ Nonuniform Probability ◆

In this example, the probability of choosing a test data point in the upper right hand corner is higher.



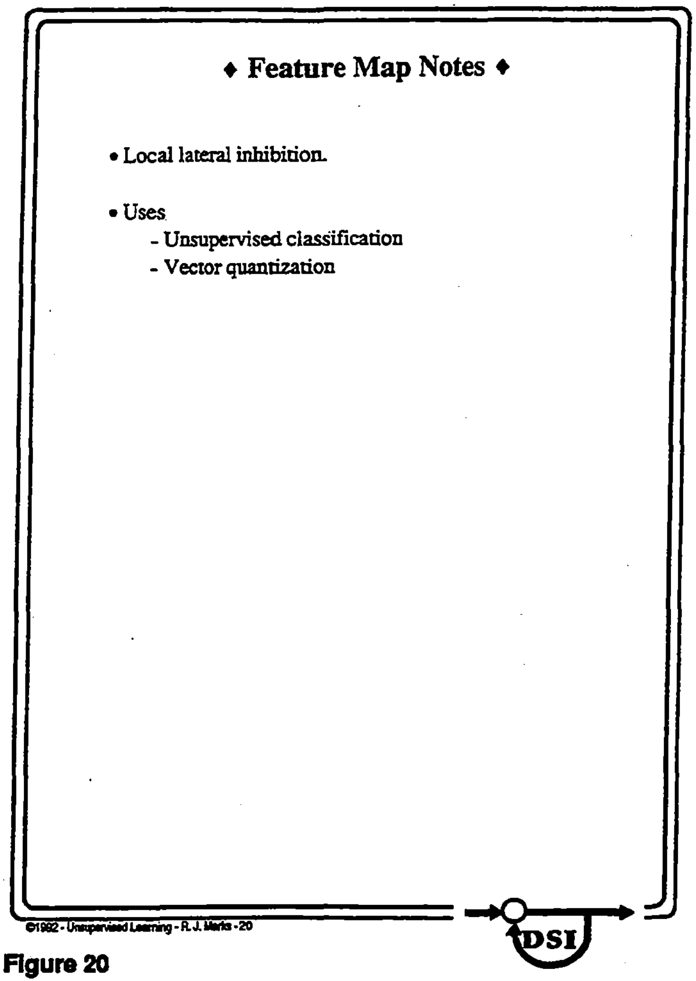
©1992 - Unsupervised Learning - R. J. Marks - 19

DSI

Figure 19

Here the net's weights have converged to other than a somewhat uniform spacing. The reason is that the joint probability density function is other than uniform. The density in the upper right hand square is larger than that elsewhere. For a large number of neurons in the competitive layer, corresponding to a large number of points on this graph, we would expect that the density of dots to be proportional to the probability density function there.

Notes Notes Notes



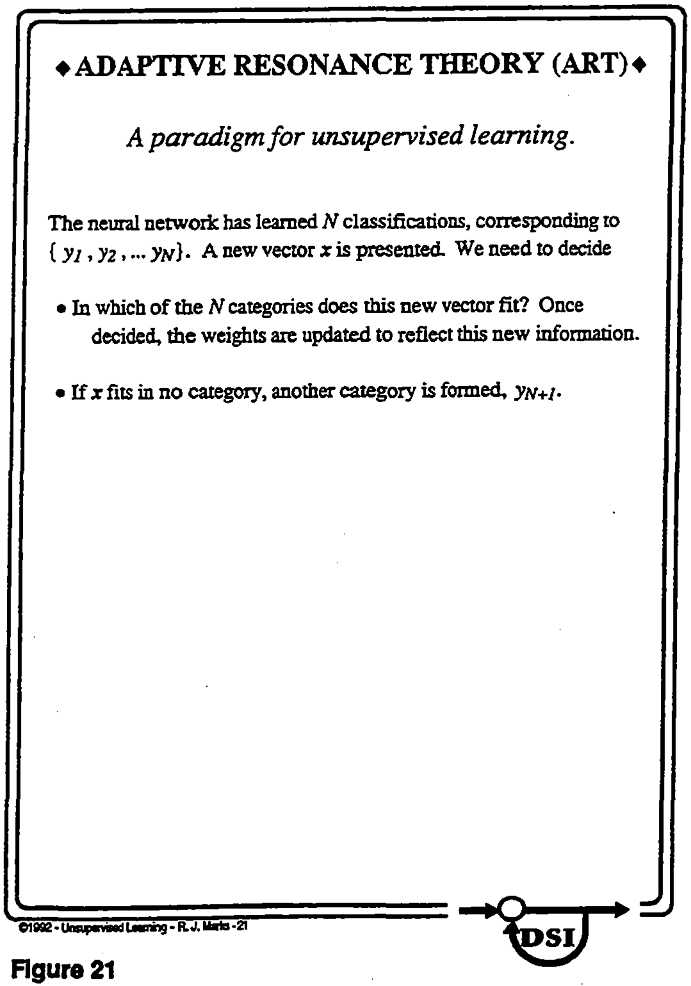
♦ Feature Map Notes ♦

- Local lateral inhibition.
- Uses
  - Unsupervised classification
  - Vector quantization

©1992 - Unsupervised Learning - R. J. Marks - 20

Figure 20

A variation of the decreasing area of learning is the a windowing of the region. One use of the Kohonen feature map is vector quantization. Suppose we have *N* vectors of length *L*, and that many of these vectors were similar. The feature map is used to sort these vectors into classes. Then, when we wish to communicate a vector in class C, we broadcast a C instead of the whole vector. At the receiver, a C is reconstructed as a class C vector through use of a code book.



◆ADAPTIVE RESONANCE THEORY (ART)◆

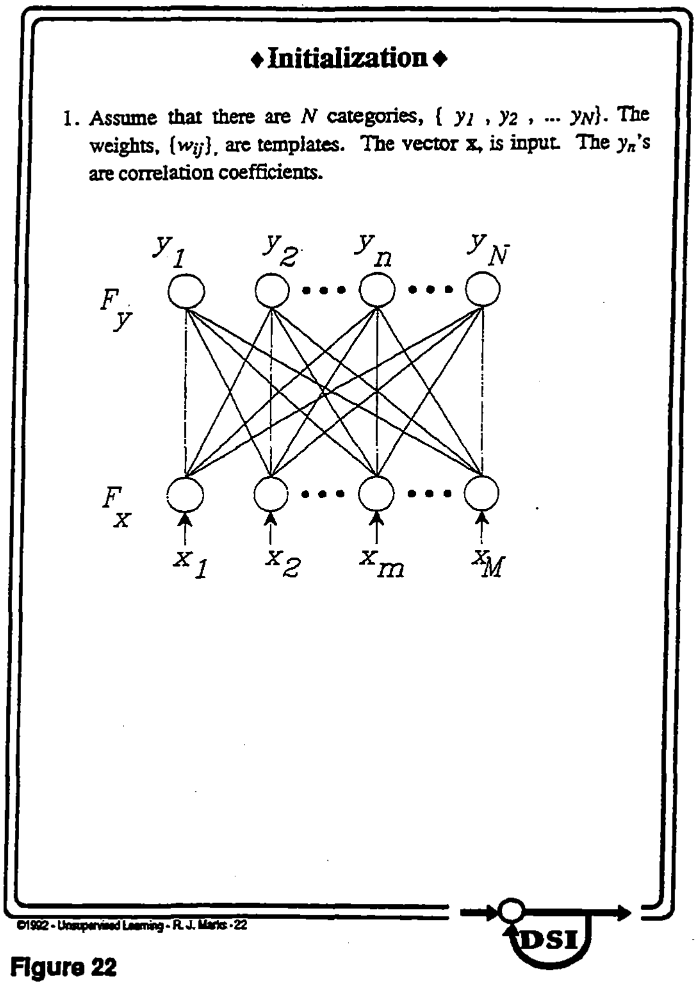
A paradigm for unsupervised learning.

The neural network has learned N classifications, corresponding to $\{ y_1 , y_2 , ... y_N \}$. A new vector x is presented. We need to decide

- In which of the N categories does this new vector fit? Once decided, the weights are updated to reflect this new information.
- If x fits in no category, another category is formed, y_{N+1}.

Figure 21

This neural network has been called 'one of the most complex neural networks ever invented' by Maureen Caudill. She continues, 'When these [ART units] are implemented in a software simulation (such as that provided by at least one of the available commercial simulators), computational overhead is so great that the neural network is unacceptably slow on anything short of a Cray.' This has been the perception of ART. The basic idea behind ART, however, is quite straightforward. We here describe its essence.



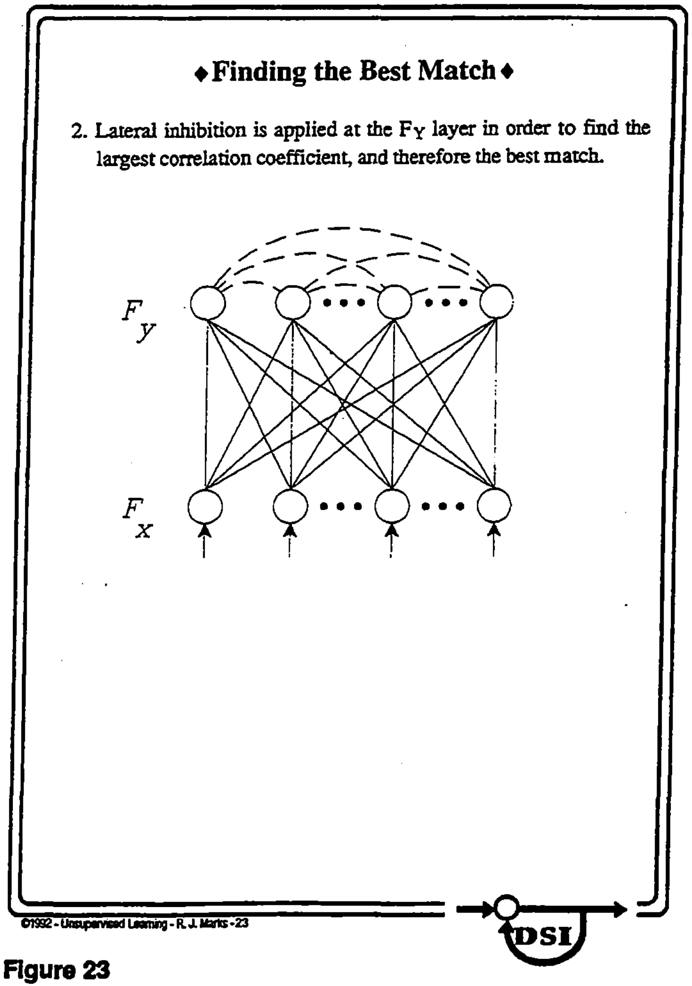
♦Initialization♦

1. Assume that there are N categories, { y_1 , y_2 , ... y_N}. The weights, {w_{ij}}, are templates. The vector **x**, is input. The y_n's are correlation coefficients.

©1992 - Unsupervised Learning - R. J. Marks - 22

Figure 22

Assume that the neural network has already learned some pattern classes. A new vector, **x**, is introduced. The neural net weights are used to compute the correlation coefficients of the templates.



♦Finding the Best Match♦

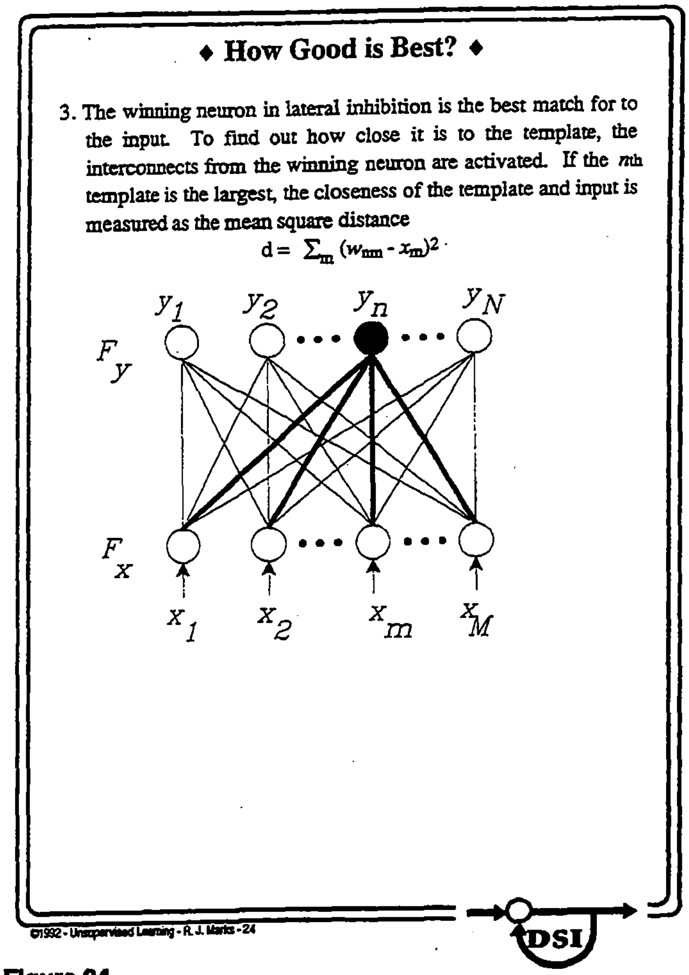
2. Lateral inhibition is applied at the F_Y layer in order to find the largest correlation coefficient, and therefore the best match.

©1992 - Unsupervised Learning - R. J. Marks - 23

Figure 23

The coefficients in the F_Y layer are subjected to lateral inhibition in order to find the maximum coefficient.

Notes Notes Notes



♦ How Good is Best? ♦

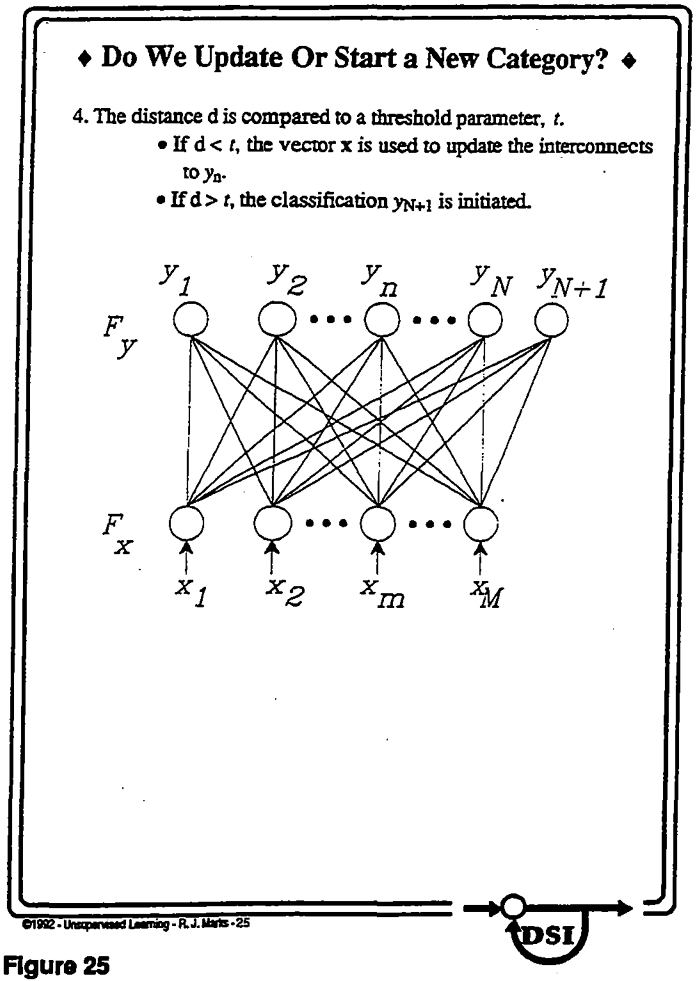
3. The winning neuron in lateral inhibition is the best match for to the input. To find out how close it is to the template, the interconnects from the winning neuron are activated. If the *n*th template is the largest, the closeness of the template and input is measured as the mean square distance

$$d = \sum_m (w_{nm} - x_m)^2$$

©1992 - Unsupervised Learning - R. J. Marks - 24

Figure 24

The winning category excites the interconnects back to the input layer. The template is compares to the input and the corresponding mean square distance is computed and compared with a threshold parameter, *t*.



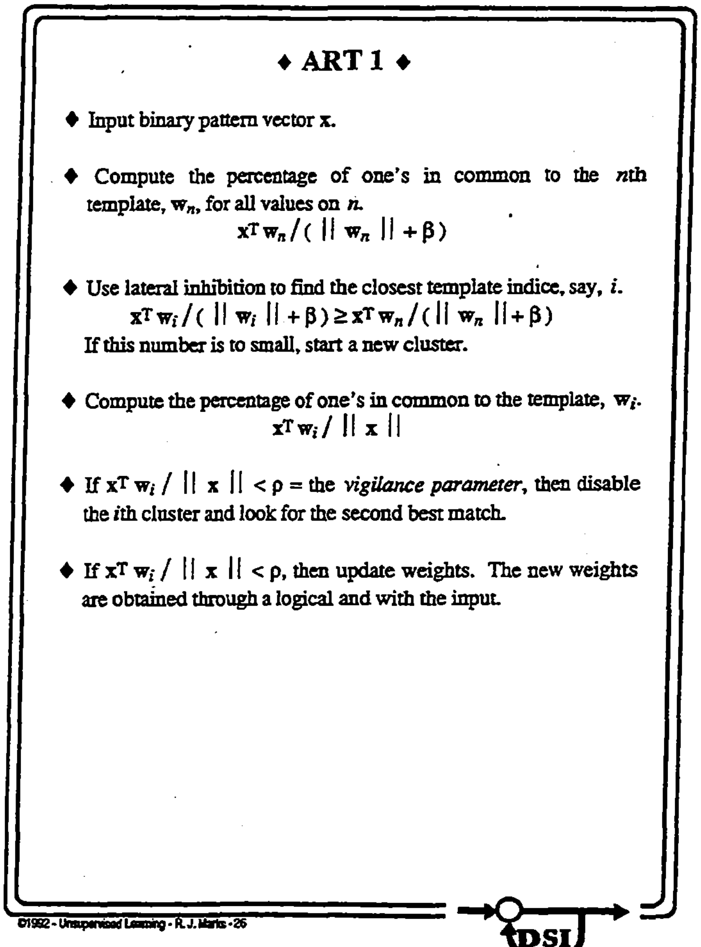
♦ Do We Update Or Start a New Category? ♦

4. The distance d is compared to a threshold parameter, t.
   - If $d < t$, the vector **x** is used to update the interconnects to y_n.
   - If $d > t$, the classification y_{N+1} is initiated.

©1992 - Unsupervised Learning - R. J. Marks - 25

Figure 25

If this distance is small enough, the weights for the template are updated. If the threshold parameter is exceeded, a new category is created. If the threshold parameter is small, there will be a greater number of categories.



♦ ART 1 ♦

- ♦ Input binary pattern vector **x**.
- ♦ Compute the percentage of one's in common to the *n*th template, $\mathbf{w}_n$, for all values on *n*.

  $$\mathbf{x}^T\mathbf{w}_n/(\;||\;\mathbf{w}_n\;||+\beta)$$

- ♦ Use lateral inhibition to find the closest template indice, say, *i*.

  $$\mathbf{x}^T\mathbf{w}_i/(\;||\;\mathbf{w}_i\;||+\beta)\geq\mathbf{x}^T\mathbf{w}_n/(\;||\;\mathbf{w}_n\;||+\beta)$$

  If this number is to small, start a new cluster.
- ♦ Compute the percentage of one's in common to the template, $\mathbf{w}_i$.

  $$\mathbf{x}^T\mathbf{w}_i/\;||\;\mathbf{x}\;||$$

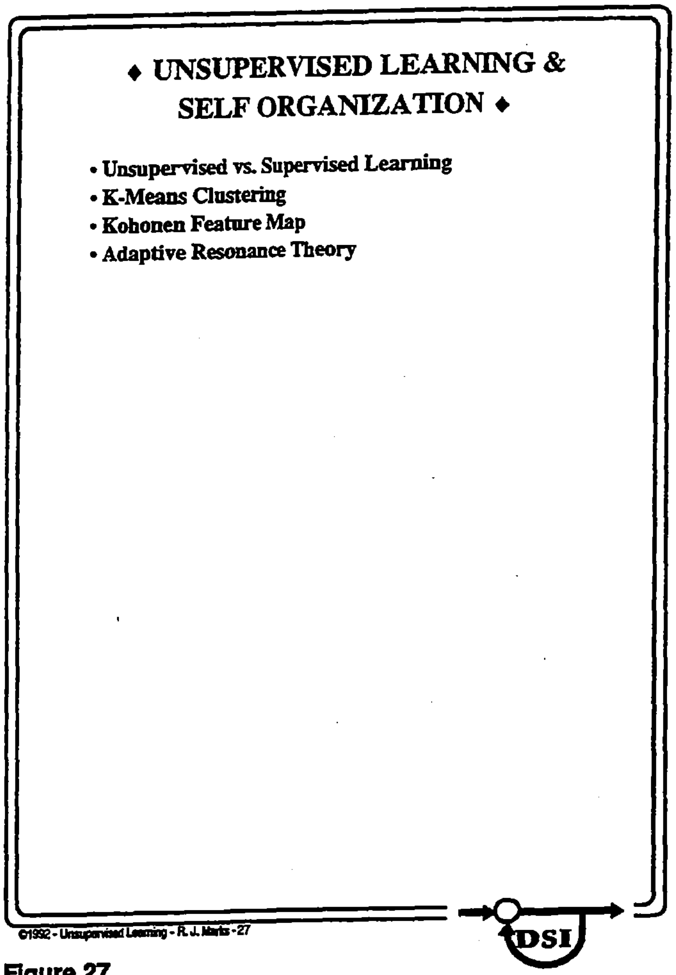
- ♦ If $\mathbf{x}^T\mathbf{w}_i/\;||\;\mathbf{x}\;||<\rho=$ the *vigilance parameter*, then disable the *i*th cluster and look for the second best match.
- ♦ If $\mathbf{x}^T\mathbf{w}_i/\;||\;\mathbf{x}\;||<\rho$, then update weights. The new weights are obtained through a logical and with the input.

©1992 - Unsupervised Learning - R. J. Marks - 26

DSI

Figure 26

ART 1 is much more complicated than the model we presented.



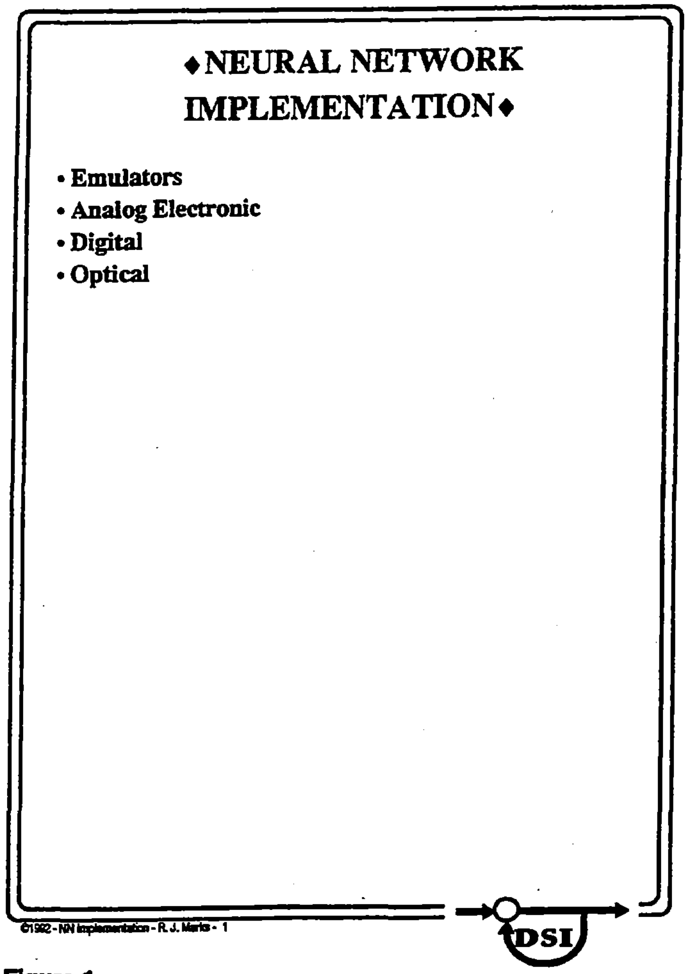
♦ UNSUPERVISED LEARNING & SELF ORGANIZATION ♦

- Unsupervised vs. Supervised Learning
- K-Means Clustering
- Kohonen Feature Map
- Adaptive Resonance Theory

Figure 27

Notes Notes Notes

Neural Network Implementation



♦NEURAL NETWORK IMPLEMENTATION♦

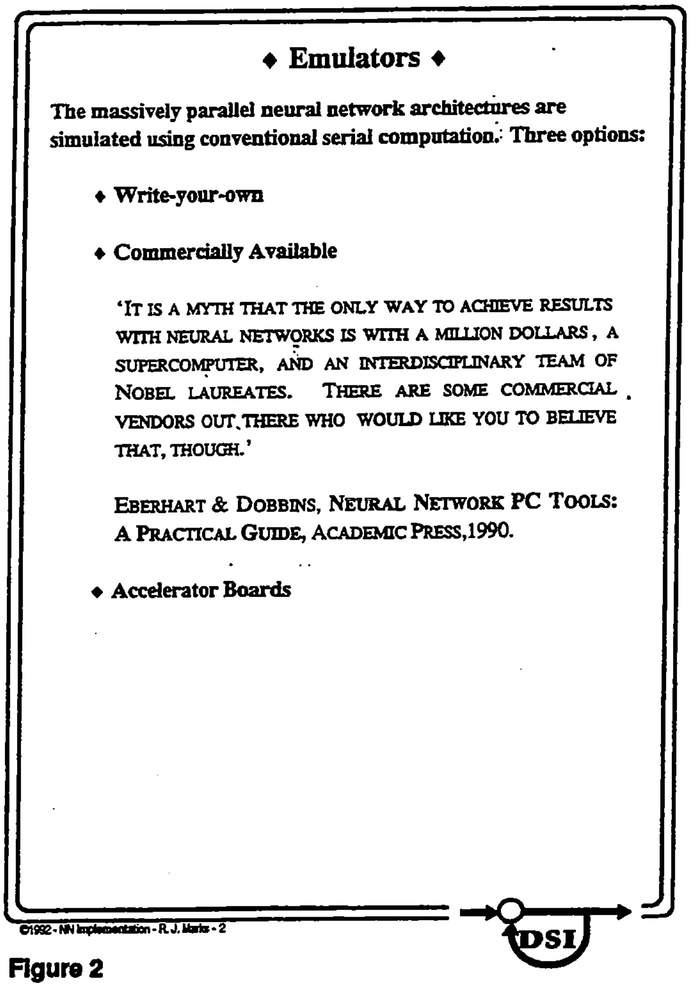
- Emulators
- Analog Electronic
- Digital
- Optical

©1992 - NN Implementation - R. J. Marks - 1

Figure 1

A set of relatively recent tutorials on artificial neural network implementation, edited by Robert J. Marks II, were published in *IEEE Circuits & Devices Magazine*.

- H.P. Graf & L.D. Jackal, "Analog electronic neural network circuits," *IEEE Circuits & Devices Magazine*, vol.5, pp.44-49 (July, 1989).
- N.H. Farhat, "Optoelectronic neural networks & learning machines," *IEEE Circuits & Devices Magazine*, vol.5, pp.32-41 (September, 1989).
- L.E. Atlas and Y. Suzuki, "Digital systems for artificial neural networks," *IEEE Circuits & Devices Magazine*, vol. 5, pp.20-24 (1989).



♦ Emulators ♦

The massively parallel neural network architectures are simulated using conventional serial computation. Three options:

- ♦ **Write-your-own**
- ♦ **Commercially Available**

> 'IT IS A MYTH THAT THE ONLY WAY TO ACHIEVE RESULTS WITH NEURAL NETWORKS IS WITH A MILLION DOLLARS, A SUPERCOMPUTER, AND AN INTERDISCIPLINARY TEAM OF NOBEL LAUREATES. THERE ARE SOME COMMERCIAL VENDORS OUT THERE WHO WOULD LIKE YOU TO BELIEVE THAT, THOUGH.'
>
> EBERHART & DOBBINS, NEURAL NETWORK PC TOOLS: A PRACTICAL GUIDE, ACADEMIC PRESS,1990.

- ♦ **Accelerator Boards**

©1992 - NN Implementation - R. J. Marks - 2

Figure 2

Most neural network algorithms can be described with a few lines of code. Commercial software is more user friendly and can display quite impressive graphics. Some books, such as that by Eberhart & Robbins, have accompanying share ware.

• An Analog Neuron •

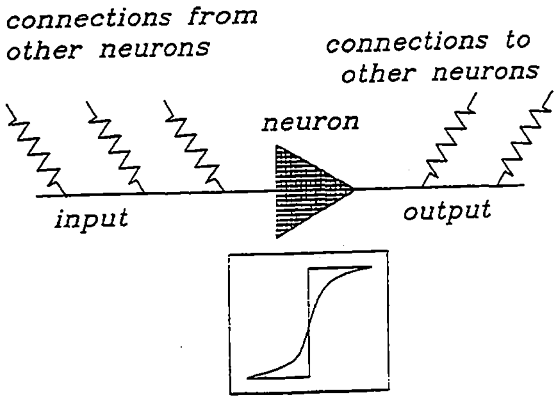
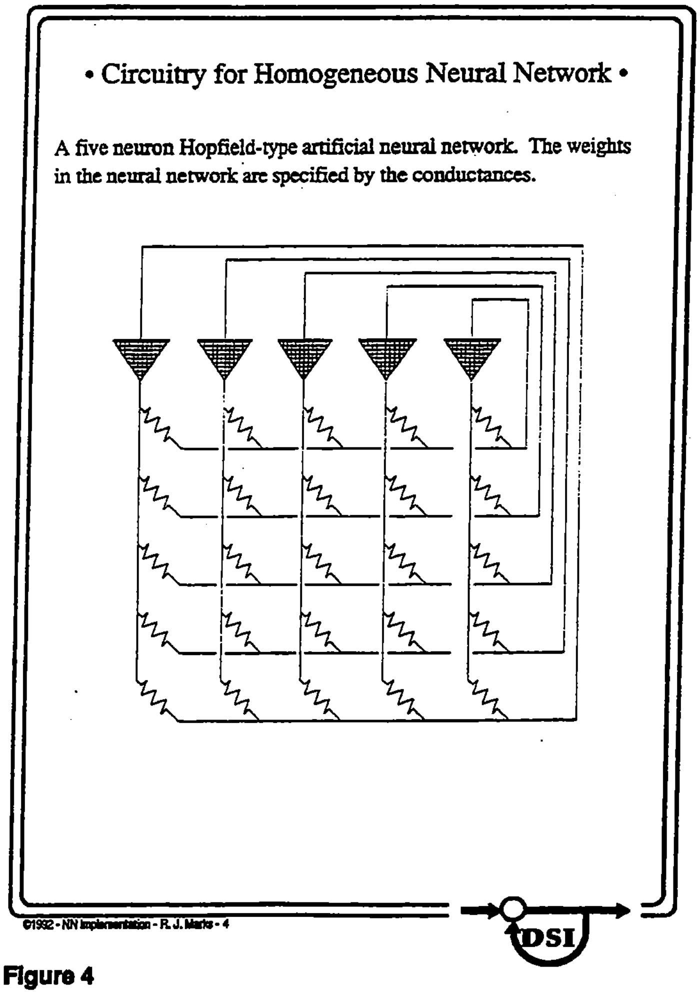


Figure 3

The weights between neurons are the conductances shown. Summing the inputs is performed using KCL. The neuron performs a nonlinear operation. Both a soft sigmoid and a hard nonlinearity are shown. The current output of the neuron is then connected to other neurons through the conductances shown.

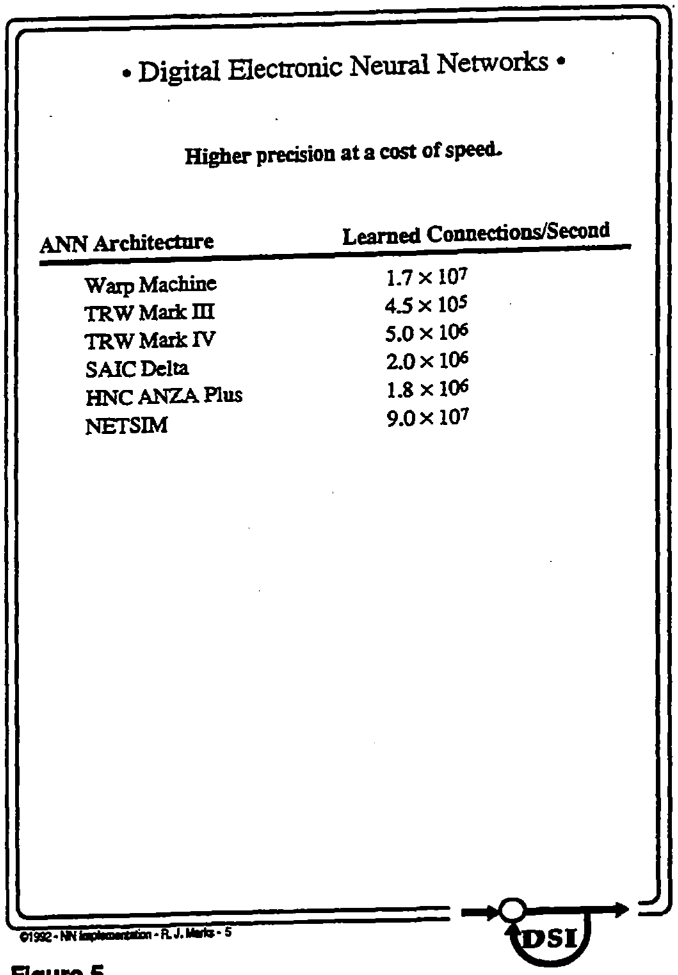


• Circuitry for Homogeneous Neural Network •

A five neuron Hopfield-type artificial neural network. The weights in the neural network are specified by the conductances.

Figure 4

For a five neuron Hopfield neural network, there are 25 interconnects shown here as conductances. Circuitry of this type has been used to solve traveling salesman and associative memory problems. The computational of analog neural networks lies typically between 10^9 and 10^{11} interconnections. Board emulators have been built with interconnect rates of up to 10^7 per second. A density of 4 resistors per square µm has been achieved. This is 4 X 10^8 resistors per square cm.



• Digital Electronic Neural Networks •

Higher precision at a cost of speed.

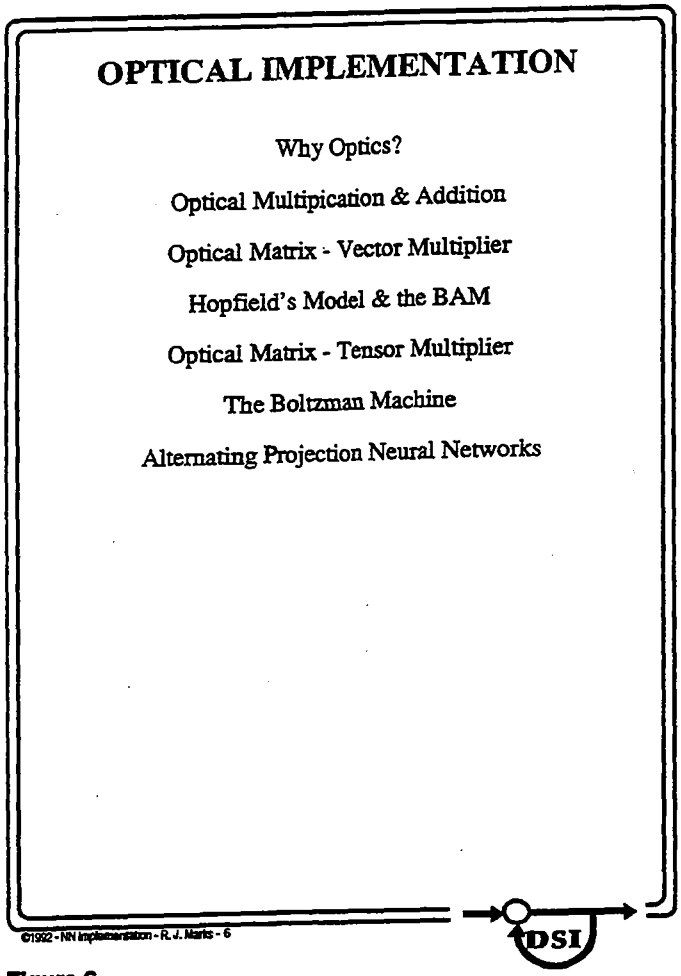
ANN Architecture	Learned Connections/Second
Warp Machine	1.7×10^7
TRW Mark III	4.5×10^5
TRW Mark IV	5.0×10^6
SAIC Delta	2.0×10^6
HNC ANZA Plus	1.8×10^6
NETSIM	9.0×10^7

Figure 5

These figures, from Atlas & Suzuki, illustrate the relative speeds of some digital systems. Their precision is required for some applications.

Notes Notes Notes

Notes Notes Notes



OPTICAL IMPLEMENTATION

Why Optics?

Optical Multipication & Addition

Optical Matrix - Vector Multiplier

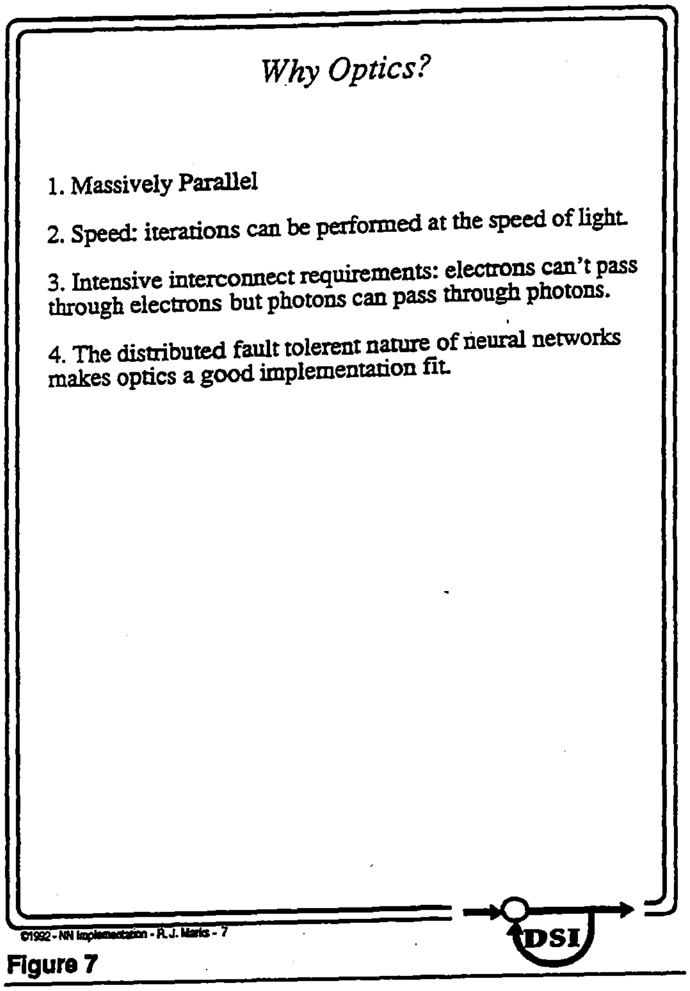
Hopfield's Model & the BAM

Optical Matrix - Tensor Multiplier

The Boltzman Machine

Alternating Projection Neural Networks

Figure 6



Why Optics?

1. Massively Parallel

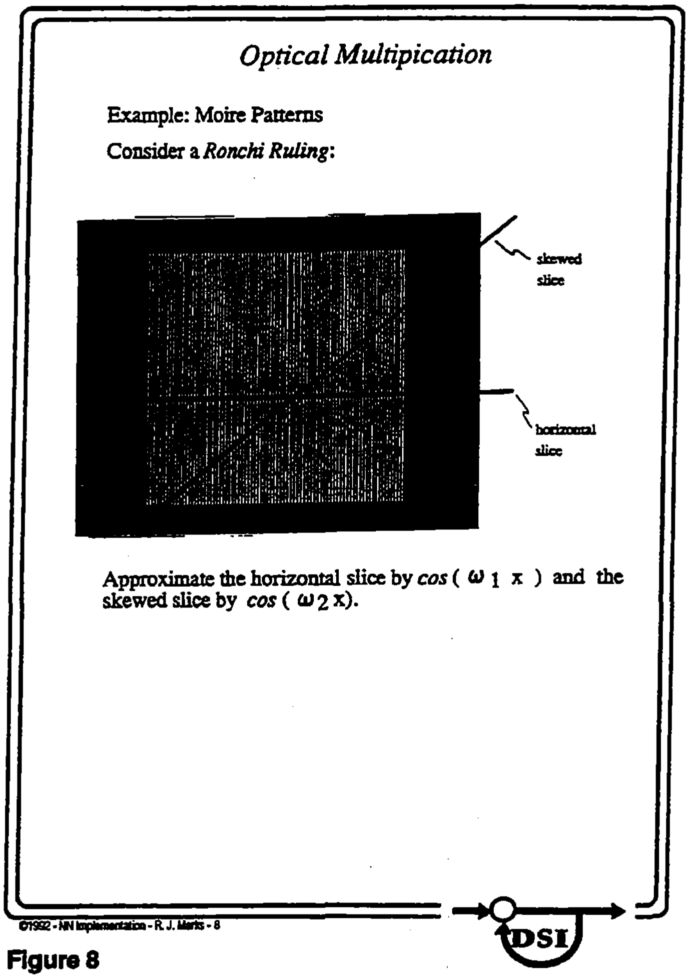
2. Speed: iterations can be performed at the speed of light.

3. Intensive interconnect requirements: electrons can't pass through electrons but photons can pass through photons.

4. The distributed fault tolerent nature of neural networks makes optics a good implementation fit.

Figure 7

Notes Notes Notes



Optical Multipication

Example: Moire Patterns

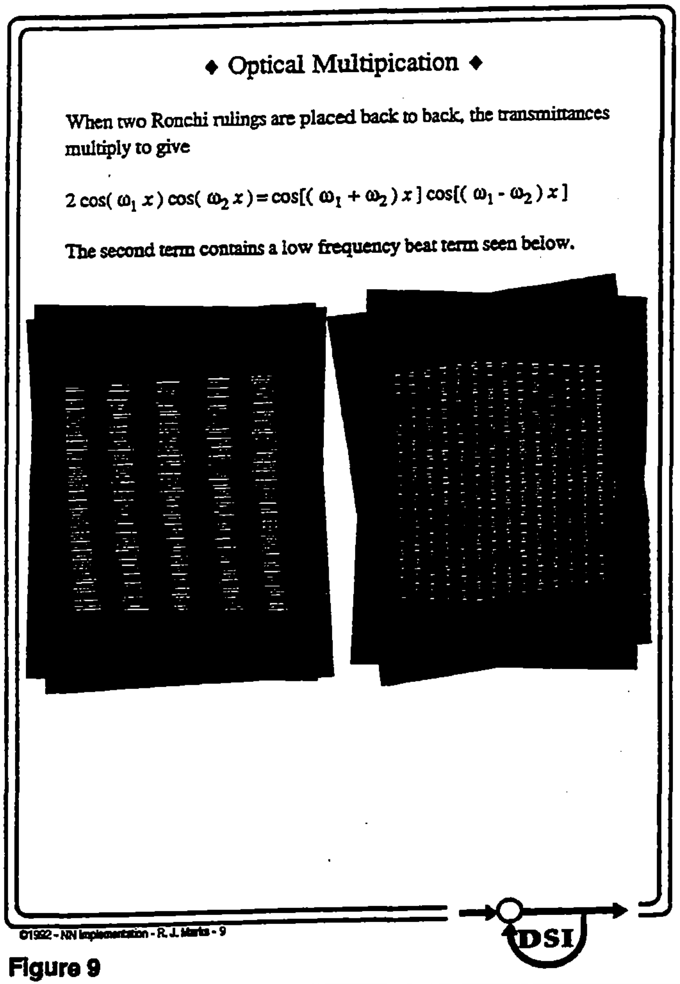
Consider a *Ronchi Ruling*:

Approximate the horizontal slice by *cos* (ω_1 x) and the skewed slice by *cos* (ω_2 x).

©1992 - NN Implementation - R. J. Marks - 8

Figure 8

Analog multiplication can be performed in parallel in the time it takes light to travel through a transparency. This will be illustrated with a Moire pattern. In the Ronchi ruling shown, the fundamental frequency of the two slices is different.



◆ Optical Multipication ◆

When two Ronchi rulings are placed back to back, the transmittances multiply to give

$$2\cos(\omega_1 x)\cos(\omega_2 x) = \cos[(\omega_1 + \omega_2)x]\cos[(\omega_1 - \omega_2)x]$$

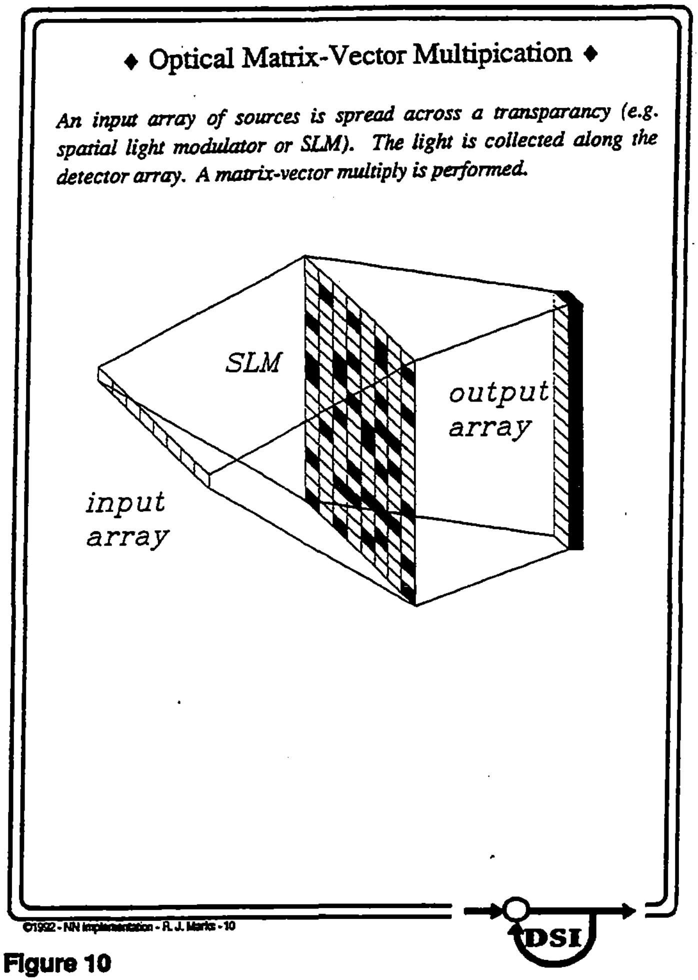
The second term contains a low frequency beat term seen below.

©1992 - NN Implementation - R. J. Marks - 9

Figure 9

Notes Notes Notes

The beat frequencies observed in the Moire patterns illustrate that optical multiplication has been performed. The massively parallel operation is performed in the time that it takes light to travel through the transparencies.



♦ Optical Matrix-Vector Multipication ♦

An input array of sources is spread across a transparancy (e.g. spatial light modulator or SLM). The light is collected along the detector array. A matrix-vector multiply is performed.

Figure 10

This simple optical processor can perform an optical matrix-vector multiply in the time it takes light to travel from the source array to the detector array. (The astigmatic focusing optics are not shown).

♦ Optical Implementation of the Hopfield Model ♦

- Neural feedback is performed electronically.
- The interconnects can be rounded to ±1. *The net still works!*

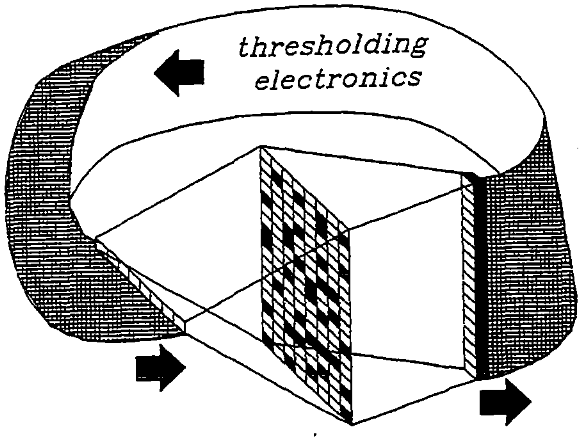
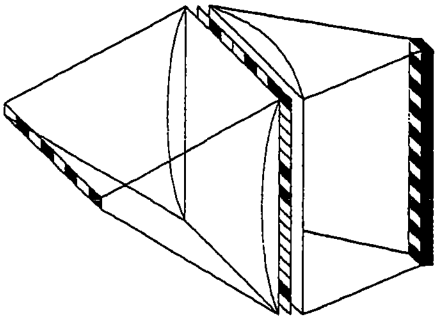


Figure 11

We are here letting optics do what it does best (parallel operations) and the electronics doe what it does best (nonlinear thresholding). In order for optics to perform a nonlinear operation, it must interact with matter.

◆ Optical Implementation of the BAM ◆

Here, detectors and sources are interlaced at both the front and back focal planes.

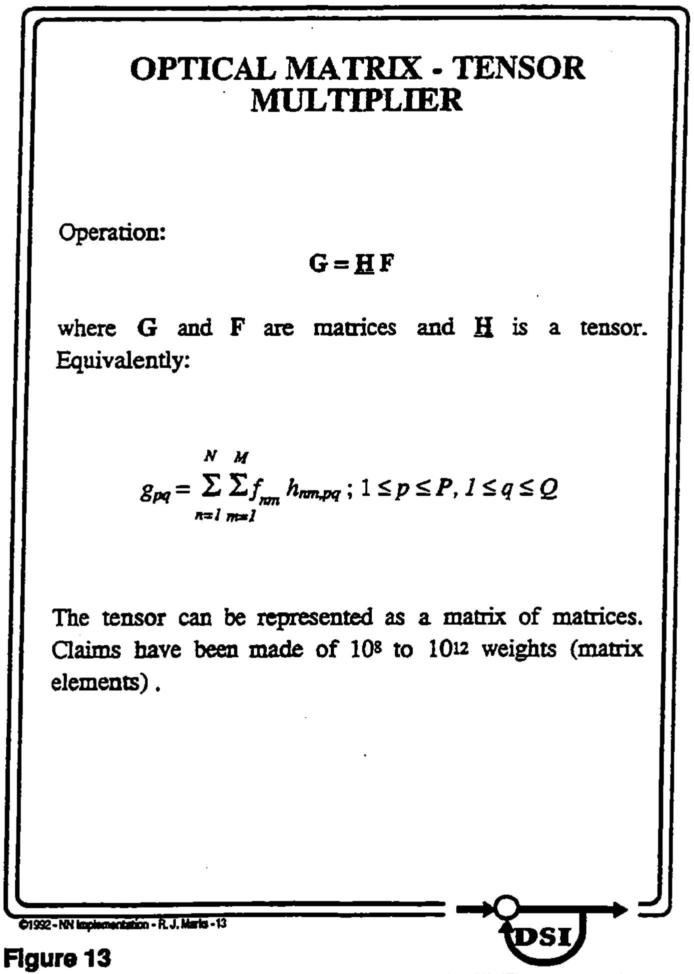


©1992 - NN Implementation - R. J. Marks - 12

DSI

Figure 12

Light is introduced from the source at the left. The detectors on the light receive the first iteration. Adjacent to the detectors are sources which illuminate as a nonlinear function of the incident illumination. The sources produce light from right to left and the process is repeated at the left hand combination detector-source array. Iteration is performed until convergence.



OPTICAL MATRIX - TENSOR MULTIPLIER

Operation:

$$\mathbf{G} = \underline{\mathbf{H}}\,\mathbf{F}$$

where **G** and **F** are matrices and $\underline{\mathbf{H}}$ is a tensor. Equivalently:

$$g_{pq} = \sum_{n=1}^{N} \sum_{m=1}^{M} f_{nm}\, h_{nm,pq}\,;\ 1 \le p \le P,\ 1 \le q \le Q$$

The tensor can be represented as a matrix of matrices. Claims have been made of 10^8 to 10^{12} weights (matrix elements) .

©1992 - NN Implementation - R. J. Marks - 13

DSI

Figure 13

The optical matrix-vector multiplier can be extended to a matrix-tensor multiplier. This allows operations on matrices.

Notes Notes Notes

Notes Notes Notes

Example:

$$g_{pq} = \sum \sum f_{nm} h_{nm,pq} ; 1 \le p \le P, 1 \le q \le Q$$

$$N=6, M=4, P=3, Q=4:$$

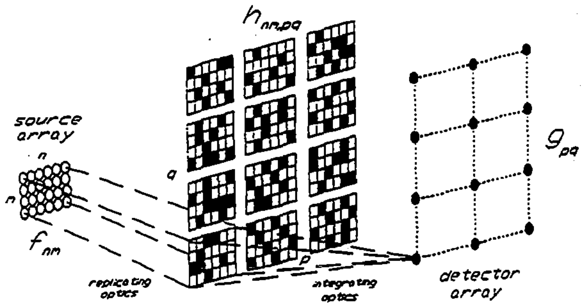
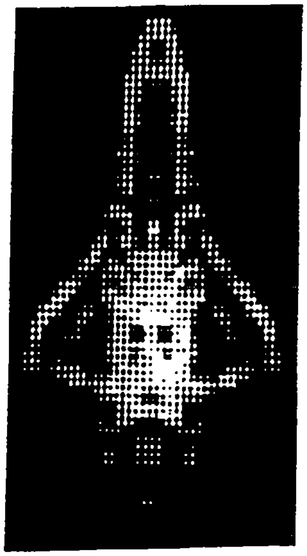


Figure 14

A tensor can be expressed as a matrix of matrices. Optics can straightforwardly be fabricated to perform the shadow replication optical processor shown here. Other methods have been proposed for the operation.

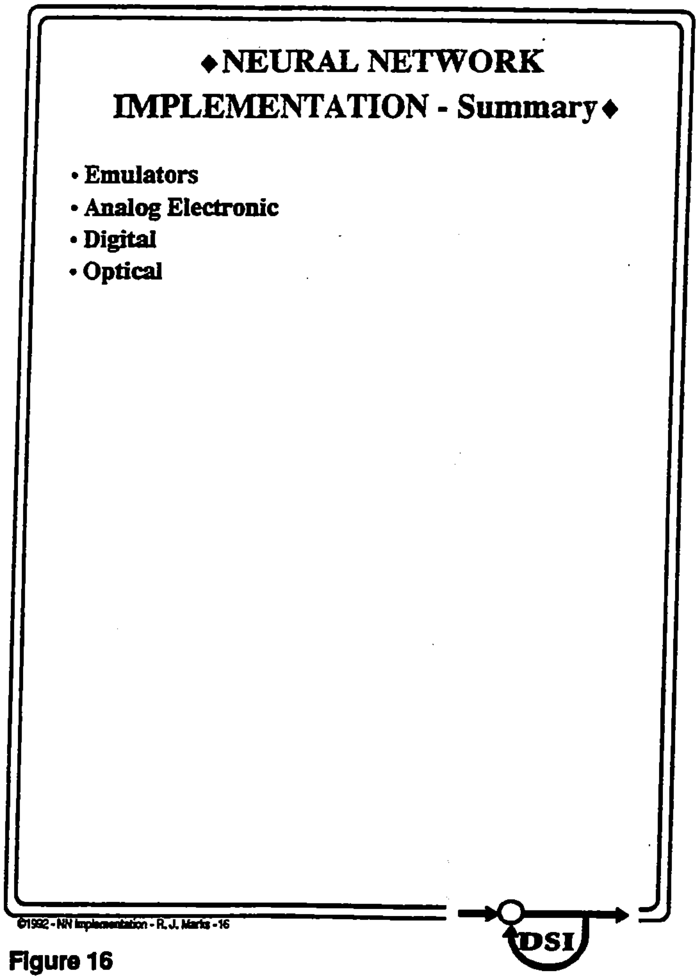
An Output Display:



©1992 - NN Implementation - R. J. Marks - 15

Figure 15

A two-dimensional array of the type shown can be processed by the matrix-tensor multiplier.



◆NEURAL NETWORK IMPLEMENTATION - Summary◆

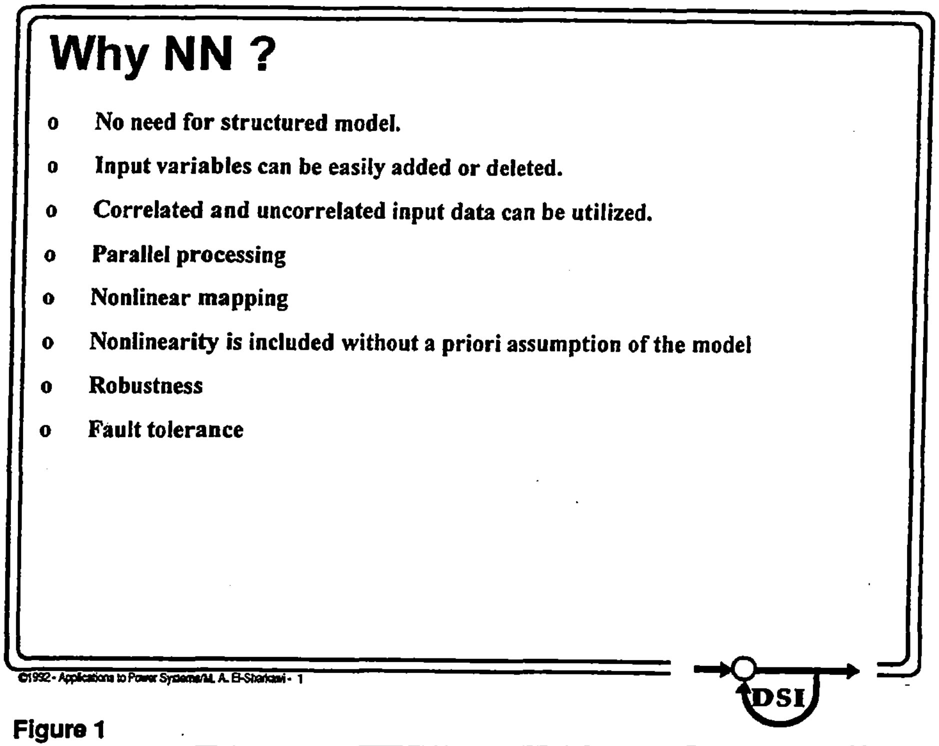
- Emulators
- Analog Electronic
- Digital
- Optical

©1992 - NN Implementation - R. J. Marks - 16

DSI

Figure 16

Application to Power Systems



Why NN ?

- No need for structured model.
- Input variables can be easily added or deleted.
- Correlated and uncorrelated input data can be utilized.
- Parallel processing
- Nonlinear mapping
- Nonlinearity is included without a priori assumption of the model
- Robustness
- Fault tolerance

Figure 1

Notes Notes Notes

Notes Notes Notes

Artificial Neural Network Classifiers (NN's)

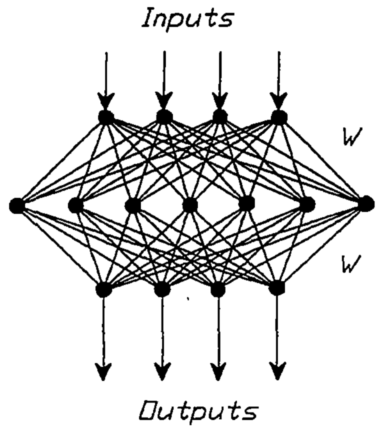


Figure 2

Neuron Structure

$$net_j = \sum_i (W_{ji} \; O_i)$$

$$O_j = f(net_j) = \frac{1}{1 + \exp \left[\frac{-(net_j + \alpha_j)}{\alpha_o} \right]}$$

$$O_j = f(net_j) = \frac{1}{1 + \exp \left[\frac{-(net_j + \alpha_j)}{\alpha_o} \right]}$$

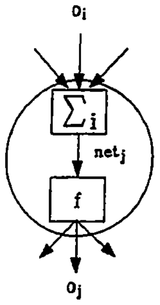
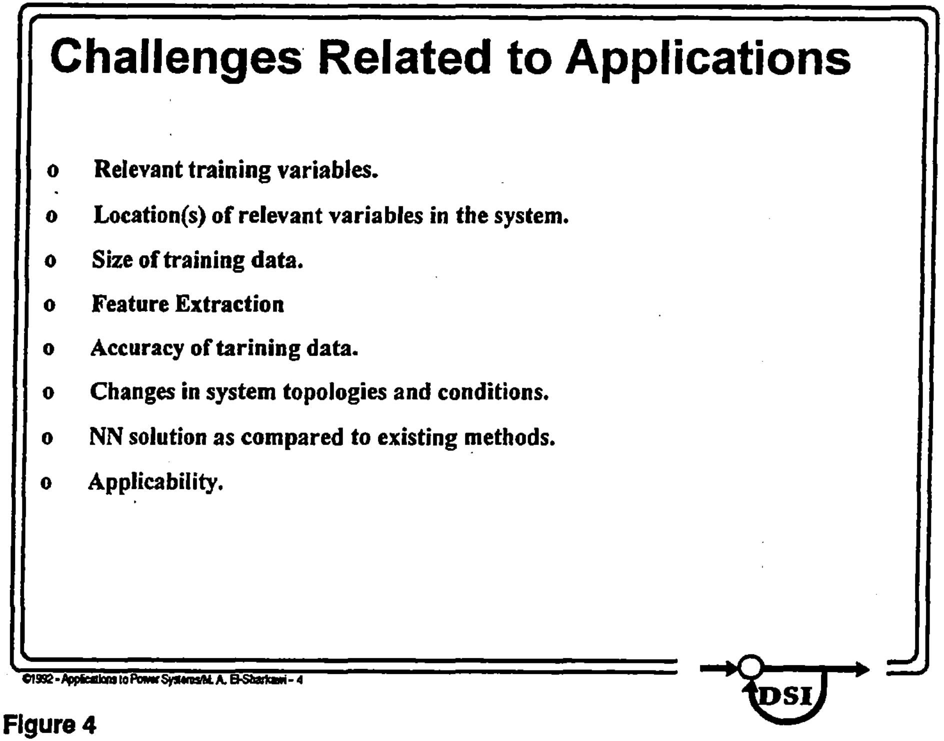


Figure 3



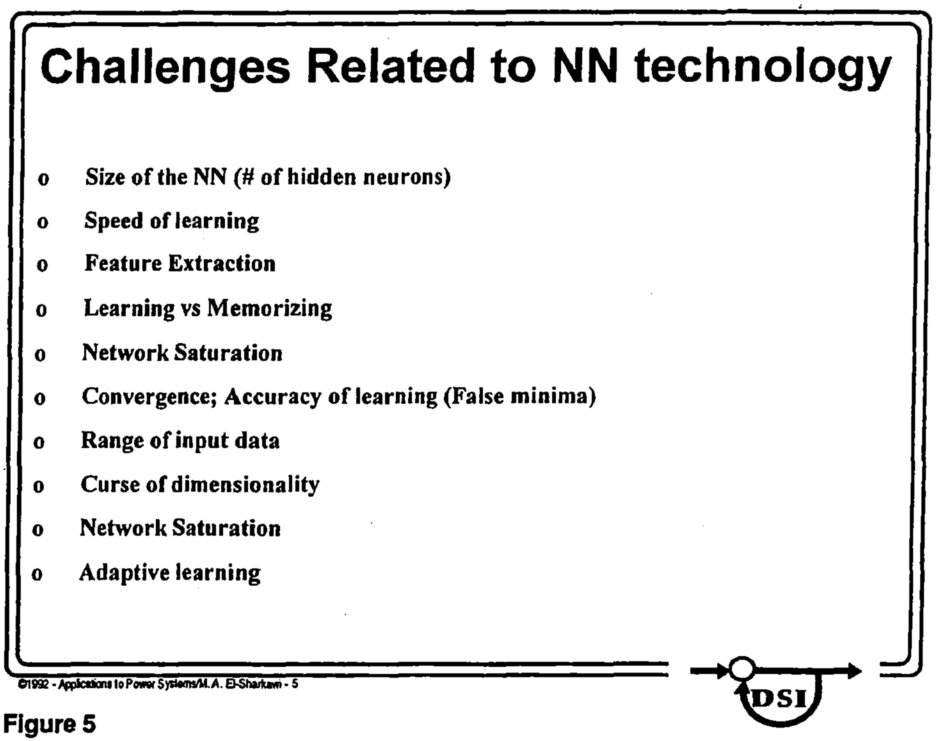
Challenges Related to Applications

- Relevant training variables.
- Location(s) of relevant variables in the system.
- Size of training data.
- Feature Extraction
- Accuracy of tarining data.
- Changes in system topologies and conditions.
- NN solution as compared to existing methods.
- Applicability.

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 4

DSI

Figure 4



Challenges Related to NN technology

- Size of the NN (# of hidden neurons)
- Speed of learning
- Feature Extraction
- Learning vs Memorizing
- Network Saturation
- Convergence; Accuracy of learning (False minima)
- Range of input data
- Curse of dimensionality
- Network Saturation
- Adaptive learning

©1992 - Applications to Power Systems/M.A. El-Sharkawi - 5

Figure 5

Notes Notes Notes

Notes Notes Notes

Memorization

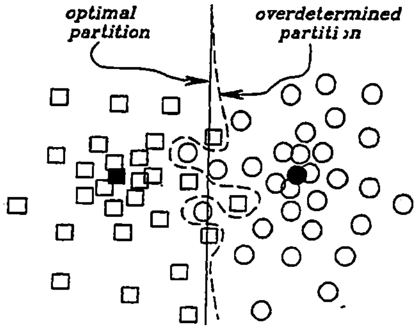
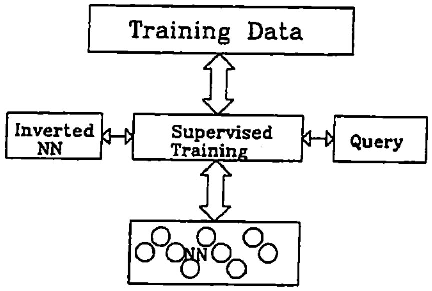


Figure 6

Notes Notes Notes

Integrated Neural Network System



Training Data

Inverted NN

Supervised Training

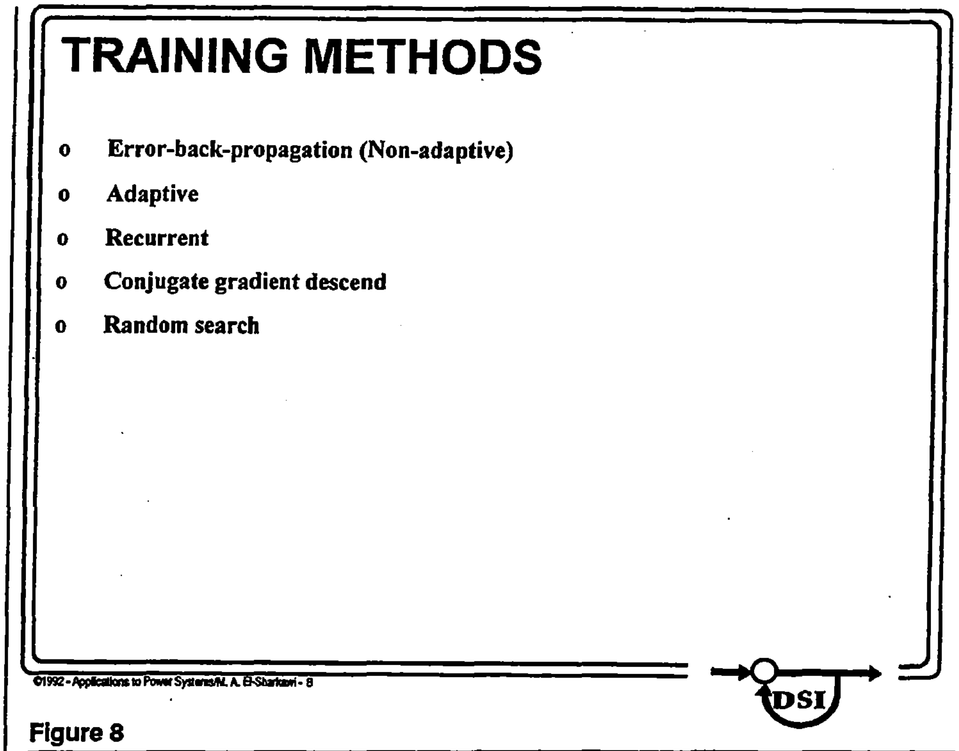
Query

NN

DSI

Figure 7

Notes Notes Notes

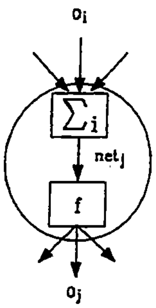


TRAINING METHODS

- Error-back-propagation (Non-adaptive)
- Adaptive
- Recurrent
- Conjugate gradient descend
- Random search

Figure 8

Error-back-propagation



$$E=\frac{1}{2}\sum_{k}(T_k-O_k)^2$$

$$\Delta W_{kj}=-\gamma\ \delta E/\delta W_{kj}$$

<u>Hidden to Output</u>

$$\Delta W_{kj}=\gamma\ \varepsilon_k O_j$$

$$\varepsilon_k=(O_k-T_k)\,O_k\,(1-O_k)$$

<u>Input to Hidden</u>

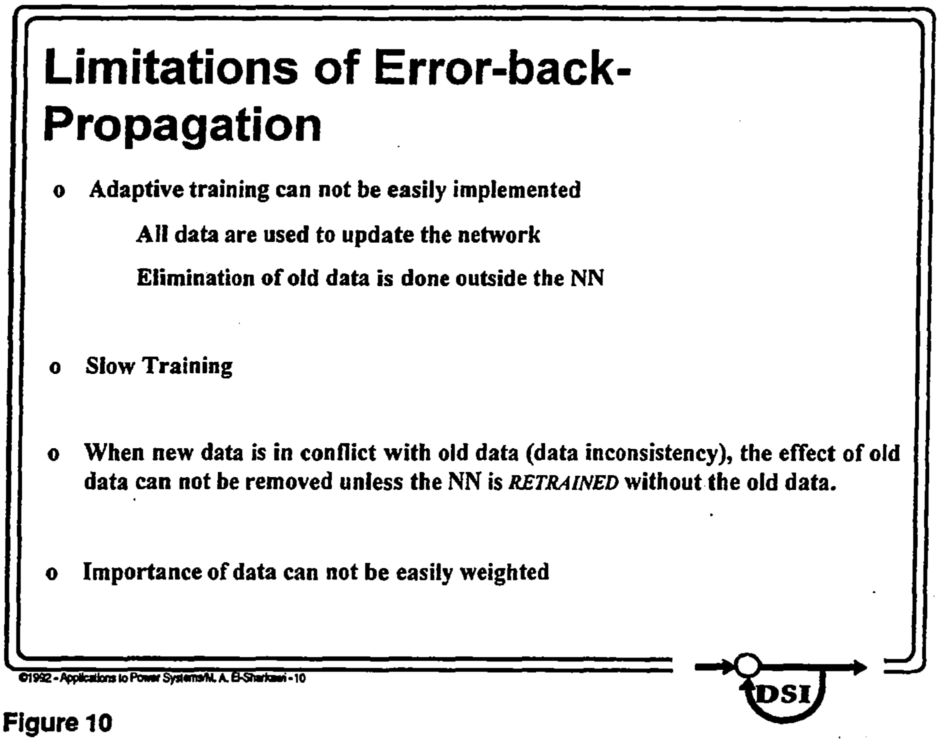
$$\Delta W_{ji}=\gamma\ \varepsilon_j\ O_i$$

$$\varepsilon_j=\ O_j\,(1-O_j)\,\Sigma_k\,(\varepsilon_k\ W_{kj})$$

Figure 9

Notes Notes Notes

Notes Notes Notes



Limitations of Error-back-Propagation

- o Adaptive training can not be easily implemented

  All data are used to update the network

  Elimination of old data is done outside the NN

- o Slow Training

- o When new data is in conflict with old data (data inconsistency), the effect of old data can not be removed unless the NN is *RETRAINED* without the old data.

- o Importance of data can not be easily weighted

DSI

Figure 10

Adaptively Trained NN

$$e(i) = \frac{1}{2} [\, t(i) - o(i) \,]^2$$

$$E(N) = \frac{1}{2} \Sigma_N [t(N) - o(N)]^2$$

$$E(N+1) = E(N) + e(N+1)$$

Objectives

1) $W(N+1) = W(N) + \Delta\, W(N+1)$ Input-to-hidden

$V(N+1) = V(N) + \Delta\, V(N+1)$ Hidden-to-output

2) Adaptive condition $\Delta\, W(N+1) < \zeta$

$\Delta\, V(N+1) < \zeta$

Figure 11

Notes Notes Notes

Adaptively Trained NN Continue)

3) **Drifting condition**

If x(N+1) = x(i); i=1,2, ... ,or N

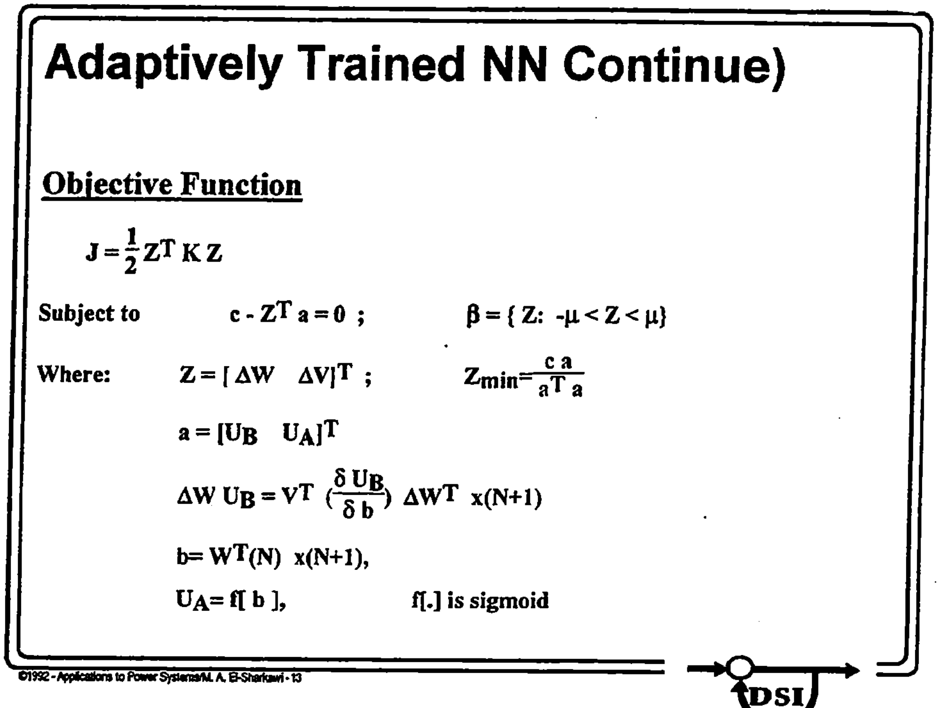
then

$\Delta W(N+1) = 0$

$\Delta V(N+1) = 0$

Figure 12

Notes Notes Notes



Adaptively Trained NN Continue)

Objective Function

$$J = \frac{1}{2} Z^T K Z$$

Subject to $c - Z^T a = 0$; $\beta = \{ Z: -\mu < Z < \mu \}$

Where: $Z = [\,\Delta W \quad \Delta V]^T$; $Z_{min} = \frac{c\,a}{a^T a}$

$a = [U_B \quad U_A]^T$

$\Delta W\, U_B = V^T \left(\frac{\delta U_B}{\delta b}\right) \Delta W^T \; x(N+1)$

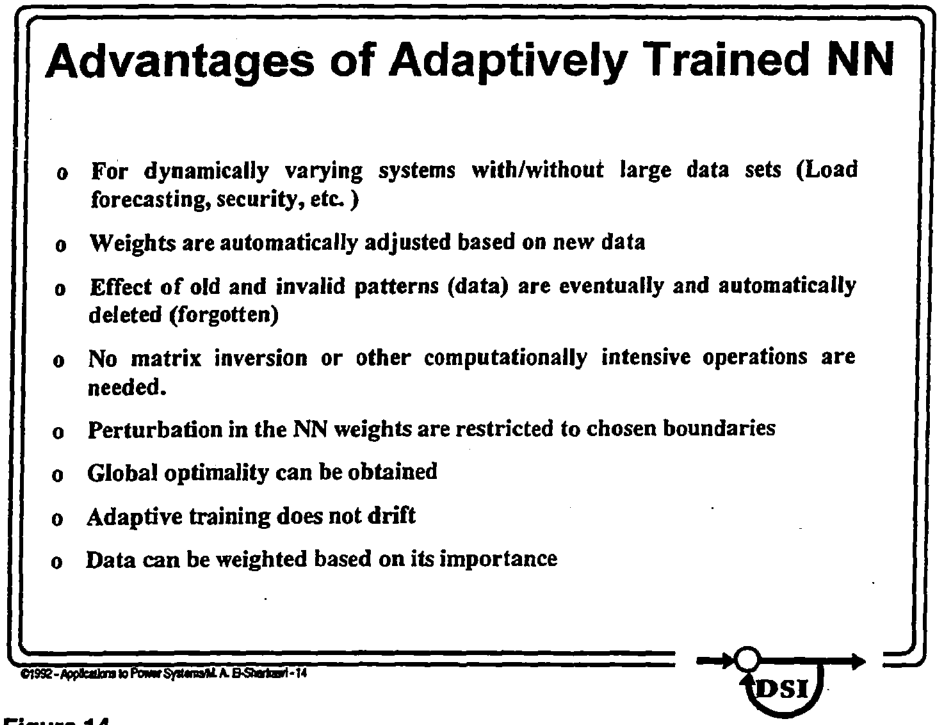
$b = W^T(N) \; x(N+1)$,

$U_A = f[\,b\,]$, f[.] is sigmoid

DSI

Figure 13

Notes Notes Notes



Advantages of Adaptively Trained NN

- For dynamically varying systems with/without large data sets (Load forecasting, security, etc.)
- Weights are automatically adjusted based on new data
- Effect of old and invalid patterns (data) are eventually and automatically deleted (forgotten)
- No matrix inversion or other computationally intensive operations are needed.
- Perturbation in the NN weights are restricted to chosen boundaries
- Global optimality can be obtained
- Adaptive training does not drift
- Data can be weighted based on its importance

Figure 14

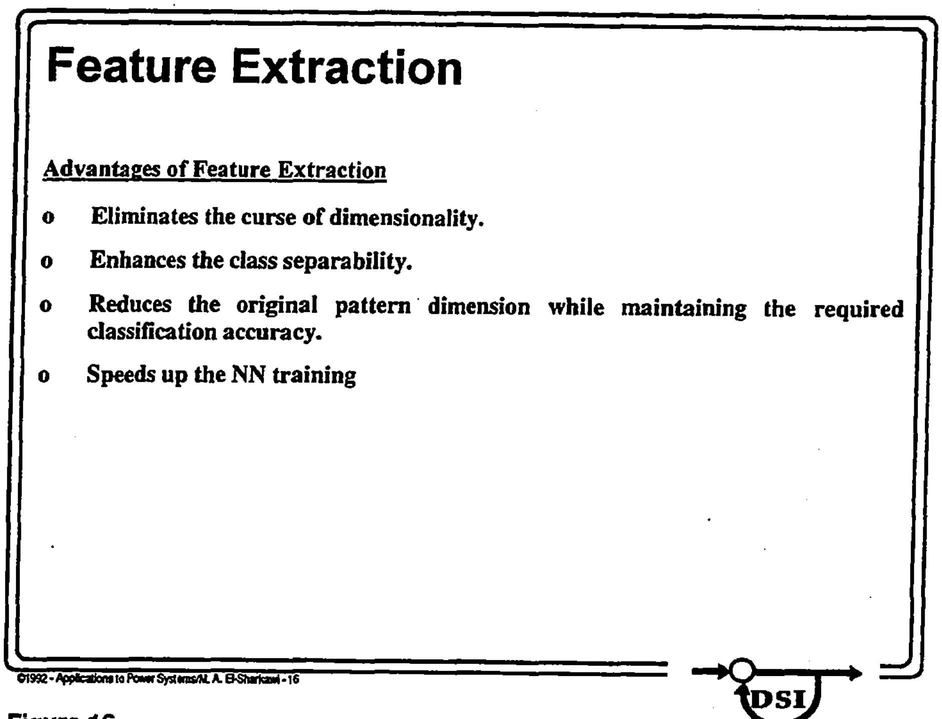
Feature Extraction

Why Feature Extraction

- In order to train a NN with reasonable accuracy, a sufficiently large data set spanning the operating space of the power system is required.
- Training the NN with such a large data is very time consuming process that may prohibit on-line applications.

Figure 15

Notes Notes Notes



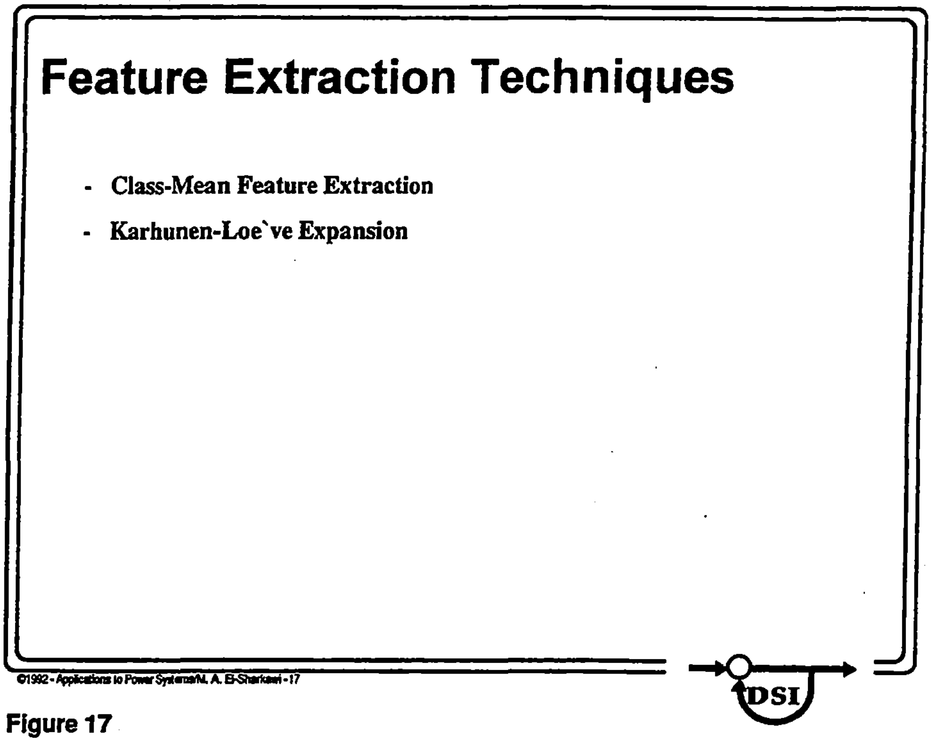
Feature Extraction

Advantages of Feature Extraction

- **Eliminates the curse of dimensionality.**
- **Enhances the class separability.**
- **Reduces the original pattern dimension while maintaining the required classification accuracy.**
- **Speeds up the NN training**

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 16

Figure 16



Feature Extraction Techniques

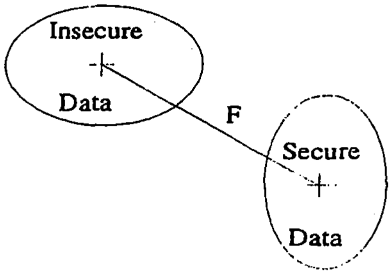
- Class-Mean Feature Extraction
- Karhunen-Loe`ve Expansion

Figure 17

Notes Notes Notes

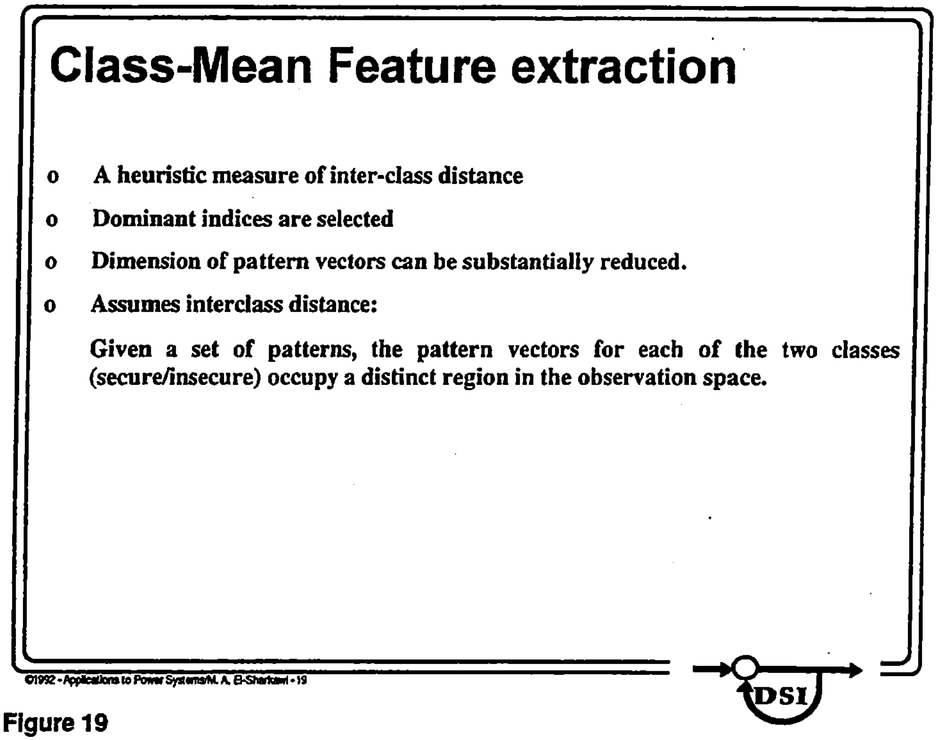
Class-Mean Feature extraction

Separation of a Single Variable



©1992 - Applications to Power Systems/M. A. El-Sharkawi - 18

Figure 18



Class-Mean Feature extraction

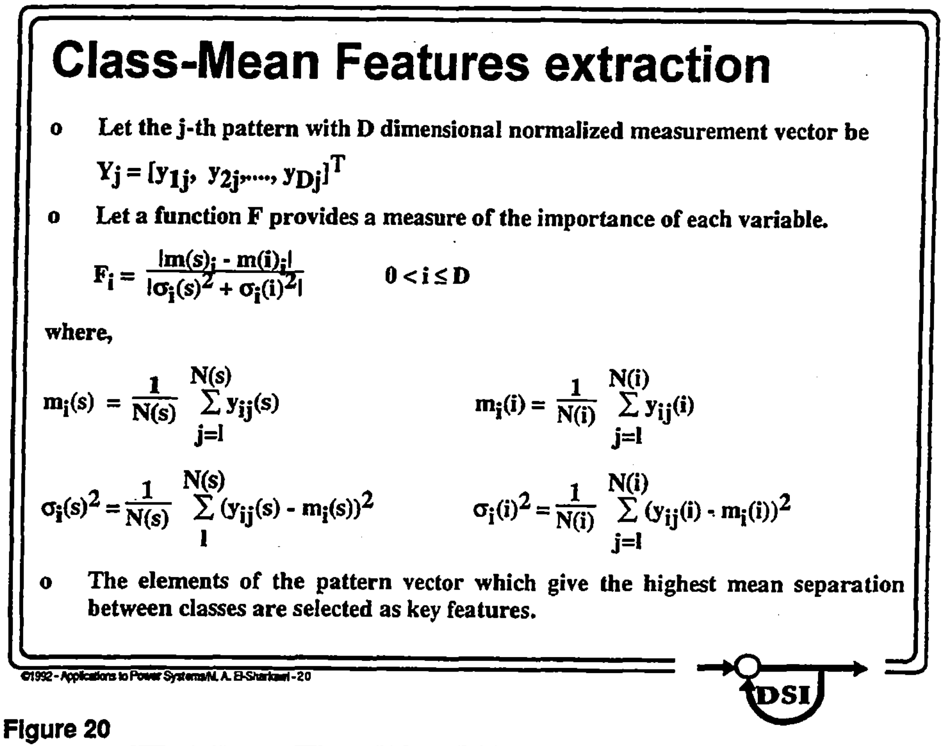
- A heuristic measure of inter-class distance
- Dominant indices are selected
- Dimension of pattern vectors can be substantially reduced.
- Assumes interclass distance:

  Given a set of patterns, the pattern vectors for each of the two classes (secure/insecure) occupy a distinct region in the observation space.

Figure 19

Notes Notes Notes

Notes Notes Notes



Class-Mean Features extraction

- Let the j-th pattern with D dimensional normalized measurement vector be $Y_j = [y_{1j}, y_{2j}, \dots, y_{Dj}]^T$
- Let a function F provides a measure of the importance of each variable.

$$F_i = \frac{|m(s)_i - m(i)_i|}{|\sigma_i(s)^2 + \sigma_i(i)^2|} \qquad 0 < i \leq D$$

where,

$$m_i(s) = \frac{1}{N(s)} \sum_{j=l}^{N(s)} y_{ij}(s) \qquad m_i(i) = \frac{1}{N(i)} \sum_{j=l}^{N(i)} y_{ij}(i)$$

$$\sigma_i(s)^2 = \frac{1}{N(s)} \sum_{l}^{N(s)} (y_{ij}(s) - m_i(s))^2 \qquad \sigma_i(i)^2 = \frac{1}{N(i)} \sum_{j=l}^{N(i)} (y_{ij}(i) - m_i(i))^2$$

- The elements of the pattern vector which give the highest mean separation between classes are selected as key features.

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 20

DSI

Figure 20

Each pattern vector should contain all possible variables affecting system security such as load powers, bus voltages, line flows, etc. With feature extraction, the dominant variables are selected. By this method, the dimension of the pattern vectors can be substantially reduced. For example, assume a pattern j with D dimensional normalized measurement vector,

$$Y_j = [y_{1j}, y_{2j}, \dots, y_{Dj}]^T$$

Assume that the dominant number of variables is d<<D. The security classification is then based on these d components. The heuristic notion of interclass distance is used to accomplish this task. Given a set of patterns with dimension D, it is reasonable to assume that the pattern vectors for each of the two classes (secure/insecure) occupy a distinct region in the observation space. The average pairwise distance between the patterns is a measure of class separability in the region with respect to the particular variable. The following function F provides a measure of the importance in each variable.

$$F_i = \frac{|m(s)_i - m(i)_i|}{|\sigma_i(s)^2 + \sigma_i(i)^2|}$$

$$0 < i \leq D$$

Where:

Notes Notes Notes

$$m_i(s) = \frac{1}{N(s)} \sum_{j=1}^{N(s)} y_{ij}(s)$$

$$\sigma_i(s)^2 = \frac{1}{N(s)} \sum_{1}^{N(s)} (y_{ij}(s) - m_i(s))^2$$

$$m_i(i) = \frac{1}{N(i)} \sum_{j=1}^{N(i)} y_{ij}(i)$$

$$\sigma_i(i)^2 = \frac{1}{N(i)} \sum_{j=1}^{N(i)} (y_{ij}(i) - m_i(i))^2$$

Subscript 's' stands for 'secure' while 'i' stands for 'insecure'. N(s) and N(i) indicate the number of secure and insecure patterns

m is the secure or insecure training sets

σ is the standard deviations.

Notes Notes Notes

Algorithm

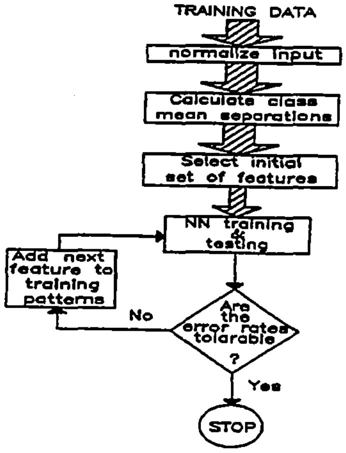

TRAINING DATA
normalize input
Calculate class mean separations
Select initial set of features
NN training & testing
Add next feature to training patterns
No
Are the error rates tolarable ?
Yes
STOP

Figure 21

The variables are ranked according to the following steps.

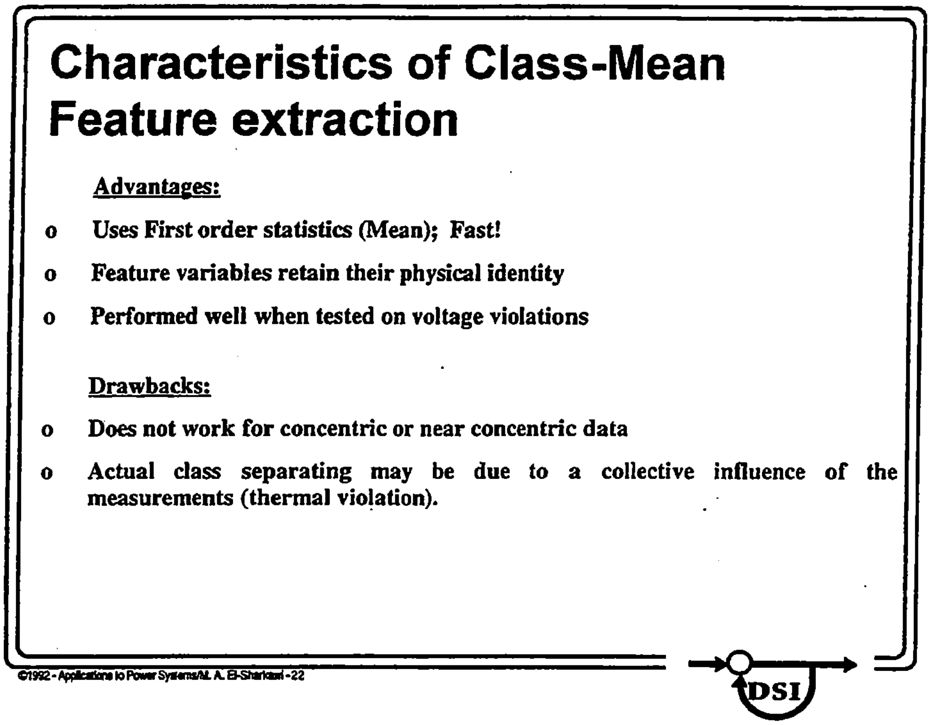
1. Calculate F_j ∀ $0 < j \le D$
2. Rank all F_j in a descending order
3. Go to the 1^{st} ranked variable.
4. Calculate correlation coefficients (CC) of all lower ranked variables with respect to the 1^{st} ranked variable. The CC is defined as,

$$CC_{ij} = \frac{E[(y_i - m_i)(y_j - m_j)]}{\sigma_i \sigma_j}$$

$$0 < j \le D$$

5. Eliminate all lower ranked variables which have a $|CC| > 0.9$
6. Go to the next highest ranked variable and go to step 4.

The process is repeated until all the variables are ranked or discarded. The resulting ordered list of variables are considered to be key features in training the NN classifier.



Characteristics of Class-Mean Feature extraction

Advantages:

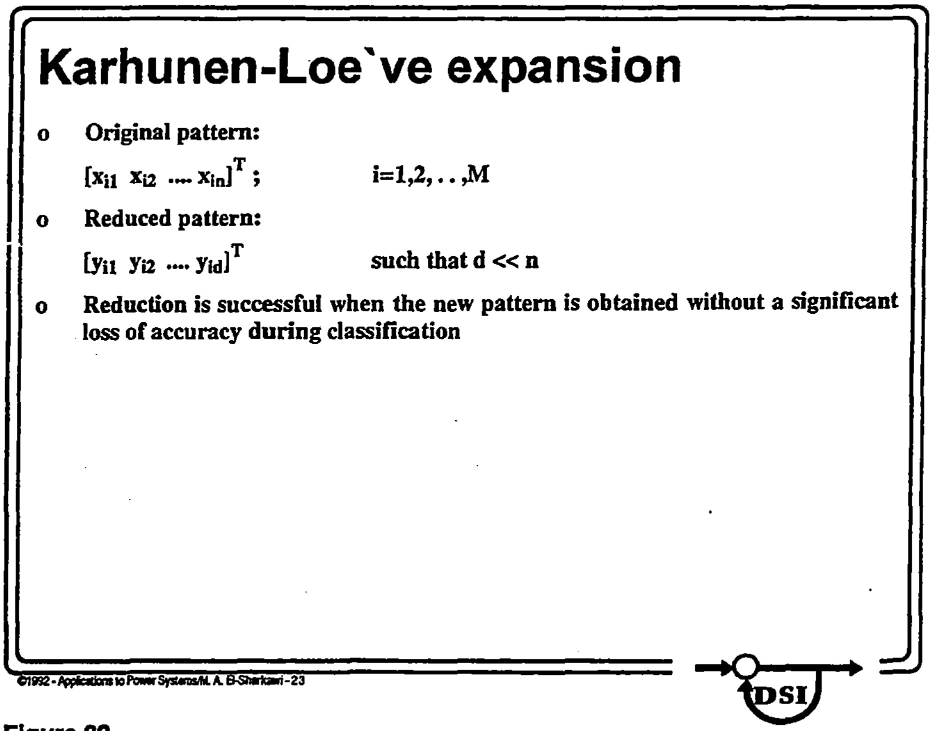
- o Uses First order statistics (Mean); Fast!
- o Feature variables retain their physical identity
- o Performed well when tested on voltage violations

Drawbacks:

- o Does not work for concentric or near concentric data
- o Actual class separating may be due to a collective influence of the measurements (thermal violation).

Figure 22

Notes Notes Notes



Karhunen-Loe`ve expansion

- o Original pattern:

  $[x_{i1}\ x_{i2}\ ....\ x_{in}]^T$; $i=1,2,..,M$

- o Reduced pattern:

  $[y_{i1}\ y_{i2}\ ....\ y_{id}]^T$ such that $d << n$

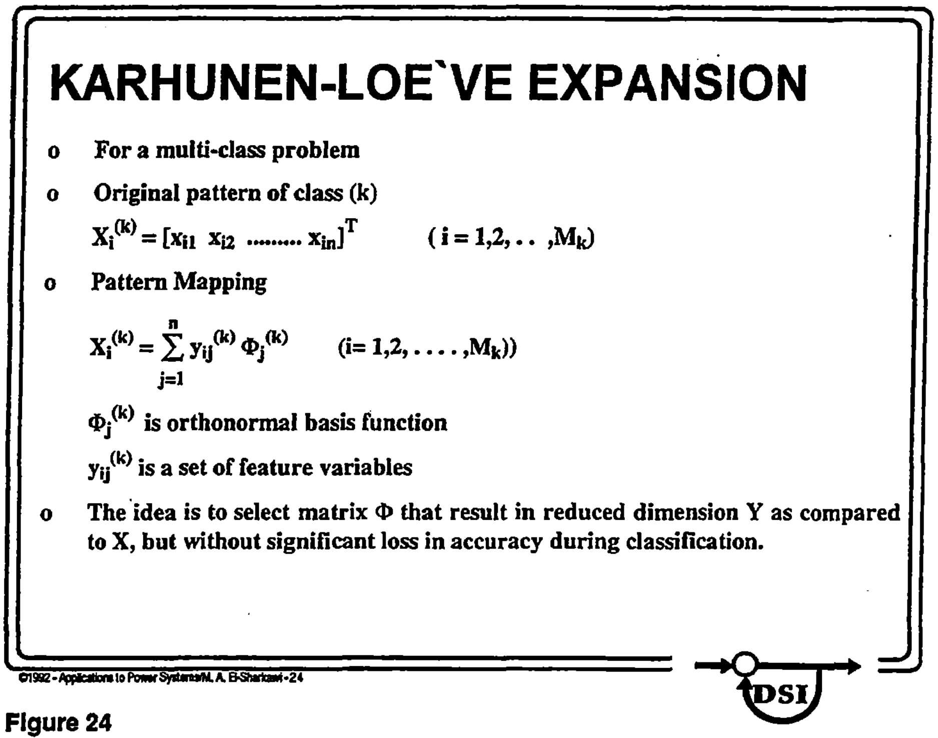
- o Reduction is successful when the new pattern is obtained without a significant loss of accuracy during classification

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 23

DSI

Figure 23

Notes Notes Notes



KARHUNEN-LOE`VE EXPANSION

- For a multi-class problem
- Original pattern of class (k)

  $X_i^{(k)} = [x_{i1}\ x_{i2}\ \ldots\ldots\ x_{in}]^T \qquad (i = 1,2,\ldots,M_k)$
- Pattern Mapping

  $$X_i^{(k)} = \sum_{j=1}^{n} y_{ij}^{(k)}\,\Phi_j^{(k)} \qquad (i= 1,2,\ldots,M_k))$$

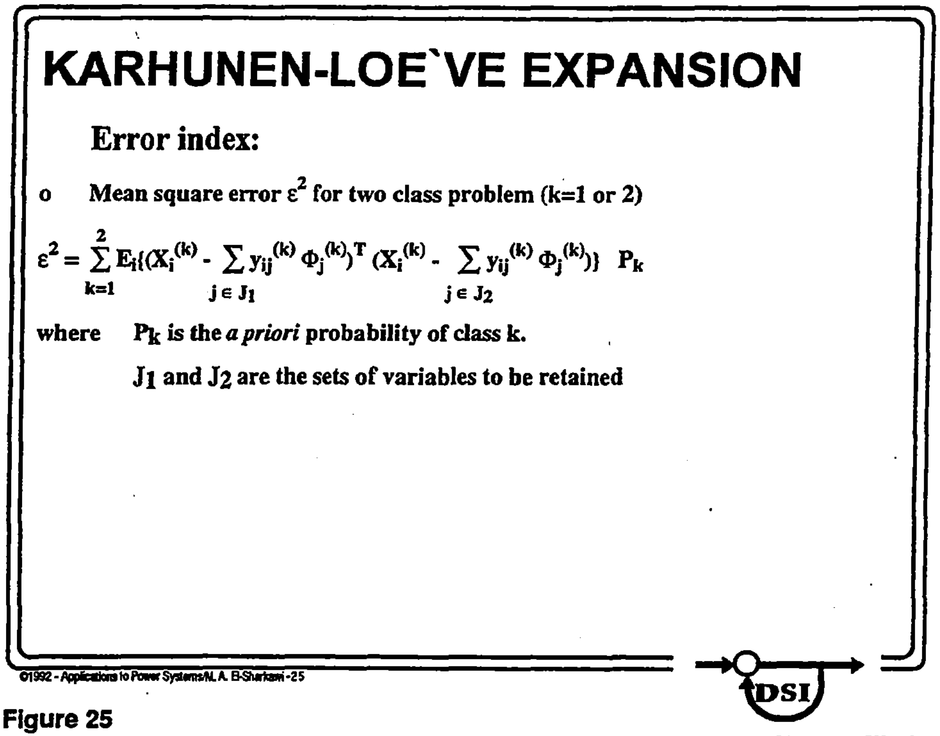
  $\Phi_j^{(k)}$ is orthonormal basis function

  $y_{ij}^{(k)}$ is a set of feature variables
- The idea is to select matrix Φ that result in reduced dimension Y as compared to X, but without significant loss in accuracy during classification.

DSI

Figure 24

Notes Notes Notes



KARHUNEN-LOE`VE EXPANSION

Error index:

o Mean square error ε^2 for two class problem (k=1 or 2)

$$\varepsilon^2 = \sum_{k=1}^{2} E_i\{(X_i^{(k)} - \sum_{j \in J_1} y_{ij}^{(k)} \Phi_j^{(k)})^T (X_i^{(k)} - \sum_{j \in J_2} y_{ij}^{(k)} \Phi_j^{(k)})\} \quad P_k$$

where P_k is the *a priori* probability of class k.

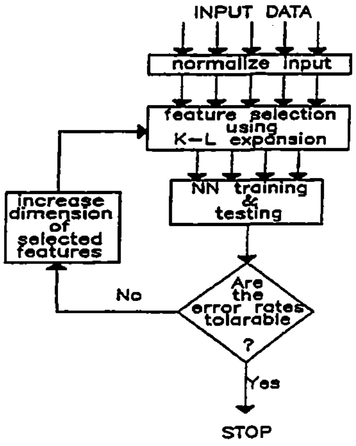
J_1 and J_2 are the sets of variables to be retained

Figure 25

Notes Notes Notes

KARHUNEN-LOE`VE EXPANSION

Algorithm



INPUT DATA

normalize input

feature selection using K–L expansion

NN training & testing

Are the error rates tolarable ?

No

increase dimension of selected features

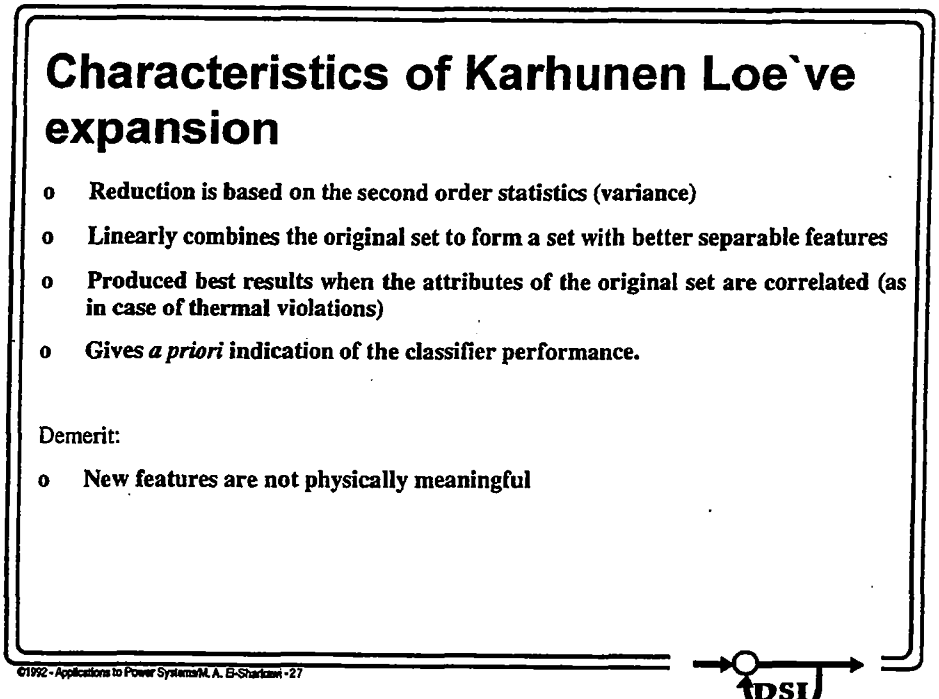
Yes

STOP

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 26

DSI

Figure 26



Characteristics of Karhunen Loe`ve expansion

- Reduction is based on the second order statistics (variance)
- Linearly combines the original set to form a set with better separable features
- Produced best results when the attributes of the original set are correlated (as in case of thermal violations)
- Gives *a priori* indication of the classifier performance.

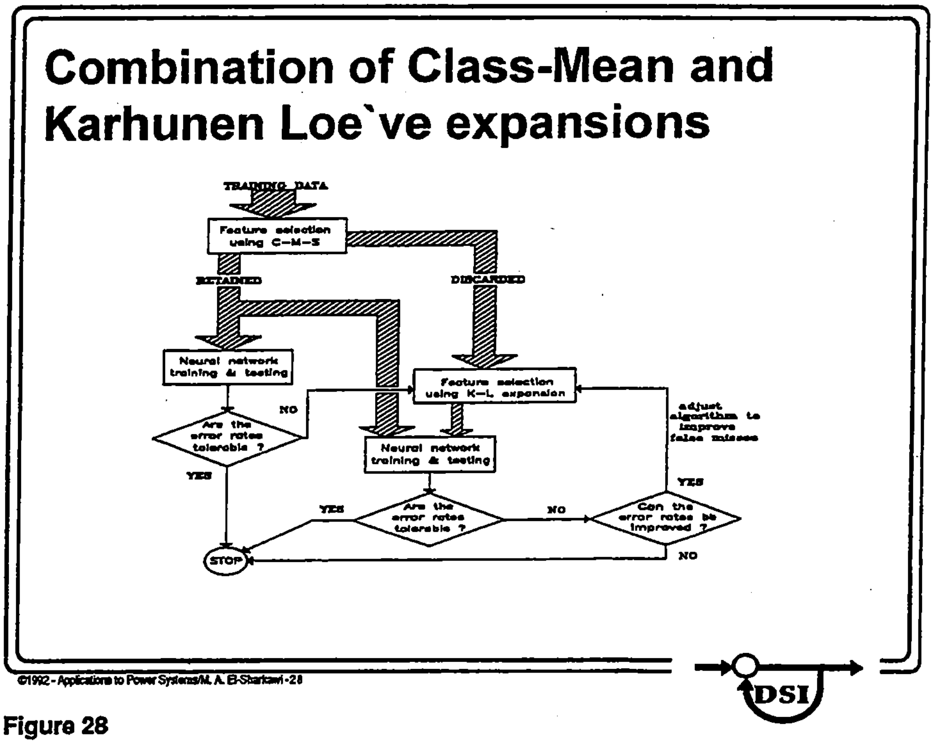
Demerit:

- New features are not physically meaningful

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 27

DSI

Figure 27



Combination of Class-Mean and Karhunen Loe`ve expansions

TRAINING DATA

Feature selection using C-M-S

RETAINED

DISCARDED

Neural network training & testing

Are the error rates tolerable ?

NO

YES

Feature selection using K-L expansion

Neural network training & testing

Are the error rates tolerable ?

YES

NO

Can the error rates be improved ?

YES

adjust algorithm to improve false misses

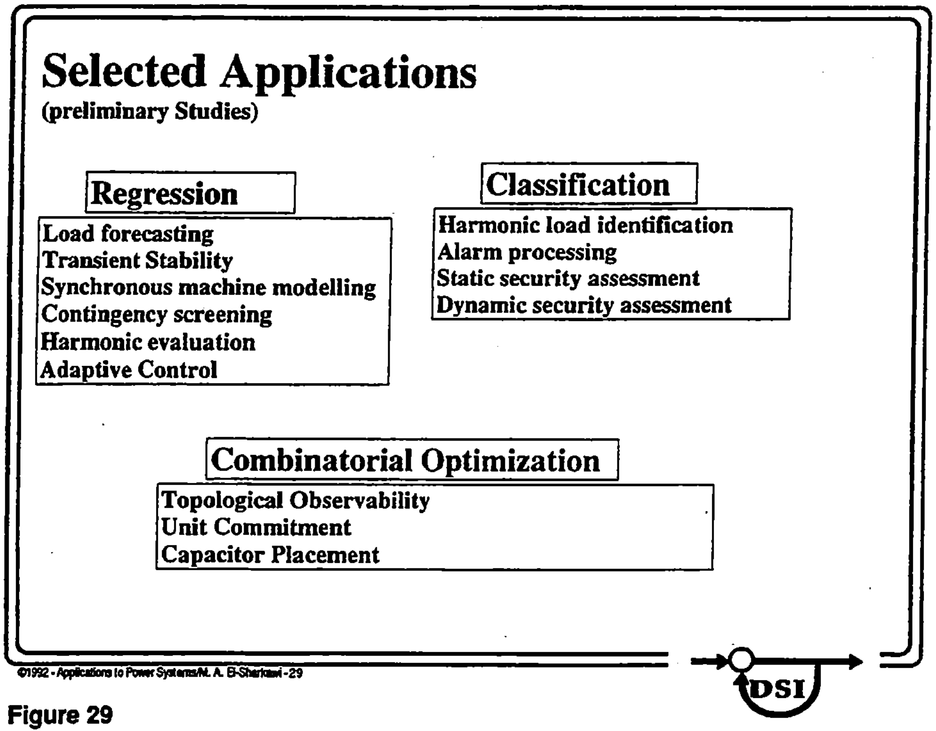
NO

STOP

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 28

Figure 28

Notes Notes Notes



Selected Applications

(preliminary Studies)

Regression

- Load forecasting
- Transient Stability
- Synchronous machine modelling
- Contingency screening
- Harmonic evaluation
- Adaptive Control

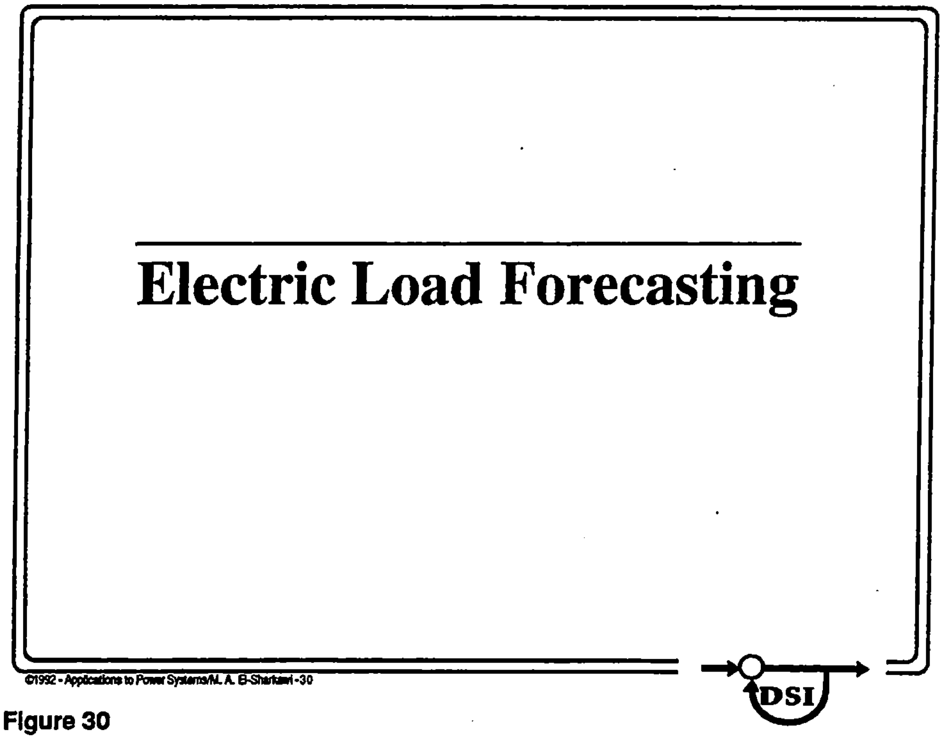
Classification

- Harmonic load identification
- Alarm processing
- Static security assessment
- Dynamic security assessment

Combinatorial Optimization

- Topological Observability
- Unit Commitment
- Capacitor Placement

Figure 29

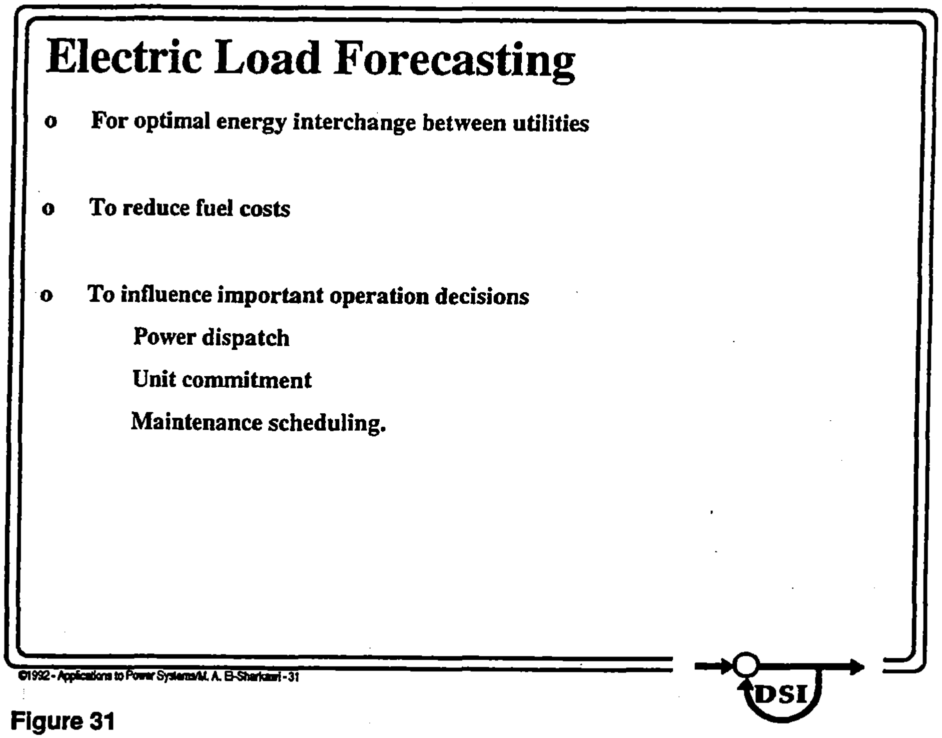


Electric Load Forecasting

Figure 30

Notes Notes Notes

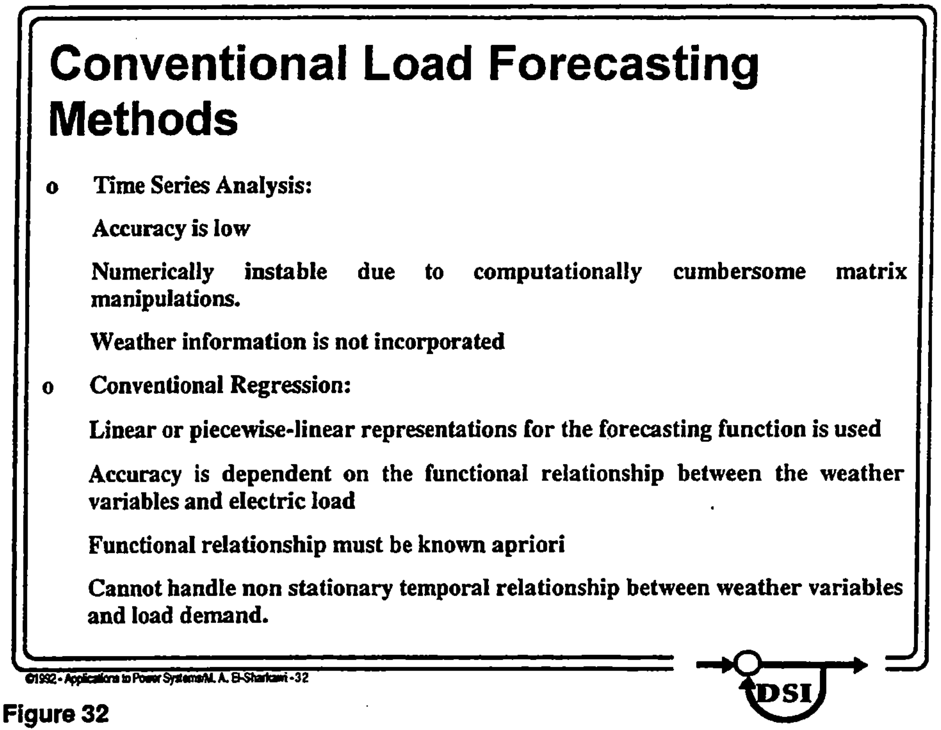
Notes Notes Notes



Electric Load Forecasting

- For optimal energy interchange between utilities
- To reduce fuel costs
- To influence important operation decisions
  - Power dispatch
  - Unit commitment
  - Maintenance scheduling.

Figure 31



Conventional Load Forecasting Methods

- Time Series Analysis:

  Accuracy is low

  Numerically instable due to computationally cumbersome matrix manipulations.

  Weather information is not incorporated

- Conventional Regression:

  Linear or piecewise-linear representations for the forecasting function is used

  Accuracy is dependent on the functional relationship between the weather variables and electric load

  Functional relationship must be known apriori

  Cannot handle non stationary temporal relationship between weather variables and load demand.

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 32

Figure 32

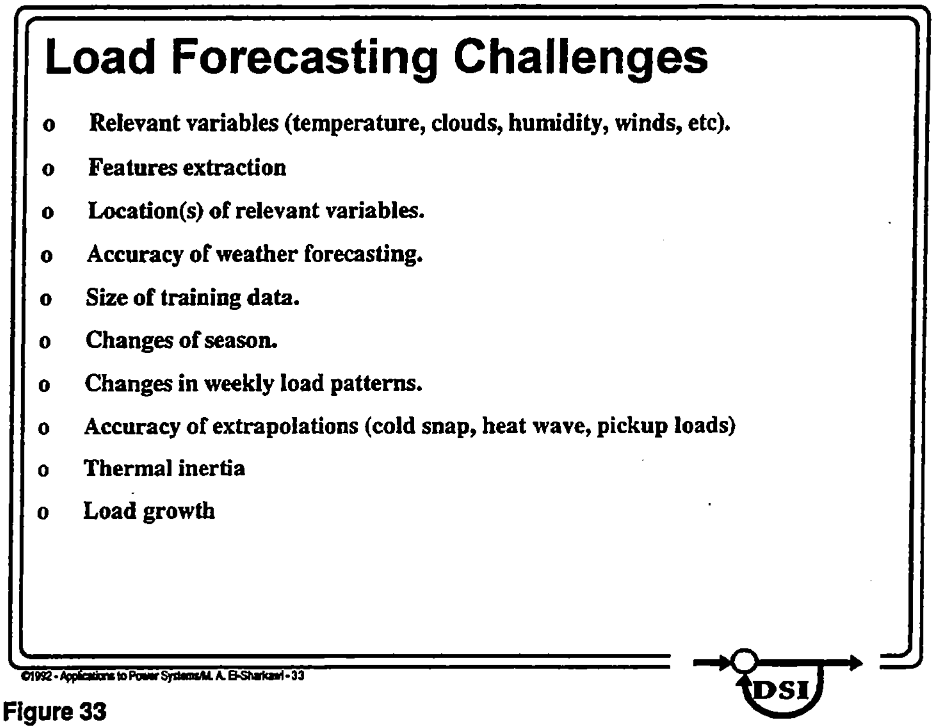
Forecasting electrical load in a power system with lead-times varying from hours to days, has obvious economic as well as other advantages. The forecasted information can be used to aid optimal energy interchange between utilities thereby saving valuable fuel costs. Forecasts also significantly influence important operations decisions such as dispatch, unit commitment and maintenance scheduling. For these reasons, considerable efforts are being invested in the development of accurate load forecasting techniques.

Most of the conventional techniques used for load forecasting can be categorized under two approaches. One treats the load demand as a time series signal and predicts the load using different time series analysis techniques. The second method recognizes the fact that the load demand is heavily dependent on weather variables. The general problem with time series approach include the inaccuracy of prediction and numerical instability [42]. The main reason for instability is not considering the weather information which is known to have a profound effect of load demand. Numerical instability is caused by computationally cumbersome matrix manipulations.

The conventional regression type approaches use linear or piecewise-linear representations for the forecasting function. The accuracy of this approach is dependent on the functional relationship between the

Notes Notes Notes

weather variables and electric load which must be known a priori. This approach cannot handle the non stationary temporal relationship between the weather variables and load demand.

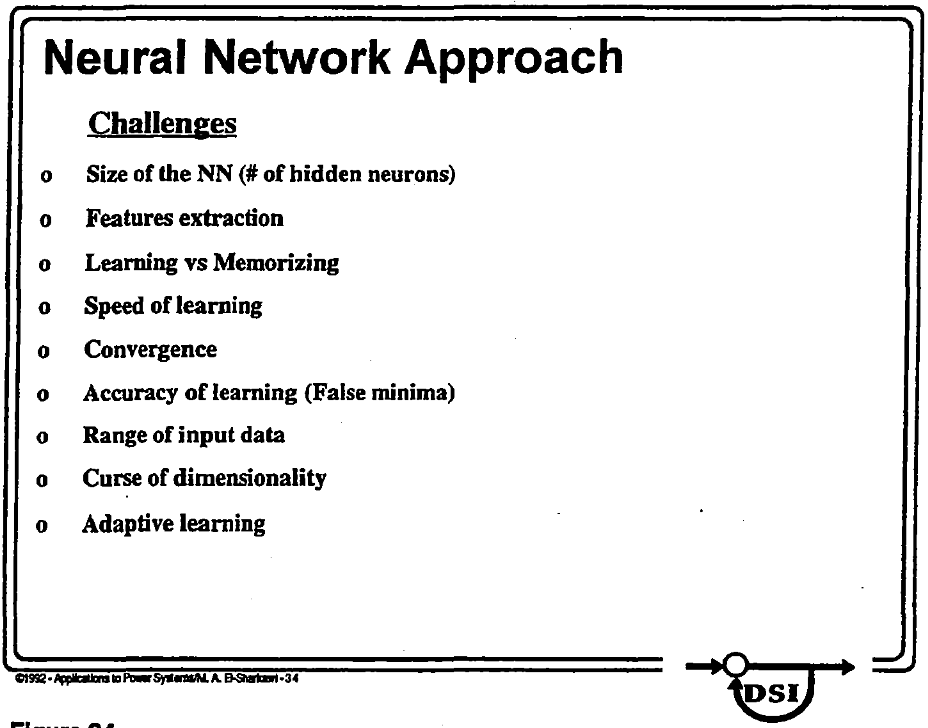


Load Forecasting Challenges

- o Relevant variables (temperature, clouds, humidity, winds, etc).
- o Features extraction
- o Location(s) of relevant variables.
- o Accuracy of weather forecasting.
- o Size of training data.
- o Changes of season.
- o Changes in weekly load patterns.
- o Accuracy of extrapolations (cold snap, heat wave, pickup loads)
- o Thermal inertia
- o Load growth

Figure 33

Notes Notes Notes



Neural Network Approach

Challenges

- Size of the NN (# of hidden neurons)
- Features extraction
- Learning vs Memorizing
- Speed of learning
- Convergence
- Accuracy of learning (False minima)
- Range of input data
- Curse of dimensionality
- Adaptive learning

Figure 34

NN can combine both time series and regression approaches to predict the load demand. A functional relationship between weather variables and electric load is not needed. This is because NN can technically generate this functional relationship by learning the training data. In other words, the nonlinear mapping between the inputs and outputs is implicitly imbedded in the NN.

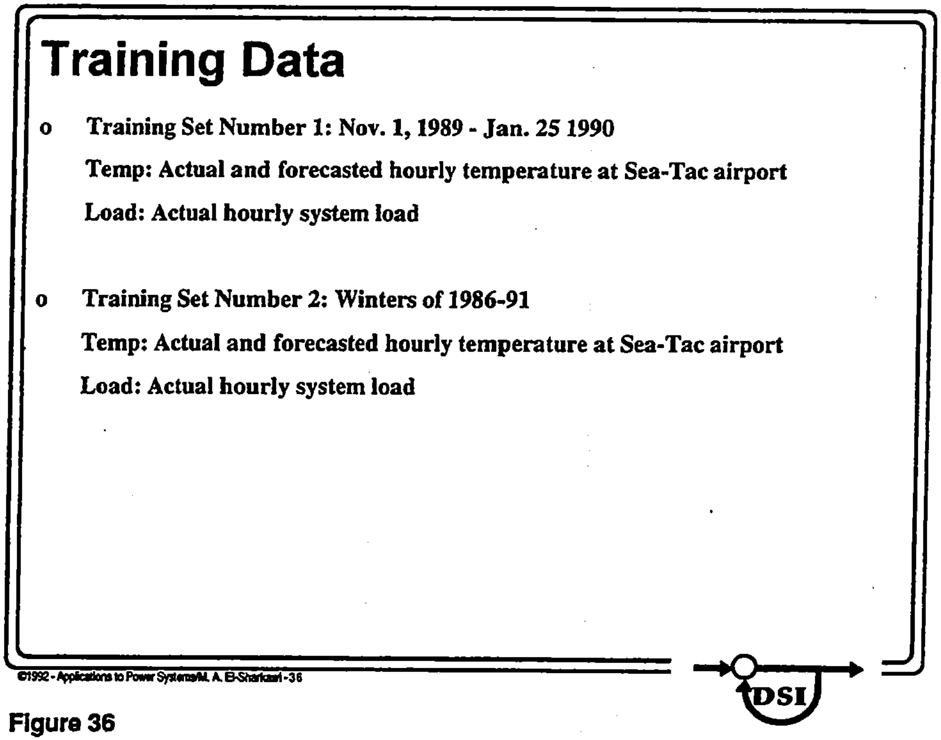
Objectives:

To forecast electric load patterns/variables based on forecasted temperature

- o AM and PM peaks of each day
- o Average load of the day
- o Hourly load (24 - 48 hours lead forecast)
- o Weekdays and weekends
- o Holidays

Figure 35

Notes Notes Notes



Training Data

- o Training Set Number 1: Nov. 1, 1989 - Jan. 25 1990

  Temp: Actual and forecasted hourly temperature at Sea-Tac airport

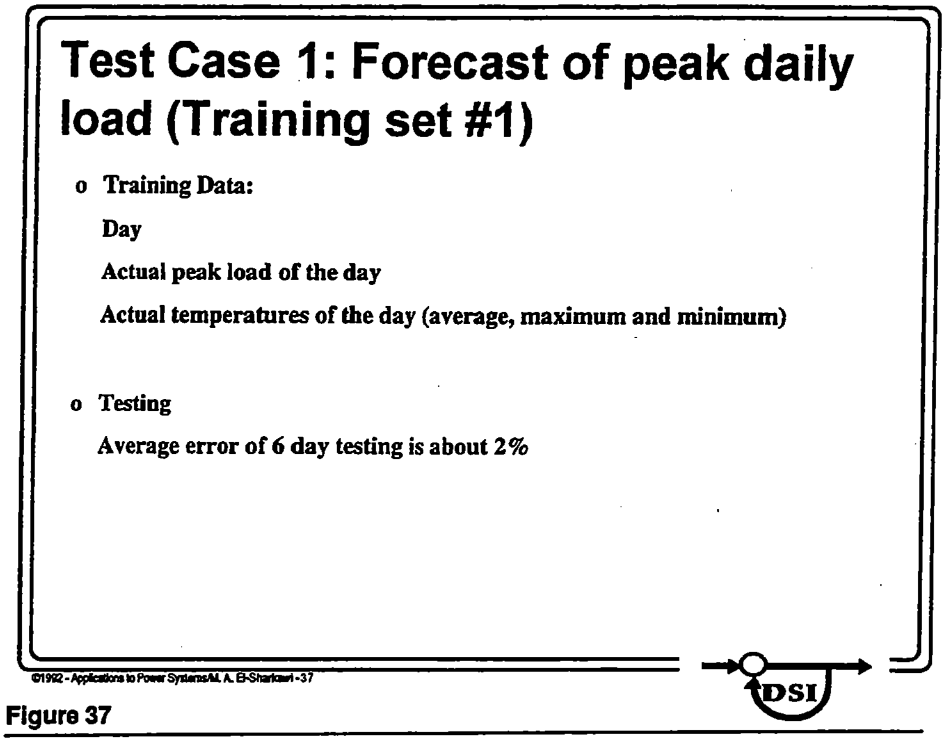
  Load: Actual hourly system load

- o Training Set Number 2: Winters of 1986-91

  Temp: Actual and forecasted hourly temperature at Sea-Tac airport

  Load: Actual hourly system load

Figure 36



Test Case 1: Forecast of peak daily load (Training set #1)

- Training Data:

  Day

  Actual peak load of the day

  Actual temperatures of the day (average, maximum and minimum)

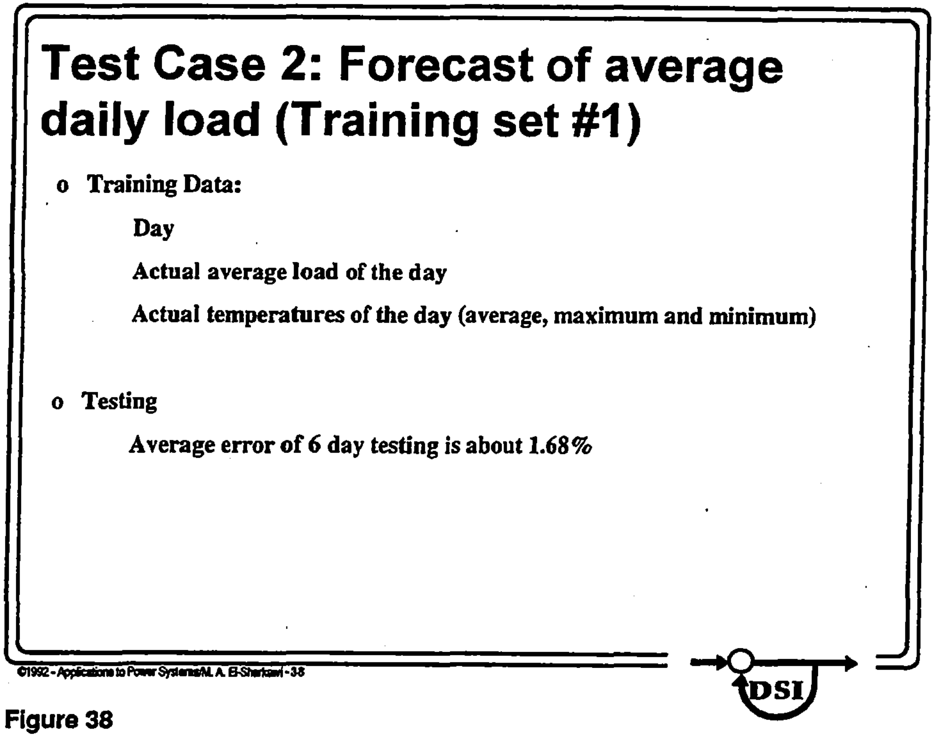
- Testing

  Average error of 6 day testing is about 2%

Figure 37

Notes Notes Notes

Notes Notes Notes



Test Case 2: Forecast of average daily load (Training set #1)

- o Training Data:

  Day

  Actual average load of the day

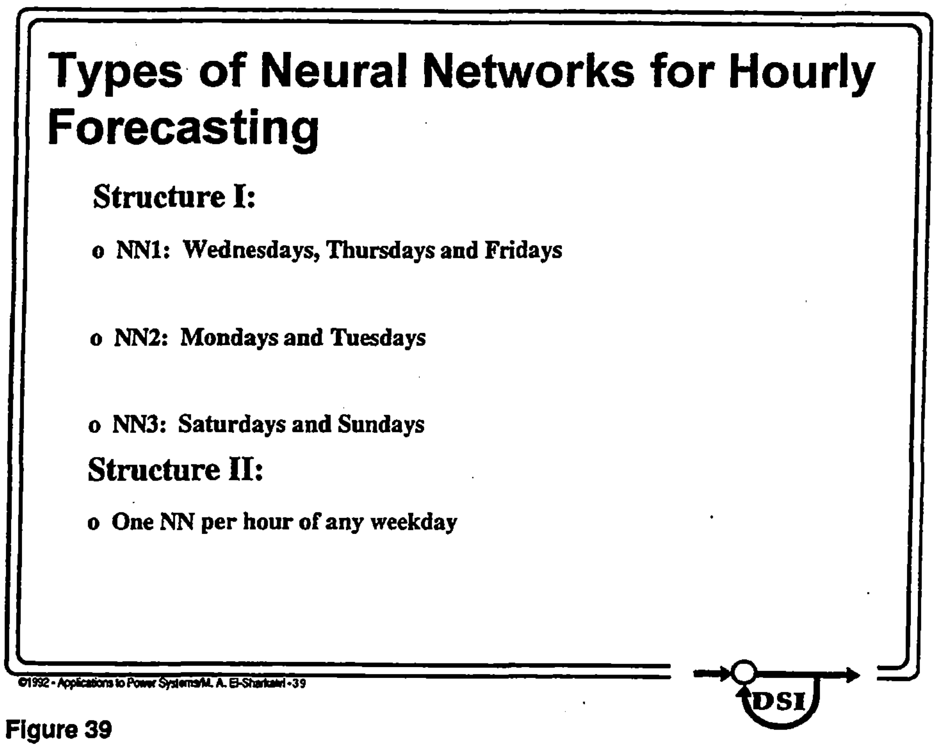
  Actual temperatures of the day (average, maximum and minimum)

- o Testing

  Average error of 6 day testing is about 1.68%

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 38

Figure 38



Types of Neural Networks for Hourly Forecasting

Structure I:

- o NN1: Wednesdays, Thursdays and Fridays
- o NN2: Mondays and Tuesdays
- o NN3: Saturdays and Sundays

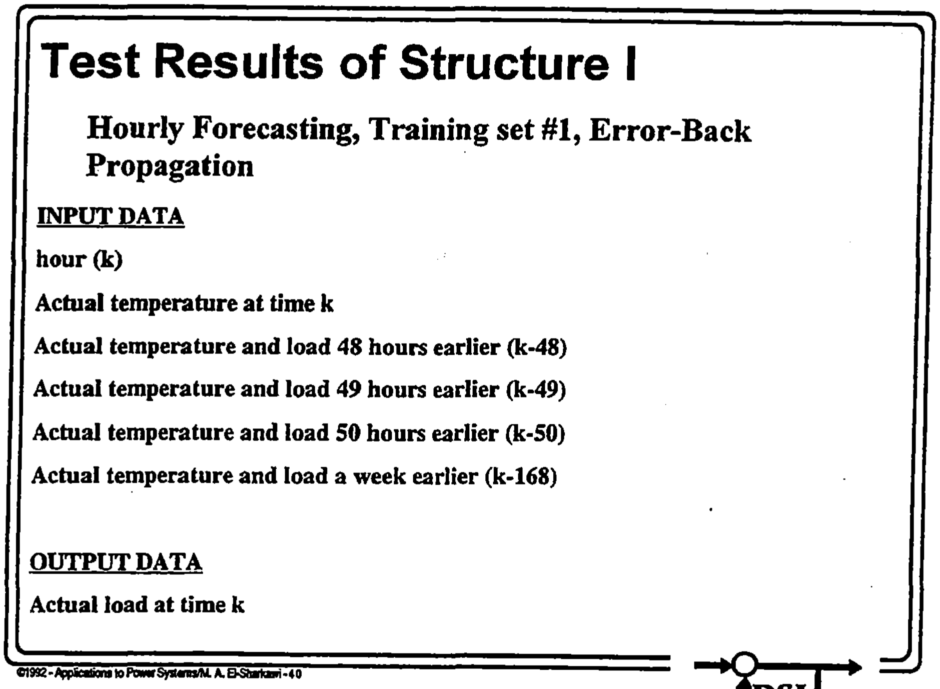
Structure II:

- o One NN per hour of any weekday

DSI

Figure 39

Notes Notes Notes



Test Results of Structure I

Hourly Forecasting, Training set #1, Error-Back Propagation

INPUT DATA

hour (k)

Actual temperature at time k

Actual temperature and load 48 hours earlier (k-48)

Actual temperature and load 49 hours earlier (k-49)

Actual temperature and load 50 hours earlier (k-50)

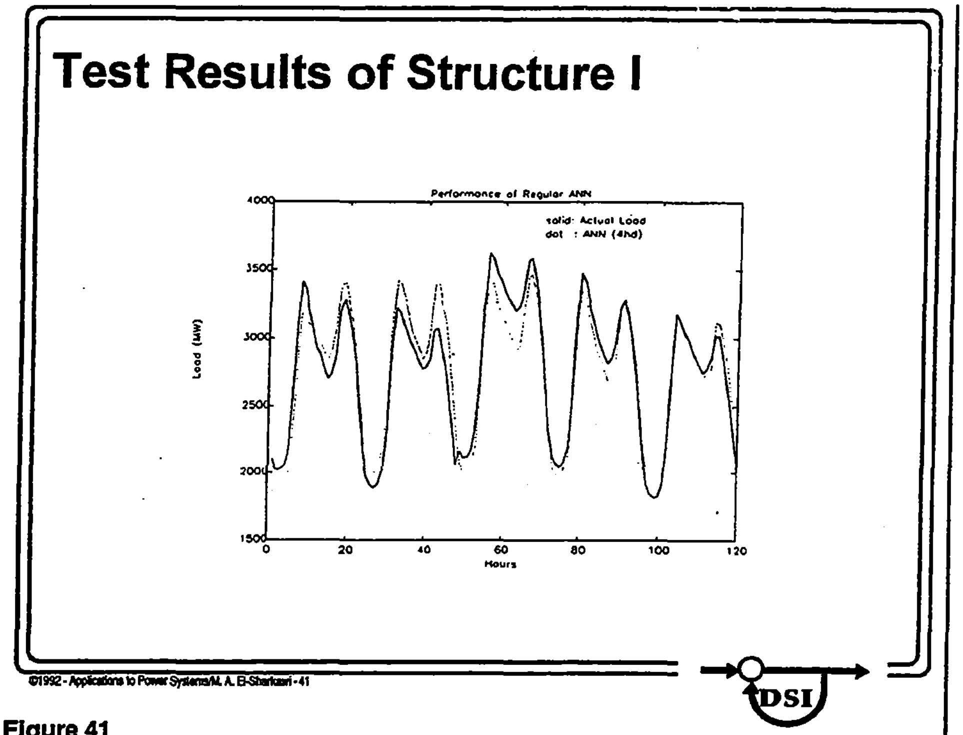
Actual temperature and load a week earlier (k-168)

OUTPUT DATA

Actual load at time k

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 40

Figure 40



Test Results of Structure I

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 41

Figure 41

The NN approach proposed in [42,54] uses previous load data combined with actual and forecasted weather variables as inputs, and the load demand as the output. As an example, to predict the load at the k^{th} hour on a 24 hour period, the NN uses the following input/output configuration.

NN inputs : k, L(24,k), T(24,k), L(m,k), T(m,k) and $T_p(k)$

NN output : L(k)

where,

k - hour of predicted load

m - lead time

L(x,k) - load at x hours before hour k

T(x,k) - temperature at x hours before hour k

$T_p(k)$ - predicted temperature at hour k

During training, the actual temperature T(k) is used instead of $T_p(k)$. Different NNs are trained to predict the load demand at varying lead

Notes Notes Notes

times. The results are reported too be better than those obtained through some of the existing extensive regression techniques.

One of the test results presented in [42] is given for brevity. Five sets of actual load and temperature data were used in the study. Each set contained data corresponding to 8 consecutive days as shown in table 1. Out of each set, data corresponding to the six weekdays were selected. No weekends or holidays were included.

Table 1: Test data sets

sets	Test data from
Set #1	01/23/89 - 01/30/89
Set #2	11/09/88 - 11/17/88
Set #3	11/18/88 - 11/29/88
Set #4	12/08/88 - 12/15/88
Set #5	12/27/88 - 01/04/89

From [42] courtesy of IEEE, © IEEE,1990

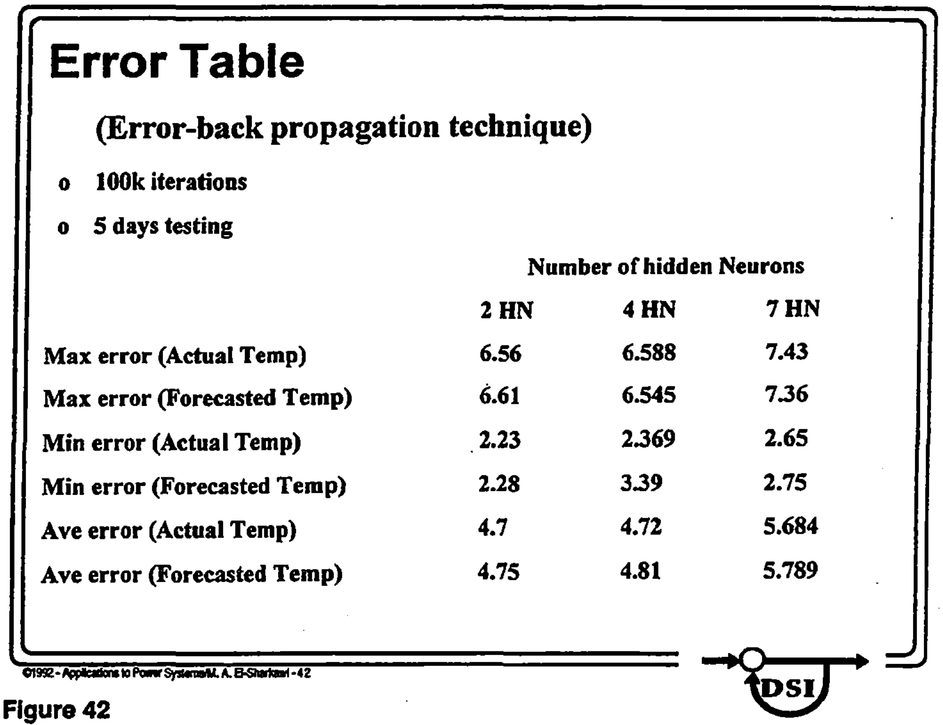
The NN was trained to forecast the hourly load with one hour lead time. Table 2 shows the forecasting error(%) of each day in the test sets. Each day's result is averaged over a 24 hour period. The average error for the 5 test sets was found to be 1.40%.

Table 2: Error(%) of hourly load forecasting with one hour lead time

days	set #1	set #2	set #3	set #4	set #5
day 1	(*)	1.20	1.41	1.17	(*)
day 2	1.67	1.48	(*)	1.58	2.18
day 3	1.08	(*)	1.04	(*)	1.68
day 4	1.40	1.34	1.42	1.20	1.73
day 5	1.30	1.41	(*)	1.20	(*)
day 6	(*)	1.51	1.29	1.68	0.98
average	1.35	1.39	1.29	1.36	1.64

(*: predicted temperature, T_p is not available)

From [42] courtesy of IEEE, © IEEE,1990



Error Table

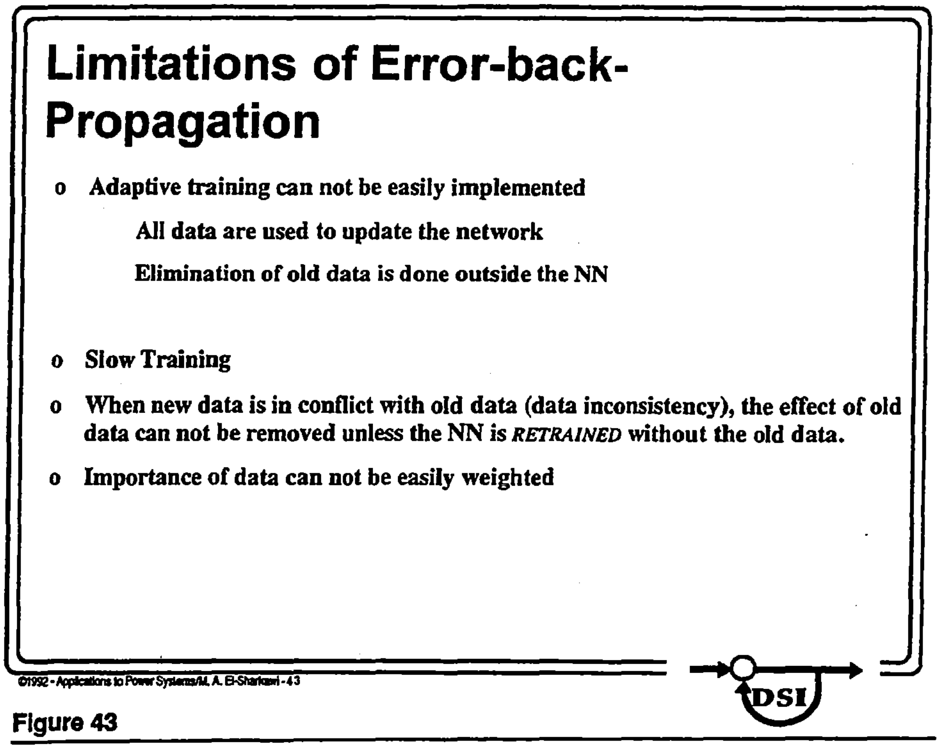
(Error-back propagation technique)

- 100k iterations
- 5 days testing

	Number of hidden Neurons		
	2 HN	4 HN	7 HN
Max error (Actual Temp)	6.56	6.588	7.43
Max error (Forecasted Temp)	6.61	6.545	7.36
Min error (Actual Temp)	2.23	2.369	2.65
Min error (Forecasted Temp)	2.28	3.39	2.75
Ave error (Actual Temp)	4.7	4.72	5.684
Ave error (Forecasted Temp)	4.75	4.81	5.789

Figure 42

Notes Notes Notes



Limitations of Error-back-Propagation

- Adaptive training can not be easily implemented

  All data are used to update the network

  Elimination of old data is done outside the NN

- Slow Training
- When new data is in conflict with old data (data inconsistency), the effect of old data can not be removed unless the NN is *RETRAINED* without the old data.
- Importance of data can not be easily weighted

Figure 43

Test Results of Structure I

(Training set #1) Adaptive NN

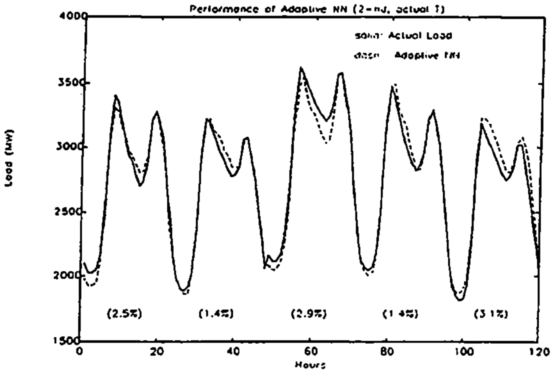


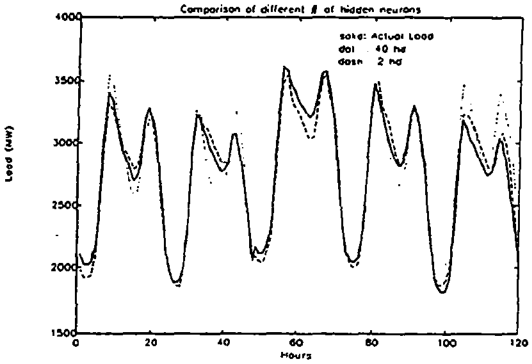
Figure 44

Notes Notes Notes

Notes Notes Notes

Effect of hidden Neurons

Structure I (Training Set #1)

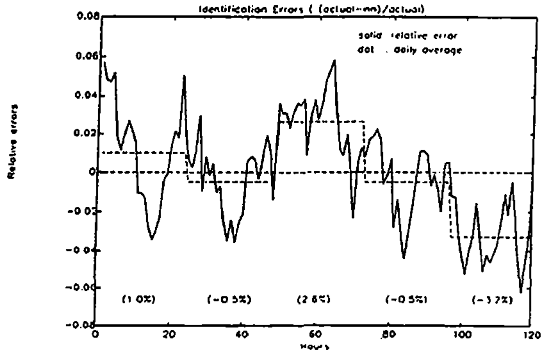


DSI

Figure 45

Error Analysis (Training Set #1)

4 Hidden Neurons

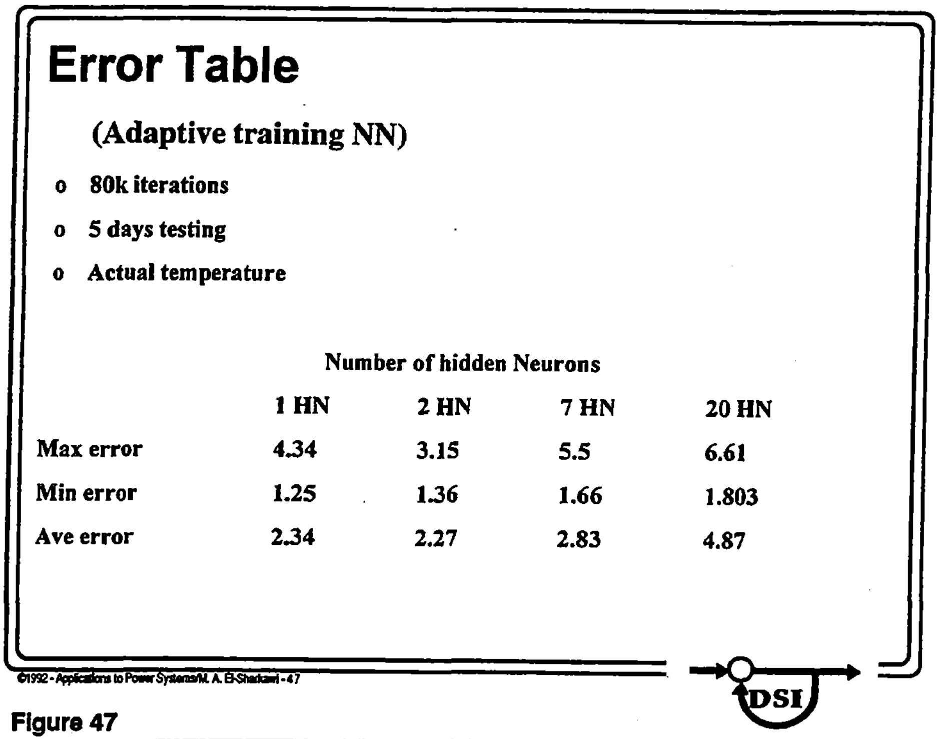


DSI

Figure 46

Notes Notes Notes

Notes Notes Notes



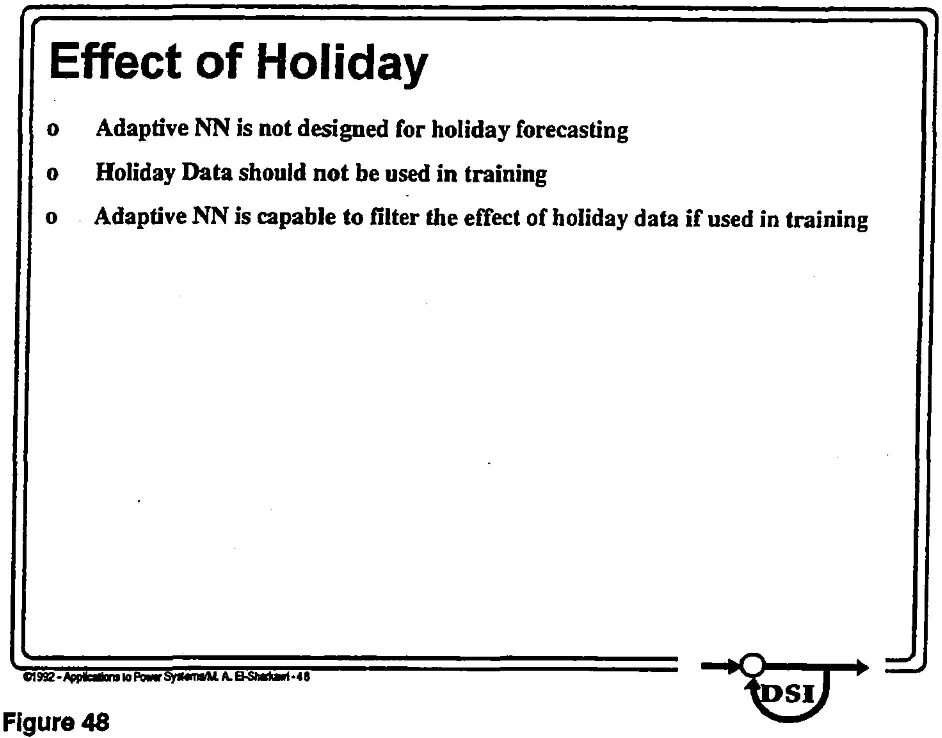
Error Table

(Adaptive training NN)

- o 80k iterations
- o 5 days testing
- o Actual temperature

	Number of hidden Neurons			
	1 HN	2 HN	7 HN	20 HN
Max error	4.34	3.15	5.5	6.61
Min error	1.25	1.36	1.66	1.803
Ave error	2.34	2.27	2.83	4.87

Figure 47



Effect of Holiday

- Adaptive NN is not designed for holiday forecasting
- Holiday Data should not be used in training
- Adaptive NN is capable to filter the effect of holiday data if used in training

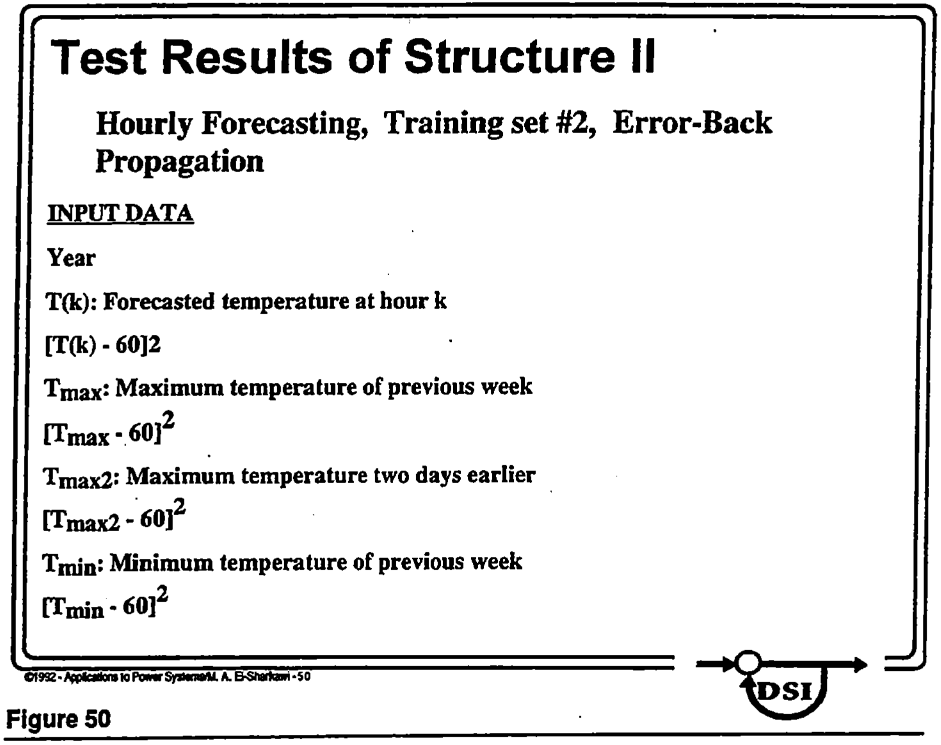
Figure 48

Notes Notes Notes

Notes Notes Notes

Effect of Holiday (Training Set #1)

Figure 49



Test Results of Structure II

Hourly Forecasting, Training set #2, Error-Back Propagation

INPUT DATA

Year

T(k): Forecasted temperature at hour k

[T(k) - 60]2

T_{max}: Maximum temperature of previous week

$[T_{max} - 60]^2$

T_{max2}: Maximum temperature two days earlier

$[T_{max2} - 60]^2$

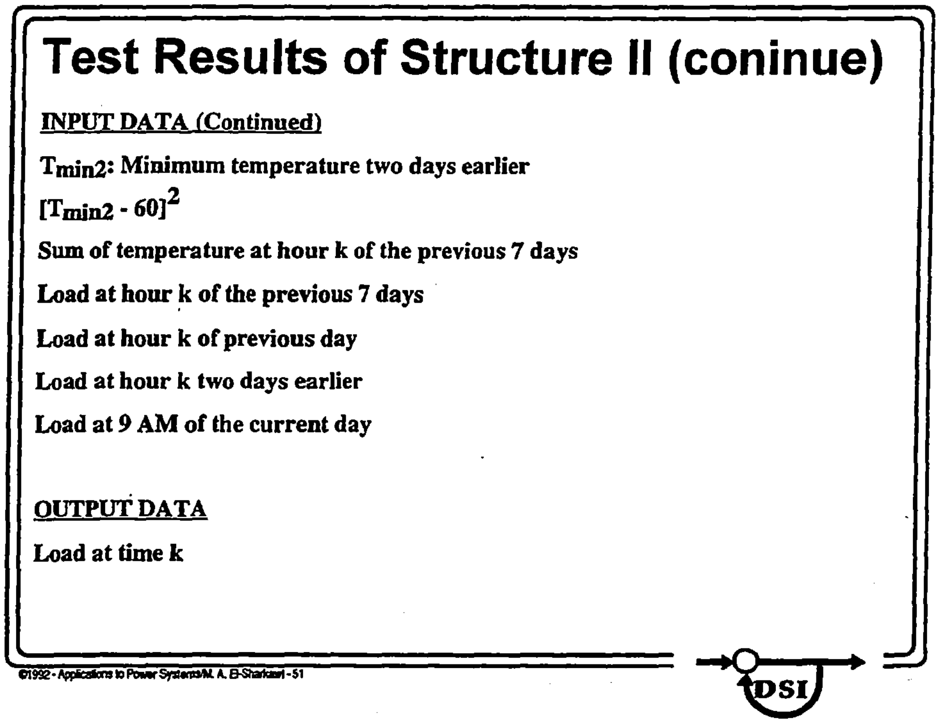
T_{min}: Minimum temperature of previous week

$[T_{min} - 60]^2$

Figure 50

Notes Notes Notes

Notes Notes Notes



Test Results of Structure II (coninue)

INPUT DATA (Continued)

T_{min2}: Minimum temperature two days earlier

$[T_{min2} - 60]^2$

Sum of temperature at hour k of the previous 7 days

Load at hour k of the previous 7 days

Load at hour k of previous day

Load at hour k two days earlier

Load at 9 AM of the current day

OUTPUT DATA

Load at time k

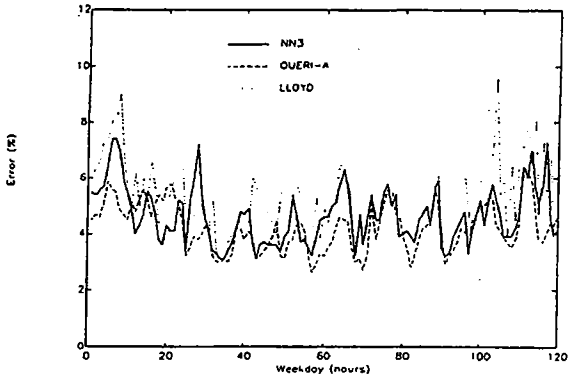
Figure 51

Notes Notes Notes

Test Results of Structure II

(Training set #2), Forecasting Contest

(Courtesy of Puget Sound Power and Light Company)

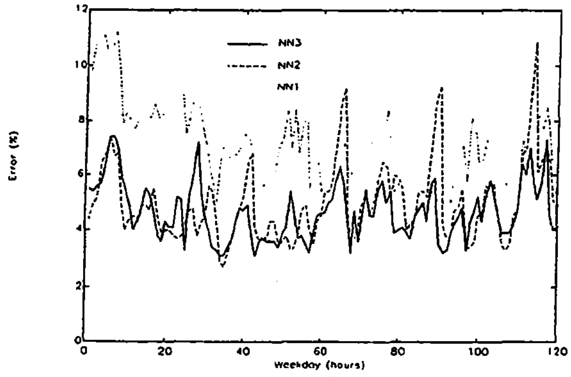


DSI

Figure 52

Notes Notes Notes

Test Results of Structure II (continue)

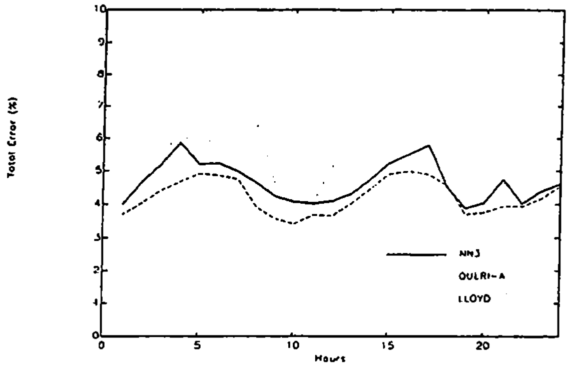


©1992 - Applications to Power Systems/M. A. El-Sharkawi - 53

DSI

Figure 53

Test Results of Structure II (continue)



DSI

Figure 54

Notes Notes Notes

Test Results of Structure II (continue)

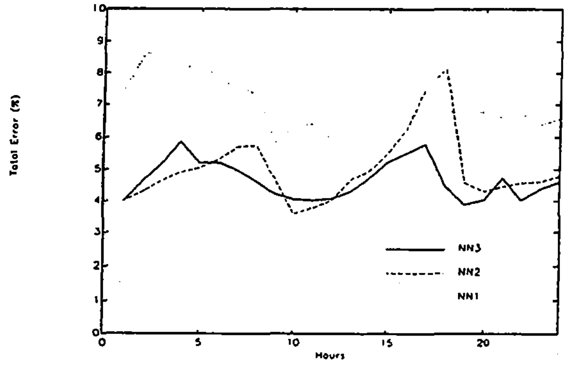
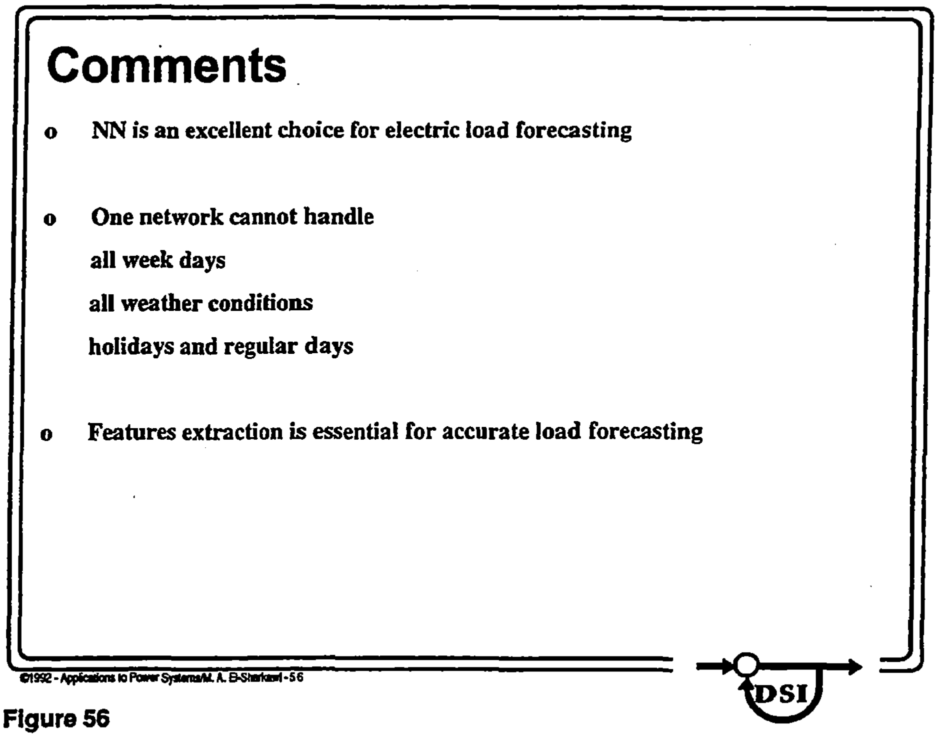


Figure 55

The figure shows the total error calculated over the weekdays hours. The error is an average of the aggregated values. Lloyd is a forecasting done by a Puget Power expert. Queri-A is the best commercial software for load forecasting among several codes being tested by Puget Power.

NN1 is referred to Structure I while NN2 is for Structure II. NN2 is another structure similar to Structure II except that one NN is forecasting three hours. Note that Structure II is designed so that a single NN is to forecast only one hour.



Comments

- o NN is an excellent choice for electric load forecasting

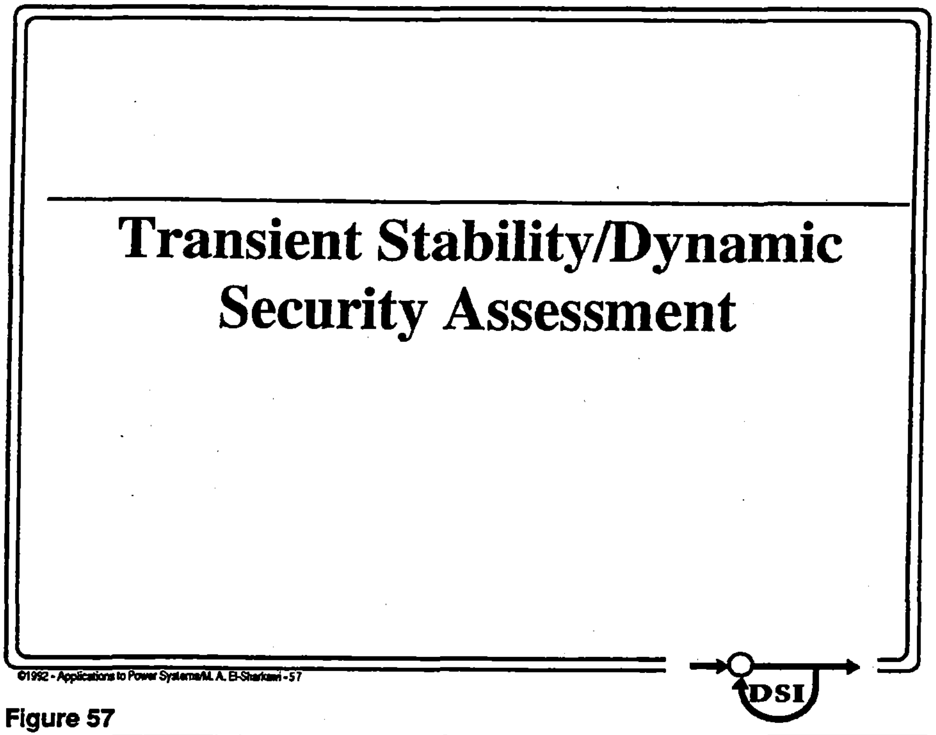
- o One network cannot handle
  all week days
  all weather conditions
  holidays and regular days

- o Features extraction is essential for accurate load forecasting

Figure 56

The results show that NN can be trained to predict the load demand by among its training patterns. However, one network cannot handle all cases where enough and sparse representation exist in the training test. For example, a NN trained to predict electric loads of normal weather conditions, may not do accurate prediction during extreme weather conditions such as cold snaps and heat waves. To predict electric loads under these conditions, a separate NN may be needed. Also the holidays cannot be accurately predicted. It is also worth mentioning that the above restrictions are also applied to all existing techniques.

Notes Notes Notes

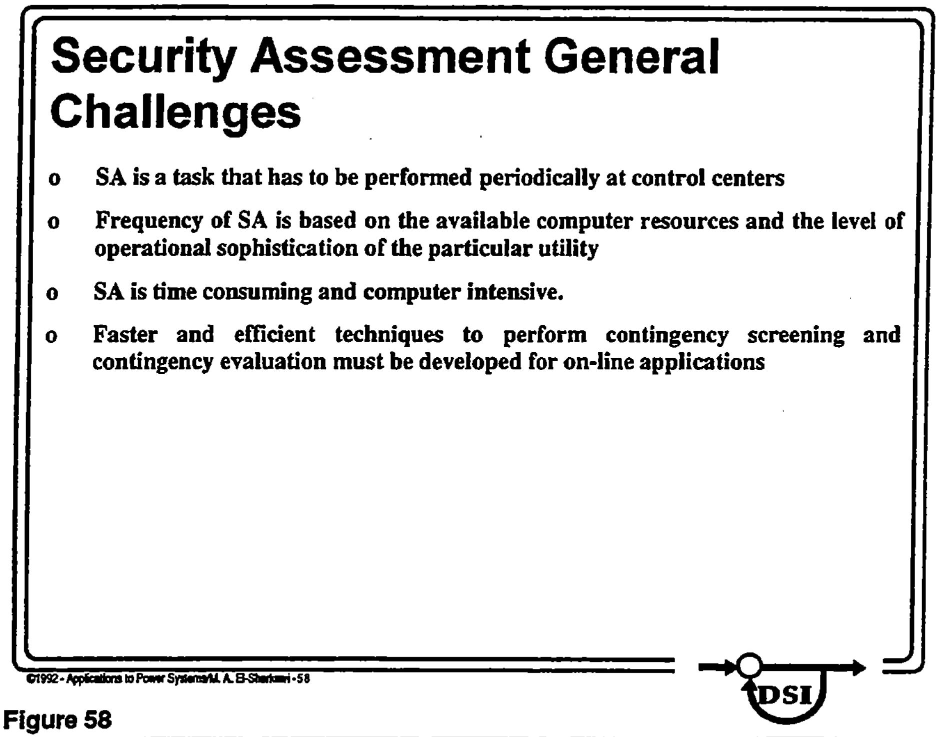


Transient Stability/Dynamic Security Assessment

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 57

Figure 57

Transient stability is determined by observing the variation of δ_i s' as a function of time in the post-fault period. Power system is said to be transiently stable for a given disturbance if the oscillations of all rotor angles damped out and eventually settled down to values within the safe operating constraints of the system. For any disturbance, the transient stability of a power system depends on three basic components: the magnitude of the disturbance, the duration of the disturbance and the speed of the protective devices.



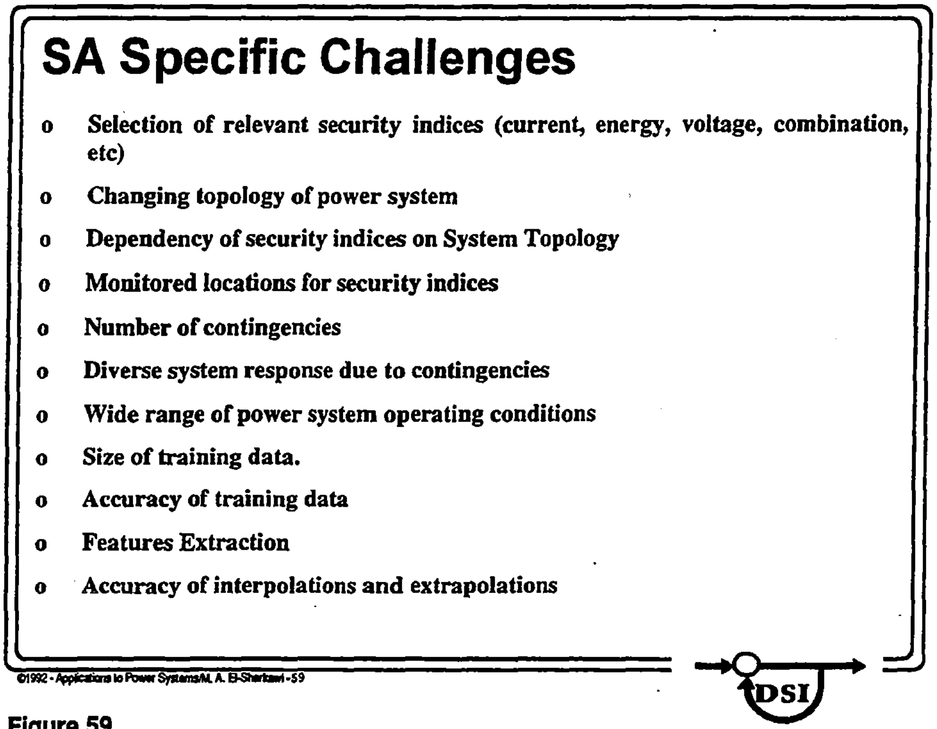
Security Assessment General Challenges

- SA is a task that has to be performed periodically at control centers
- Frequency of SA is based on the available computer resources and the level of operational sophistication of the particular utility
- SA is time consuming and computer intensive.
- Faster and efficient techniques to perform contingency screening and contingency evaluation must be developed for on-line applications

Figure 58

Notes Notes Notes

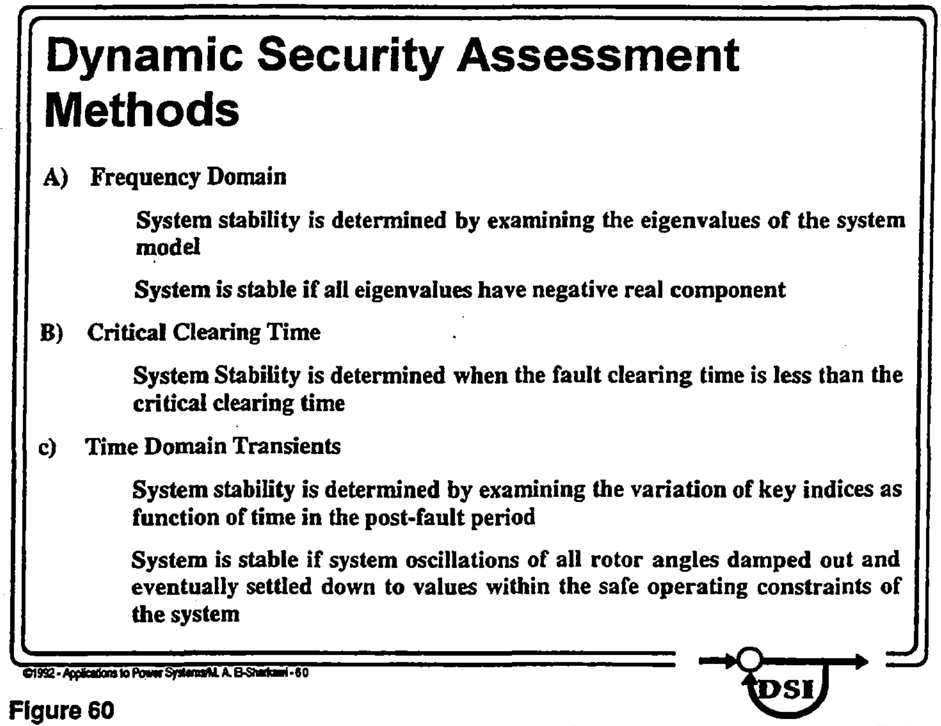
Notes Notes Notes



SA Specific Challenges

- o Selection of relevant security indices (current, energy, voltage, combination, etc)
- o Changing topology of power system
- o Dependency of security indices on System Topology
- o Monitored locations for security indices
- o Number of contingencies
- o Diverse system response due to contingencies
- o Wide range of power system operating conditions
- o Size of training data.
- o Accuracy of training data
- o Features Extraction
- o Accuracy of interpolations and extrapolations

Figure 59



Dynamic Security Assessment Methods

A) Frequency Domain

System stability is determined by examining the eigenvalues of the system model

System is stable if all eigenvalues have negative real component

B) Critical Clearing Time

System Stability is determined when the fault clearing time is less than the critical clearing time

c) Time Domain Transients

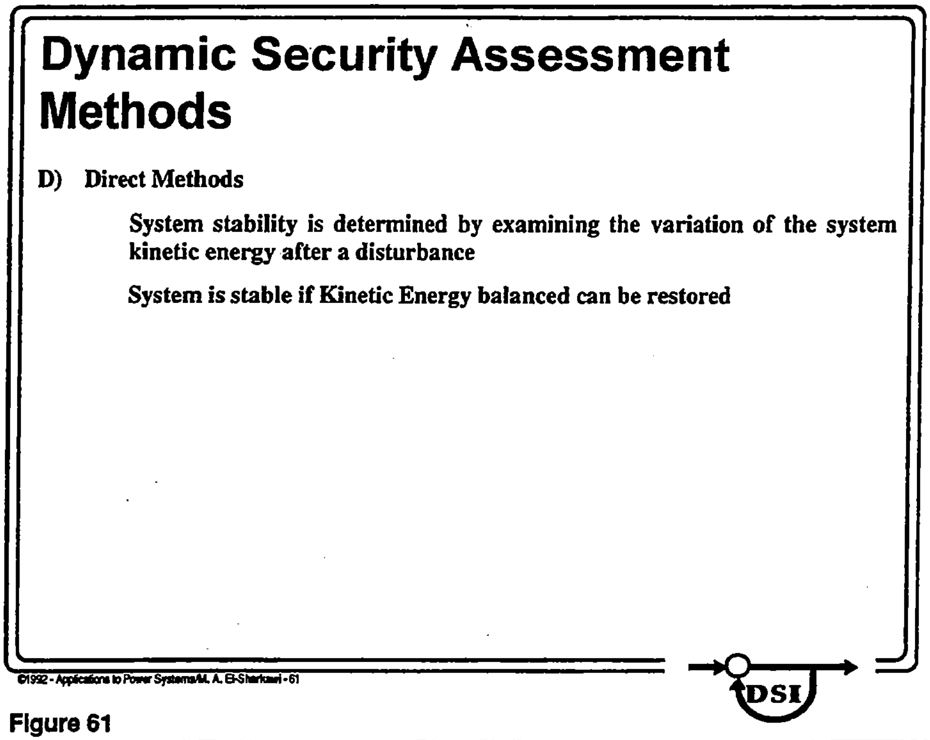
System stability is determined by examining the variation of key indices as function of time in the post-fault period

System is stable if system oscillations of all rotor angles damped out and eventually settled down to values within the safe operating constraints of the system

Figure 60

Notes Notes Notes

Notes Notes Notes



Dynamic Security Assessment Methods

D) Direct Methods

System stability is determined by examining the variation of the system kinetic energy after a disturbance

System is stable if Kinetic Energy balanced can be restored

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 61

Figure 61

Notes Notes Notes

A) Frequency Domain Approach of Dynamic Security

DSI

Figure 62

Notes Notes Notes

Modular NN Concept

- A single NN approach may be an enormous computational exercise for large power systems;
  - Large number of attributes
  - A wide range of operating conditions.
- One way of reducing the dimensional complexity is to use a modular approach
  - Security problem is divide into smaller tasks
  - Topology is reduced
  - Features extraction is implemented

Figure 63

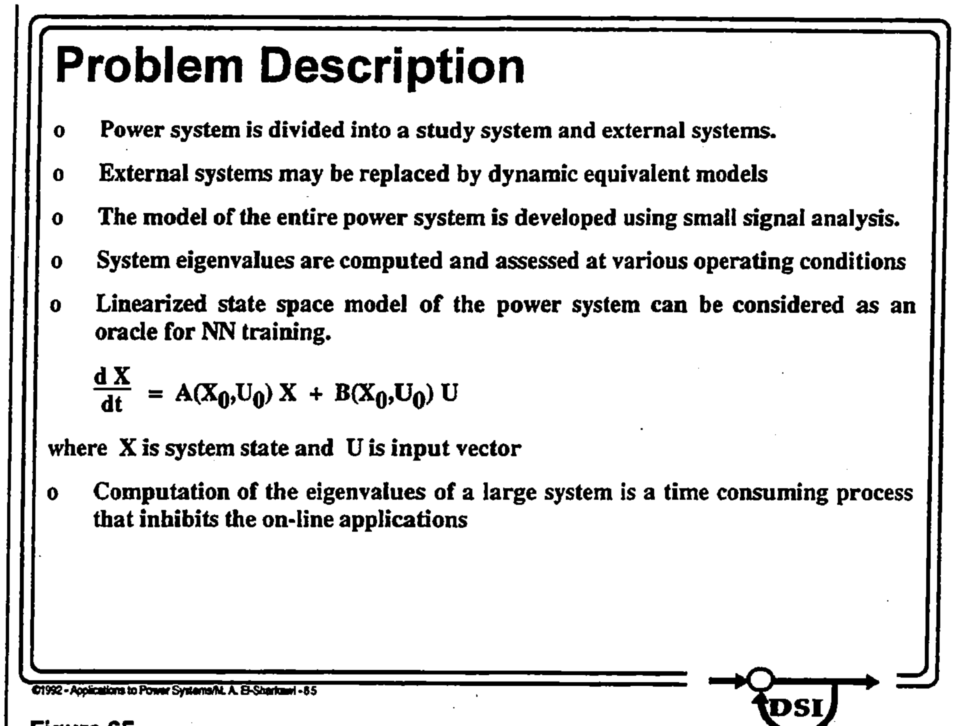
Dynamic Security Scenario

- o Small signal stability analysis
- o Power system model is linearized around a selected operating point
- o stability is predicted by evaluating system's eigenvalues
- o linearization and eigenvalue analysis must be repeated for all topologies and operating conditions
- o On-line DSA may not be possible

DSI

Figure 64

In dynamic security, or small signal stability analysis, the power system model is linearized around a selected operating point and the corresponding system eigenvalues evaluated to predict system stability. For a power system to be evaluated at all possible operating conditions, the linearization and eigenvalue analysis has to be repeated for all the cases. This is a time consuming process that poses a challenge to performing dynamic security assessment (DSA) on-line. Thus NN may provide a potential avenue toward achieving this objective.



Problem Description

- o Power system is divided into a study system and external systems.
- o External systems may be replaced by dynamic equivalent models
- o The model of the entire power system is developed using small signal analysis.
- o System eigenvalues are computed and assessed at various operating conditions
- o Linearized state space model of the power system can be considered as an oracle for NN training.

$$\frac{dX}{dt} = A(X_0,U_0)\,X + B(X_0,U_0)\,U$$

where X is system state and U is input vector

- o Computation of the eigenvalues of a large system is a time consuming process that inhibits the on-line applications

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 85

DSI

Figure 65

In dynamic security assessment, the power system stability is evaluated via frequency domain analysis. The power system is divided into a study system and an external system. The external system can be replaced by a dynamic equivalent models while the study system is modelled in detail. The model of the entire power system is developed using the small signal analysis. The eigenvalues of the system are then computed and assessed at various operating conditions [16]. The linearized state space model of the power system can be considered as an oracle for NN training. The linearized model is derived by combining the set of state and algebraic equations listed in section 6.3 for all generators in the study area of the power system. The composite linearized state spaces equation take the form,

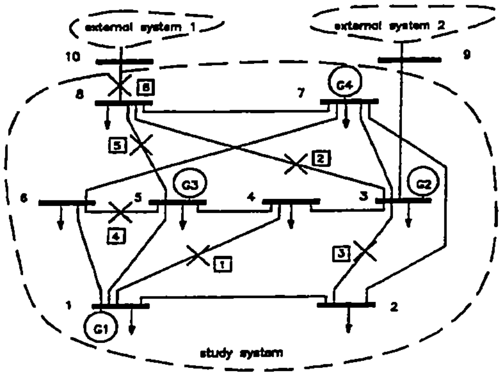
$$\frac{d\Uparrow X}{dt} = A\,(X_0,U_0)\Uparrow X + B\,(X_0,U_0)\Uparrow U$$

where $X = X_0 + \Uparrow X$ and $U = U_0 + \Uparrow U$ are the state and input vectors for the system. The stability of the system is determined by calculating the eigenvalues of the system matrix $A\,(X_0,U_0)$. Any eigenvalue with a non-negative real component is unstable mode of operation.

The stability of the power system as described above is heavily dependent on the operating condition and topology of the power system. The computation of the eigenvalues of a large system is a time consuming process that inhibits the on-line applications.

Notes Notes Notes

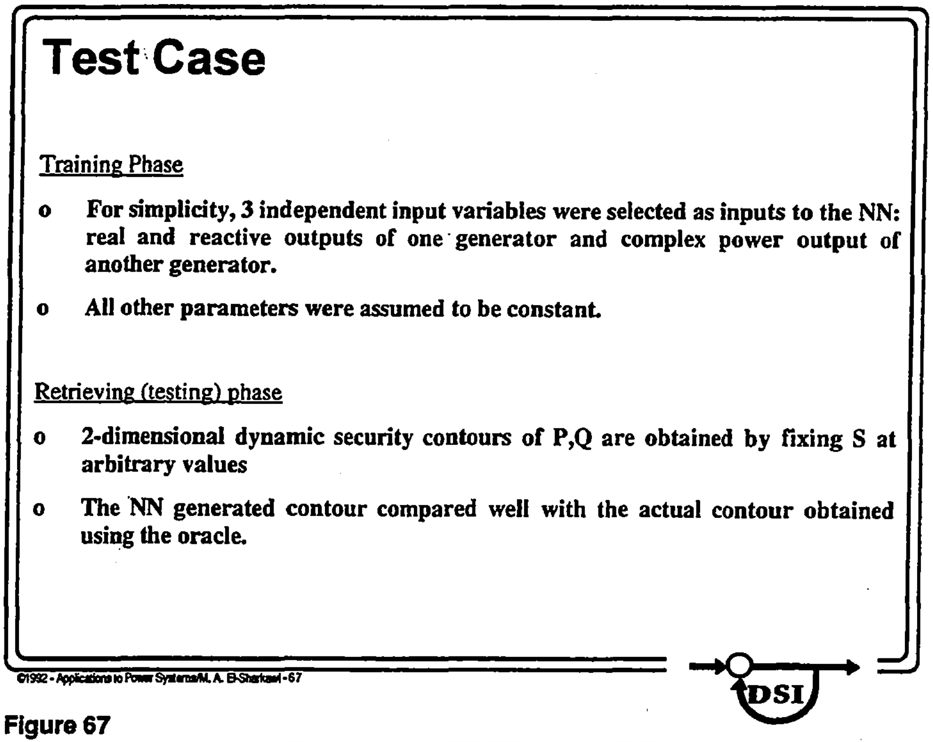
Test Case Extended IEEE-8 Bus System



- o 10 buses (b), 16 lines, 2000 unbiased patterns (i), 30 attributes
- o 2000 patterns, each with 30 attribute

Figure 66

Notes Notes Notes



Test Case

Training Phase

- For simplicity, 3 independent input variables were selected as inputs to the NN: real and reactive outputs of one generator and complex power output of another generator.
- All other parameters were assumed to be constant.

Retrieving (testing) phase

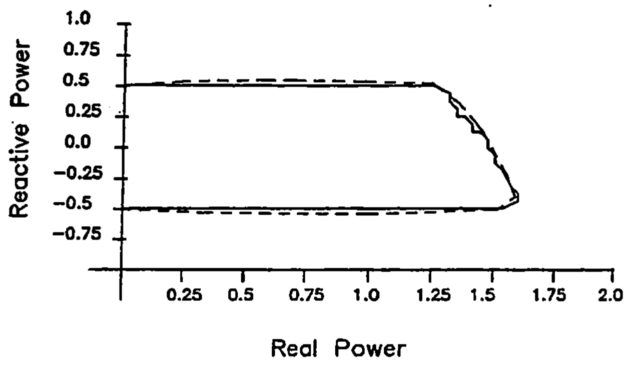
- 2-dimensional dynamic security contours of P,Q are obtained by fixing S at arbitrary values
- The NN generated contour compared well with the actual contour obtained using the oracle.

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 67

Figure 67

Notes Notes Notes

Sample of test results



©1992 - Applications to Power Systems/M. A. El-Sharkawi - 68

Figure 68

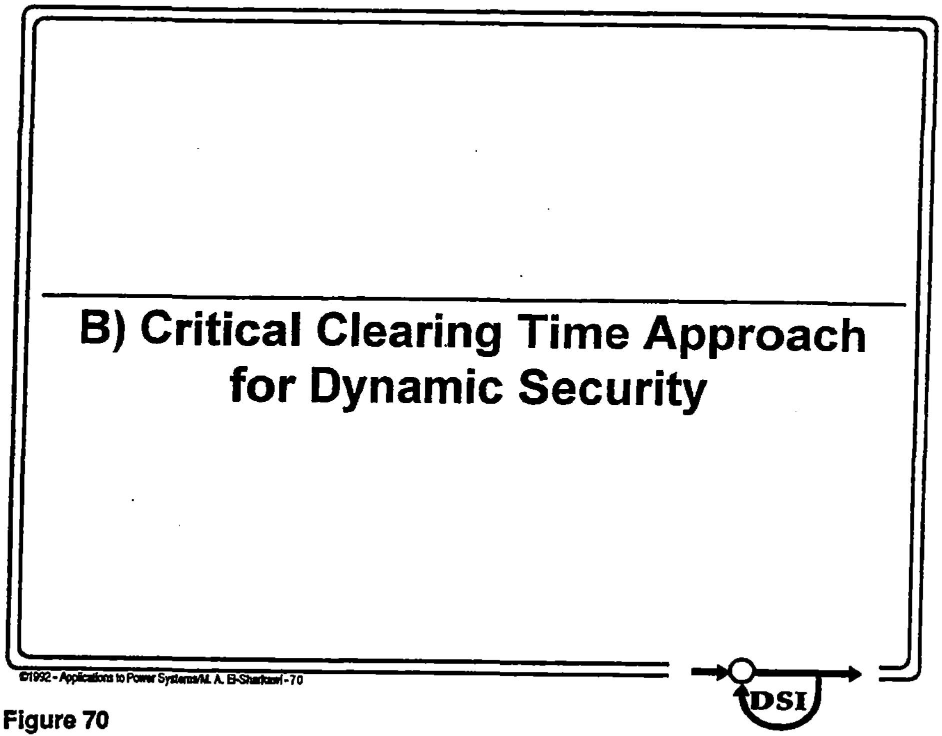
Training data for dynamic security assessment can be generated off-line by using an oracle. Training data can also include measurements of previous assessments. A multi-layer perceptron is trained using back-propagation to learn the dynamic security status with respect to a selected set of variables U within a defined operating space [16]. A test example of 9 bus, 3 generators was used to validate the method. For simplicity, 3 independent input variables were selected as inputs to the NN. They were the real and reactive outputs (P,Q) of one generator and complex power output ($S = \sqrt{P^2 + Q^2}$) of another generator. All other parameters were assumed to be constant. In the retrieving (testing) phase, 2-dimensional dynamic security contours of P,Q are obtained by fixing S at arbitrary values. The NN generated contour compared well with the actual contour obtained using the oracle [16].

Comments

- The dimensionality of the security contours is a function of the size of the system under investigation.
- In a high dimensional operational space where a combination of correlated and uncorrelated variables forms the input space, the development of a NN based system for assessing dynamic security is a challenging problem.

DSI

Figure 69



B) Critical Clearing Time Approach for Dynamic Security

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 70

Figure 70

Concept of CCT

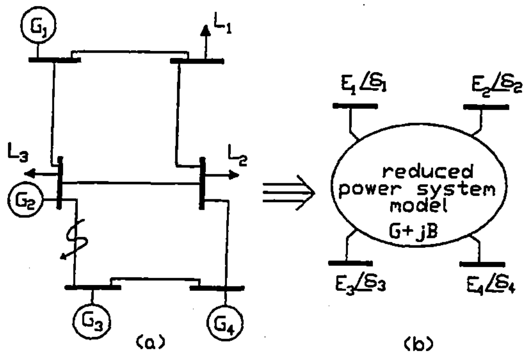


Figure 71

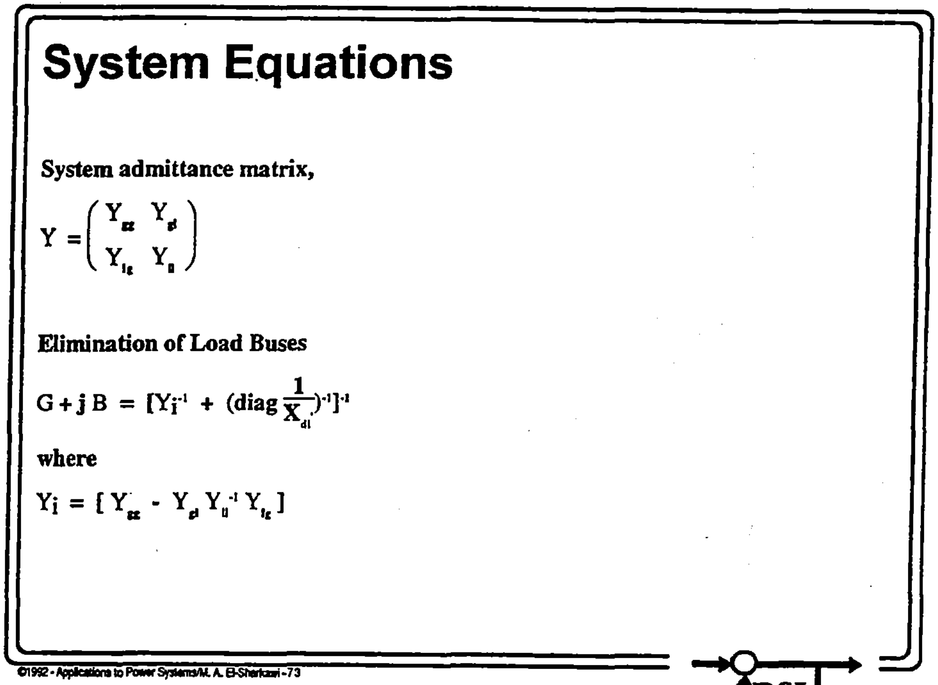
Figure (a) shows a small test power system. It has 6 buses with 4 generators and three loads. Since transient stability analysis is focused on the generator dynamics through a few cycles following the fault, certain simplifying assumptions can be made. All generators are replaced by the corresponding internal emfs (E) behind a transient reactance (X_d') Each load is replaced by a fixed admittance based on the pre-fault power flow. These assumptions are combined with generic circuit reduction techniques, to reduce the topology of the original power system to one that is shown in Figure (b). This reduced power system forms the basis for transient stability calculations.

Critical Clearing Time (CCT)

- **Fault Scenario**
  1. **Transmission line fault**
  2. **Faulted line is isolated**
  3. **After fault is cleared, line is reclosed**

- **If the fault is cleared and line is reclosed before the (CCT), the power system remains stable**

Figure 72



System Equations

System admittance matrix,

$$Y = \begin{pmatrix} Y_{gg} & Y_{gl} \\ Y_{lg} & Y_{ll} \end{pmatrix}$$

Elimination of Load Buses

$$G + jB = [Y_i^{-1} + (\mathrm{diag}\,\frac{1}{X_{di}})^{-1}]^{-1}$$

where

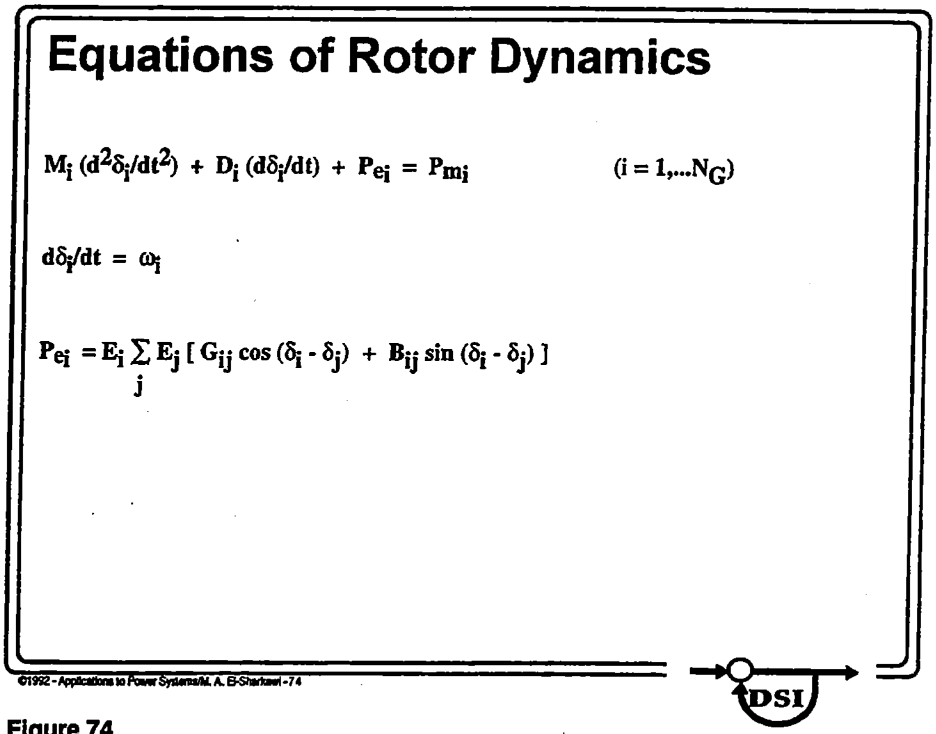
$$Y_i = [\,Y_{gg} - Y_{gl}\,Y_{ll}^{-1}\,Y_{lg}\,]$$

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 73

DSI

Figure 73

Subscripts g and l stand for generator and load buses respectively. The modified admittance matrix is corresponding to the reduced power system where all load buses are eliminated as shown in Figure b.



Equations of Rotor Dynamics

$$M_i\,(d^2\delta_i/dt^2) + D_i\,(d\delta_i/dt) + P_{ei} = P_{mi} \qquad (i = 1,\ldots N_G)$$

$$d\delta_i/dt = \omega_i$$

$$P_{ei} = E_i \sum_j E_j\,[\,G_{ij}\cos(\delta_i - \delta_j) + B_{ij}\sin(\delta_i - \delta_j)\,]$$

©1992 - Applications to Power Systems/M. A. El-Sharkawi -74

Figure 74

M_i,D_i - inertia and damping constants of the ith generator

P_{ei} - electrical power output of ith generator

P_{mi} - mechanical power input to the ith generator

E_i - equivalent field voltage behind the transient reactance X_d'

G_{ij},B_{ij} - real & imaginary parts of the reduced admittance matrix

δ_i - rotor angle of the ith generator relative to a synchronous reference

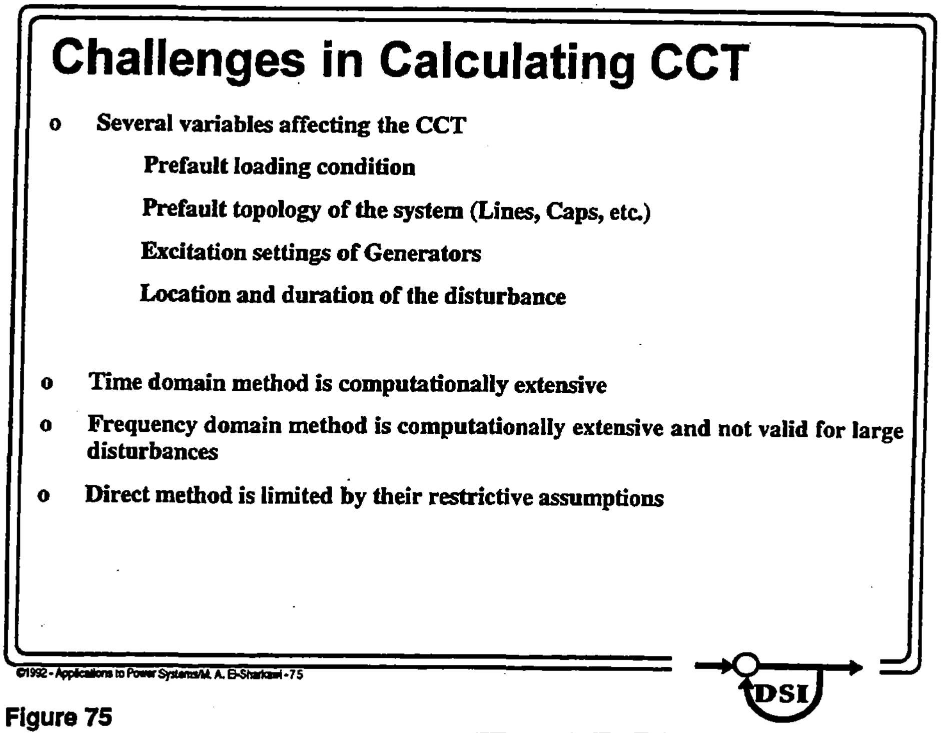
δ_i - angular velocity of ith generator relative to the same synchronous reference

N_G - number of generators in the system

The first two equations are the differential equations governing the rotor dynamics of the ith generator. The third equation gives the electrical power output of the ith generator calculated by applying Kirchoffs Laws.

Notes Notes Notes

The case study involves a transmission line fault. It is assumed that the line section is first isolated and then successfully reclosed. There exists a threshold parameter known as the *Critical Clearing Time* (CCT) where if the fault is cleared before this time, the power system remains stable. However, if the fault is cleared after the CCT, the power system is likely to become unstable. Hence, stability analysis may involve the calculation of the CCT for a given contingency.



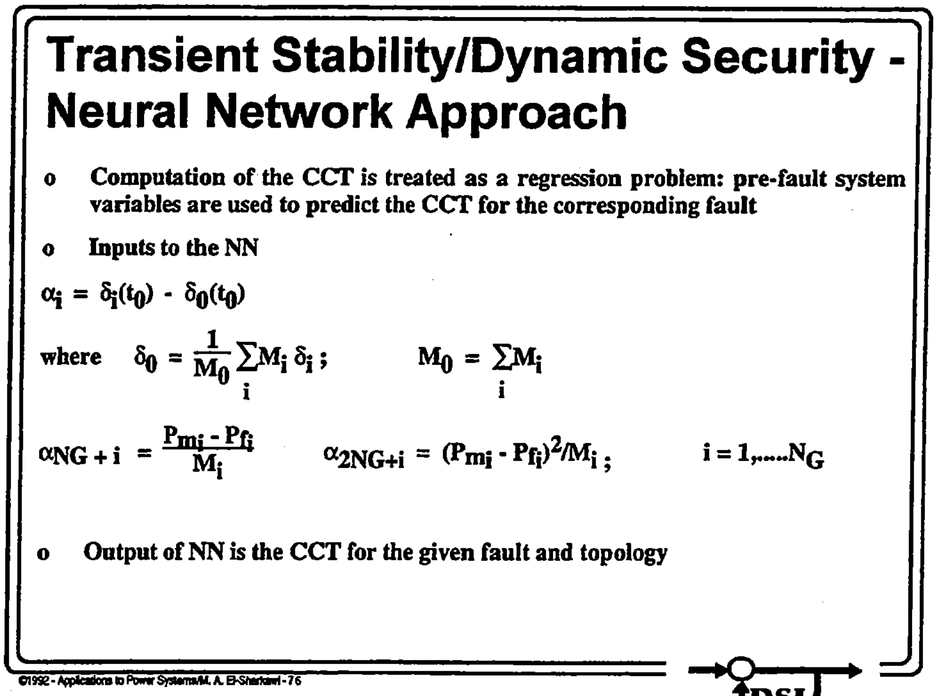
Challenges in Calculating CCT

- Several variables affecting the CCT
  - Prefault loading condition
  - Prefault topology of the system (Lines, Caps, etc.)
  - Excitation settings of Generators
  - Location and duration of the disturbance
- Time domain method is computationally extensive
- Frequency domain method is computationally extensive and not valid for large disturbances
- Direct method is limited by their restrictive assumptions

Figure 75

CCT is a complex function of pre-fault system conditions, disturbance structure and the post-fault conditions. There are two commonly used methods for calculating CCT, namely 1) Numerical integration and 2) Liapunov-type stability criteria [53]. The first method involves extensive time domain simulation of the power system while the scope of the second method is limited by its restrictive assumptions. Due to the many possible pre-fault operating conditions and types of faults, computational effort needed to assess the CCT for each of these scenarios is prohibitive.

Notes Notes Notes



Transient Stability/Dynamic Security - Neural Network Approach

- Computation of the CCT is treated as a regression problem: pre-fault system variables are used to predict the CCT for the corresponding fault
- Inputs to the NN

$$\alpha_i = \delta_i(t_0) - \delta_0(t_0)$$

where $\delta_0 = \frac{1}{M_0}\sum_i M_i \delta_i$; $\quad M_0 = \sum_i M_i$

$$\alpha_{NG+i} = \frac{P_{mi} - P_{fi}}{M_i} \qquad \alpha_{2NG+i} = (P_{mi} - P_{fi})^2/M_i; \qquad i = 1,....N_G$$

- Output of NN is the CCT for the given fault and topology

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 76

Figure 76

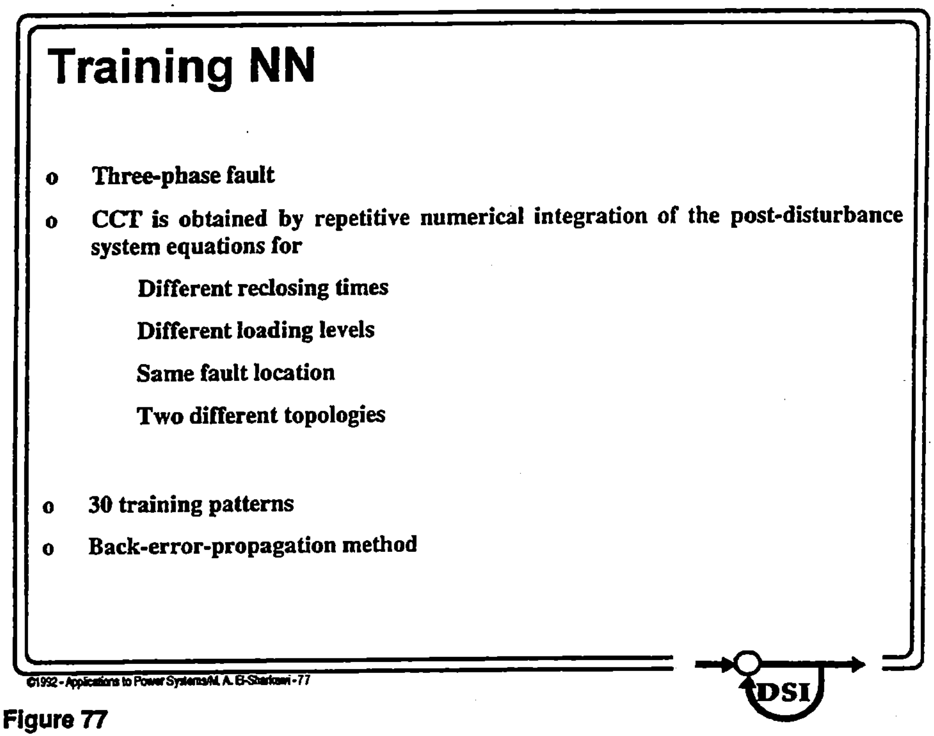
The estimation of CCT can be looked at as a regression problem where pre-fault system parameters are used to predict the CCT for the corresponding fault. A multi-layer perceptron was proposed to be trained using back-propagation to learn a set of input attributes and the corresponding CCTs for a specified fault under varying operating conditions [53].

The inputs to the NN (α_i) for a specified contingency are selected as given in the above equations.

M_0 is known as the center of mass while δ_0 is the center of angle. P_{fi} corresponds to the reduced electrical power output of the ith generator during fault initiation. This change from the steady state electrical power P_{ei} is brought about due to the change in network impedance caused by the fault and also due to the effect of the transient reactance of the generators.

The NN input quantity given by the second equation gives a measure of the rotor angle deviation at the instant of fault clearing. The input quantity described by the third equation is a measure of the individual acceleration energy of the generators of the system accumulated during the fault [53].

The output of the NN is the CCT corresponding to the given contingency under the described inputs.



Training NN

- Three-phase fault
- CCT is obtained by repetitive numerical integration of the post-disturbance system equations for

  Different reclosing times

  Different loading levels

  Same fault location

  Two different topologies

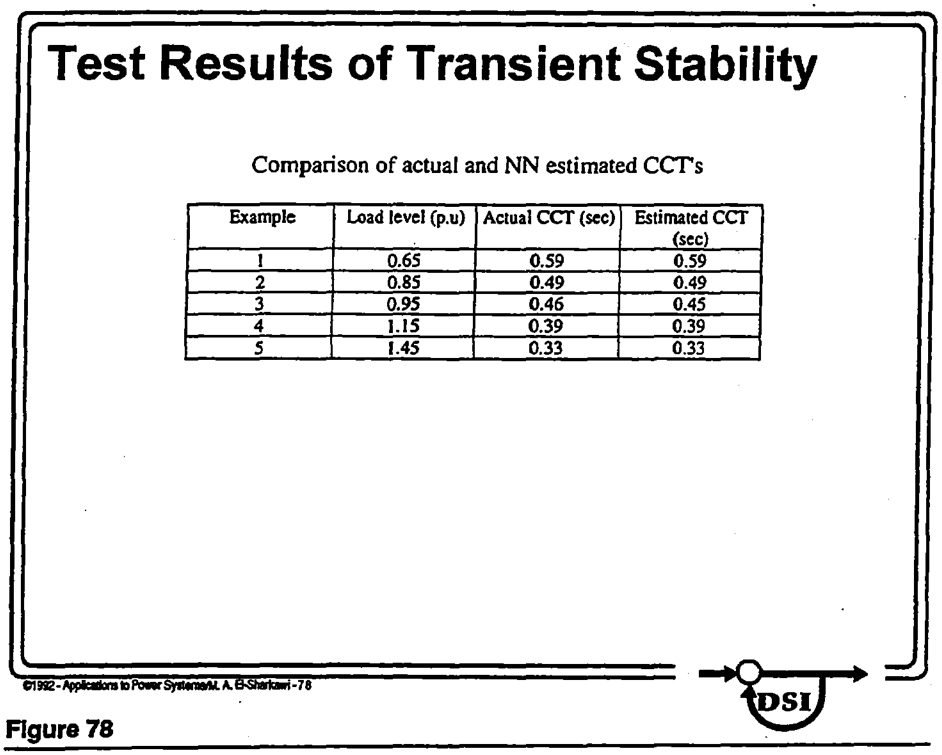
- 30 training patterns
- Back-error-propagation method

Figure 77

During generation of training data, CCT for the corresponding input quantities is obtained by repetitive numerical integration of the post-disturbance system equations using different reclosing times. The CCT would correspond to the maximum time for reclosure after the initial isolation of the line in order to maintain synchronous operation.

For a specific test of the algorithm, a 3-phase fault was simulated at location shown in figure (a). The CCT was calculated for the case where the fault was initially isolated by tripping the line and the system subsequently restored by reclosing the line. 30 training patterns were generated for a combination of different loading levels and two different base power system topologies. The trained NN was used to estimate the CCT for the same type of fault under varying load levels and varying topologies. The estimated CCT was compared to the analytical value calculated through numerical integration. Close comparison of results was reported.

Notes Notes Notes



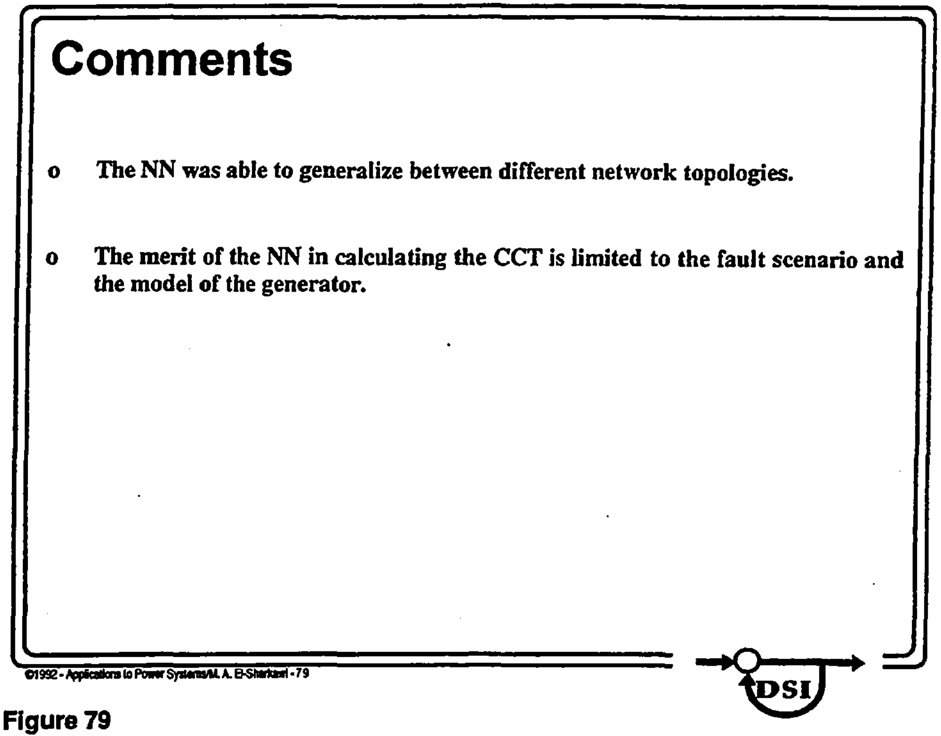
Test Results of Transient Stability

Comparison of actual and NN estimated CCT's

Example	Load level (p.u)	Actual CCT (sec)	Estimated CCT (sec)
1	0.65	0.59	0.59
2	0.85	0.49	0.49
3	0.95	0.46	0.45
4	1.15	0.39	0.39
5	1.45	0.33	0.33

Figure 78

The table shows a sample of the actual and NN estimated CCTs for a three-phase fault on one transmission line. The fault clearing strategy is line reclosing. The NN is trained and tested for different load levels which are obtained by perturbing all loads between 0.6 and 2.0 around a selected nominal value.



Comments

- The NN was able to generalize between different network topologies.
- The merit of the NN in calculating the CCT is limited to the fault scenario and the model of the generator.

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 79

DSI

Figure 79

The ability of a NN to generalize between different network topologies was observed. This adaptability was facilitated by providing training data corresponding to couple of different base topologies. This is a key idea that could be applied to training NNs for problems with time varying power system topologies.

So far, the merit of the NN in calculating the CCT is limited to the above mentioned fault scenario and the restrictive second order model of the generator. Simulations are also restricted to simple 3-phase line faults. The ability of the NN to predict CCT under more complicated fault scenarios is not clear. The training data should be produced by using a higher order generator model to include other transients caused by the presence of damper windings and excitation systems.

Notes Notes Notes

D) Time Domain Transients for Dynamic Security

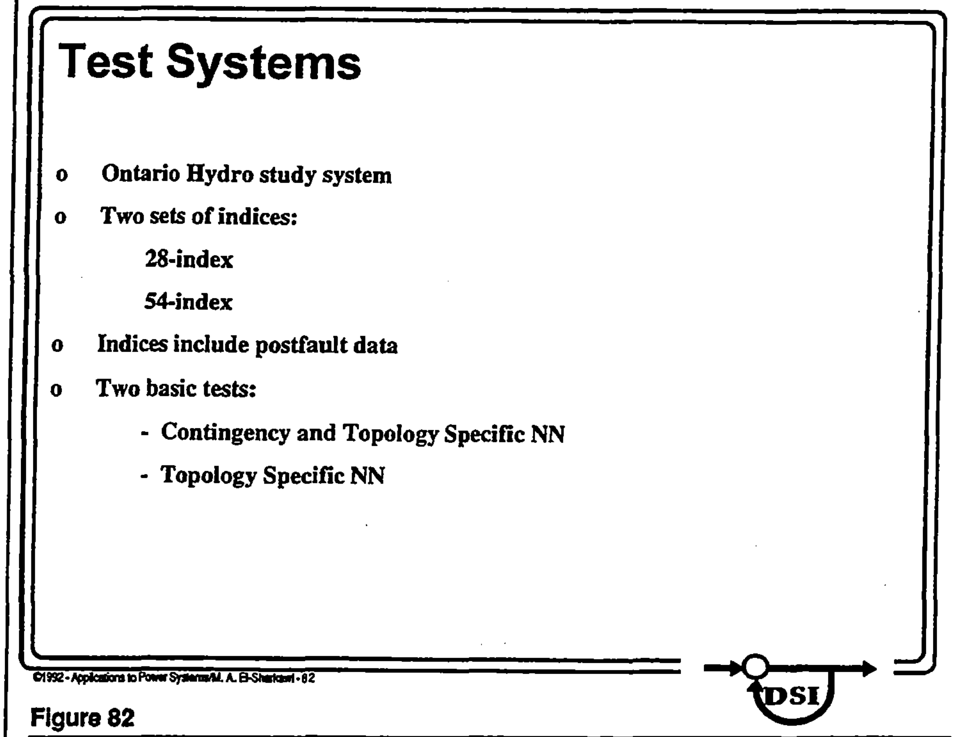
Figure 80

Concept of Time Domain Transients

- Off-line transient analysis
- Selected indices are computed
  - Weighted and aggregated currents in transmission lines
  - Voltages at specific buses
- Indices are selected according to operators' recommendations
- One or two cycles of simulations are used
- Decision on system stability is made by experts

Figure 81

Notes Notes Notes



Test Systems

- Ontario Hydro study system
- Two sets of indices:
  - 28-index
  - 54-index
- Indices include postfault data
- Two basic tests:
  - Contingency and Topology Specific NN
  - Topology Specific NN

Figure 82

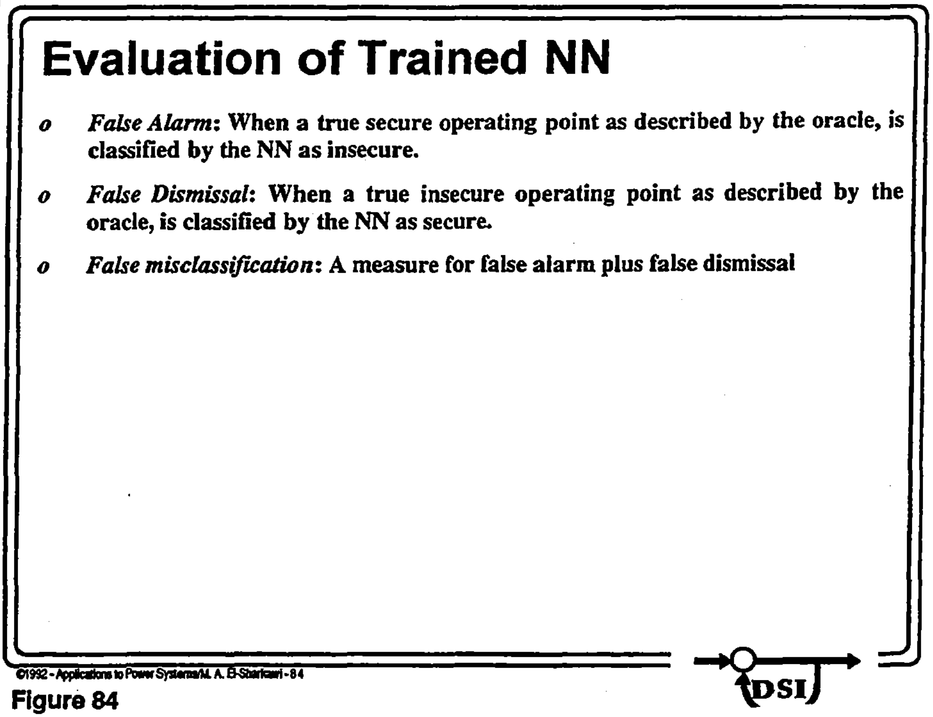
Contingency and Topology Specific NN

- One NN per contingency
- 63 patterns for each contingency
- Each pattern is composed of 28 indices
- All patterns are normalized between 0 and 1
- Patterns vectors are randomly shuffled
- Patterns are split into two sets: training and testing
- Training set has 50 patterns
- Testing set has 13 patterns

DSI

Figure 83

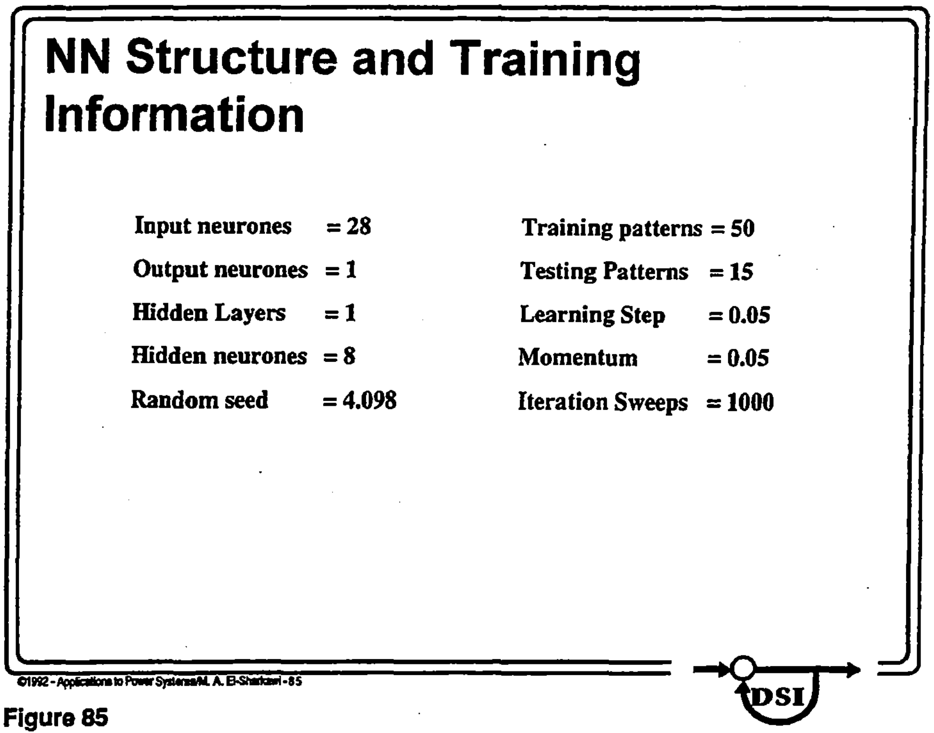
Notes Notes Notes



Evaluation of Trained NN

- *False Alarm*: **When a true secure operating point as described by the oracle, is classified by the NN as insecure.**
- *False Dismissal*: **When a true insecure operating point as described by the oracle, is classified by the NN as secure.**
- *False misclassification*: **A measure for false alarm plus false dismissal**

Figure 84

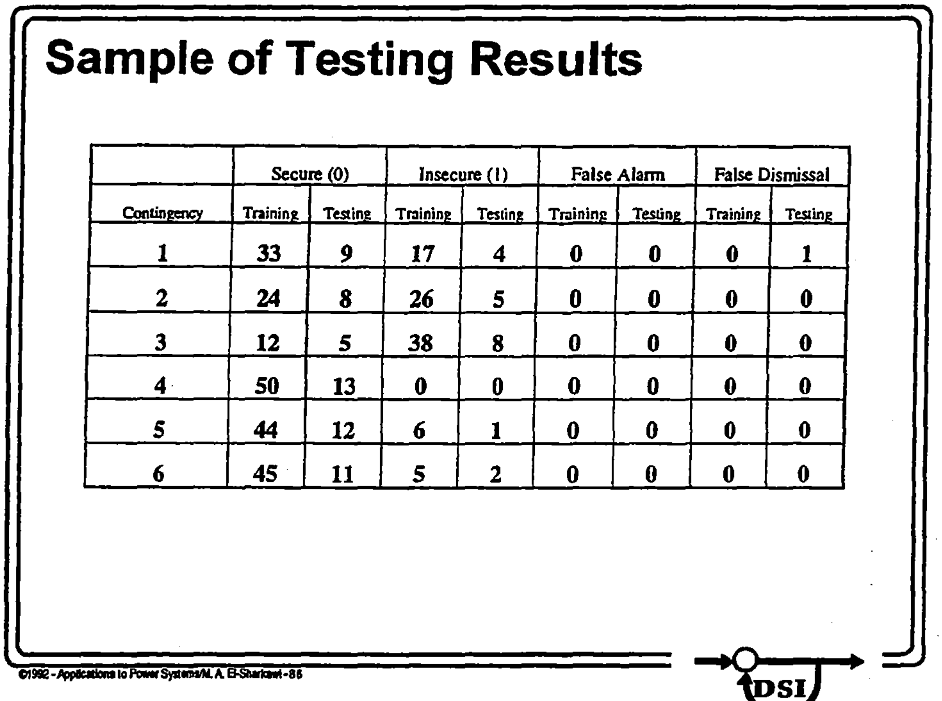


NN Structure and Training Information

Input neurones	= 28	Training patterns	= 50
Output neurones	= 1	Testing Patterns	= 15
Hidden Layers	= 1	Learning Step	= 0.05
Hidden neurones	= 8	Momentum	= 0.05
Random seed	= 4.098	Iteration Sweeps	= 1000

Figure 85

Notes Notes Notes



Sample of Testing Results

	Secure (0)		Insecure (1)		False Alarm		False Dismissal	
Contingency	Training	Testing	Training	Testing	Training	Testing	Training	Testing
1	33	9	17	4	0	0	0	1
2	24	8	26	5	0	0	0	0
3	12	5	38	8	0	0	0	0
4	50	13	0	0	0	0	0	0
5	44	12	6	1	0	0	0	0
6	45	11	5	2	0	0	0	0

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 86

DSI

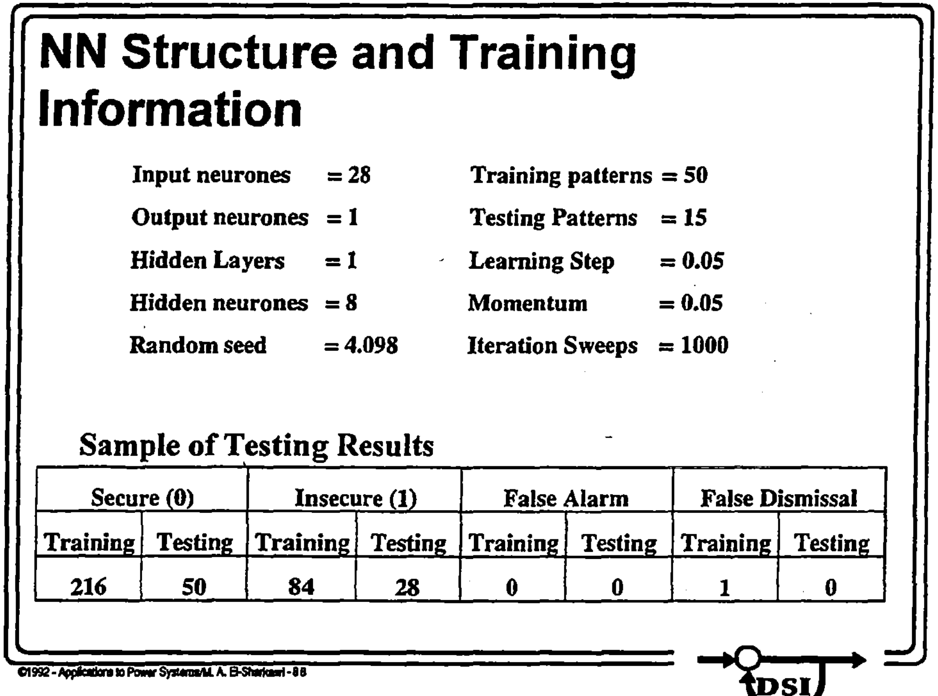
Figure 86

Notes Notes Notes

Topology Specific NN

- One NN for 6 contingencies
- 378 patterns
- Each pattern is composed of 28 indices
- All patterns are normalized between 0 and 1
- Patterns vectors are randomly shuffled
- Patterns are split into two sets: Training and testing
- Training set has 300 patterns
- Testing set has 78 patterns

Figure 87



NN Structure and Training Information

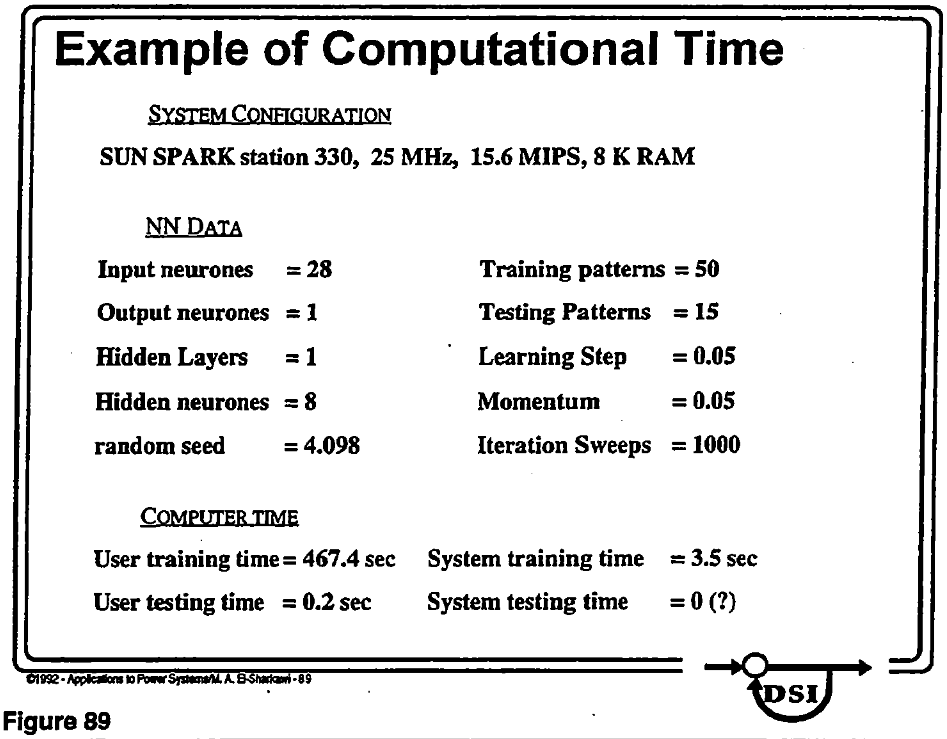
Input neurones	= 28	Training patterns	= 50
Output neurones	= 1	Testing Patterns	= 15
Hidden Layers	= 1	Learning Step	= 0.05
Hidden neurones	= 8	Momentum	= 0.05
Random seed	= 4.098	Iteration Sweeps	= 1000

Sample of Testing Results

Secure (0)		Insecure (1)		False Alarm		False Dismissal	
Training	Testing	Training	Testing	Training	Testing	Training	Testing
216	50	84	28	0	0	1	0

DSI

Figure 88



Example of Computational Time

System Configuration

SUN SPARK station 330, 25 MHz, 15.6 MIPS, 8 K RAM

NN Data

Input neurones	= 28	Training patterns	= 50
Output neurones	= 1	Testing Patterns	= 15
Hidden Layers	= 1	Learning Step	= 0.05
Hidden neurones	= 8	Momentum	= 0.05
random seed	= 4.098	Iteration Sweeps	= 1000

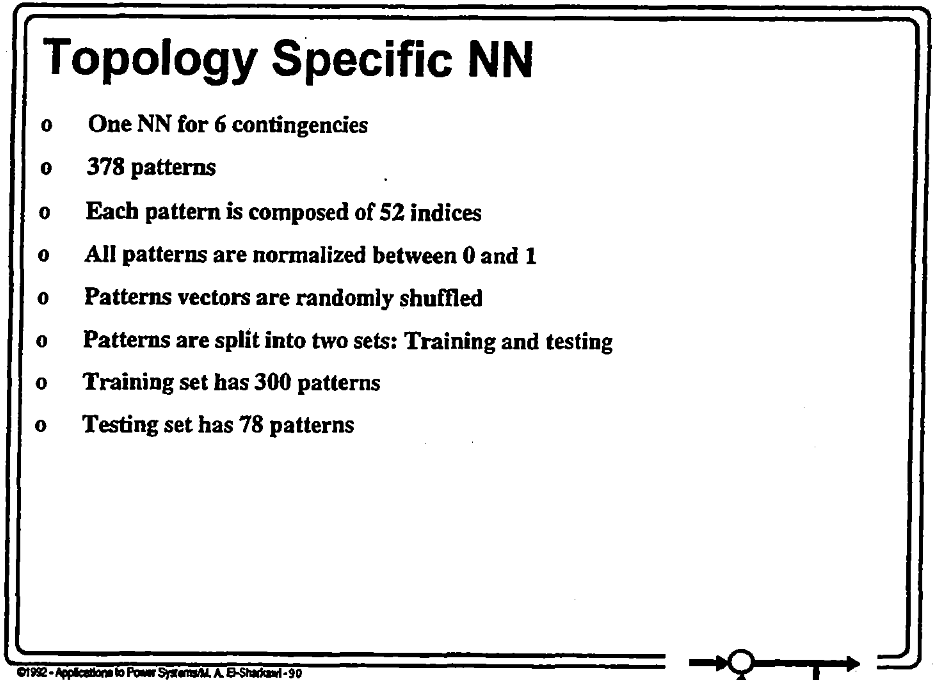
Computer time

User training time	= 467.4 sec	System training time	= 3.5 sec
User testing time	= 0.2 sec	System testing time	= 0 (?)

DSI

Figure 89

Notes Notes Notes



Topology Specific NN

- One NN for 6 contingencies
- 378 patterns
- Each pattern is composed of 52 indices
- All patterns are normalized between 0 and 1
- Patterns vectors are randomly shuffled
- Patterns are split into two sets: Training and testing
- Training set has 300 patterns
- Testing set has 78 patterns

Figure 90

NN Structure and Training Information

Input neurones	= 52	Training patterns	= 300
Output neurones	= 1	Testing Patterns	= 78
Hidden Layers	= 1	Learning Step	= 0.05
Hidden neurones	= 3	Momentum	= 0.05
Random seed	= 4.098	Iteration Sweeps	= 2320

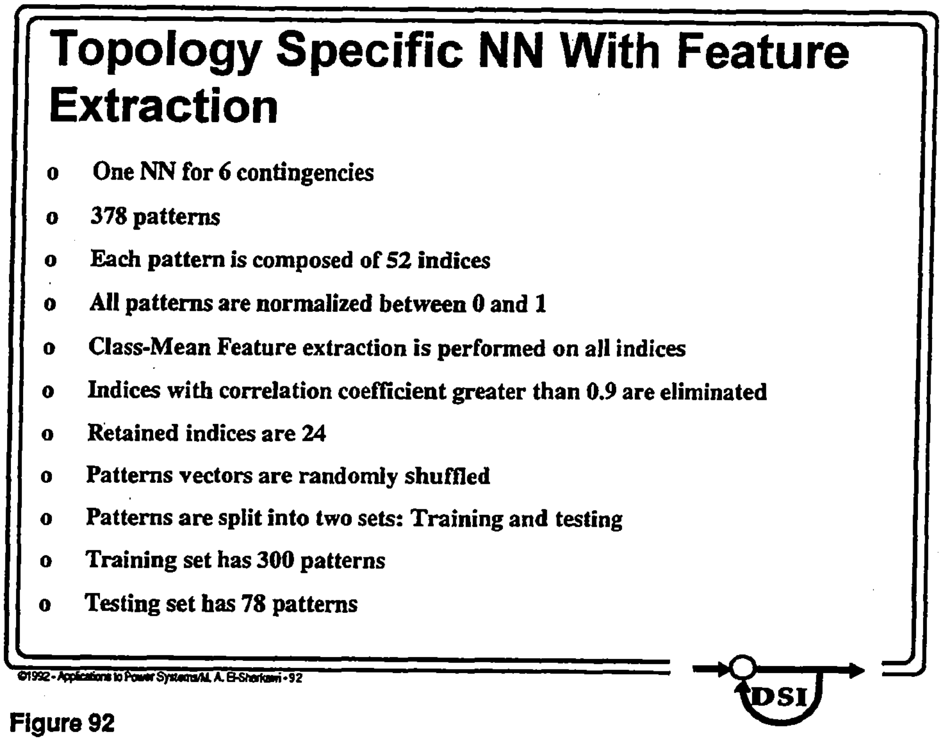
CPU Time = 57.6 s

Sample of Testing Results

Secure (0)		Insecure (1)		False Alarm		False Dismissal	
Training	Testing	Training	Testing	Training	Testing	Training	Testing
210	56	90	22	2	0	2	0

Figure 91

Notes Notes Notes

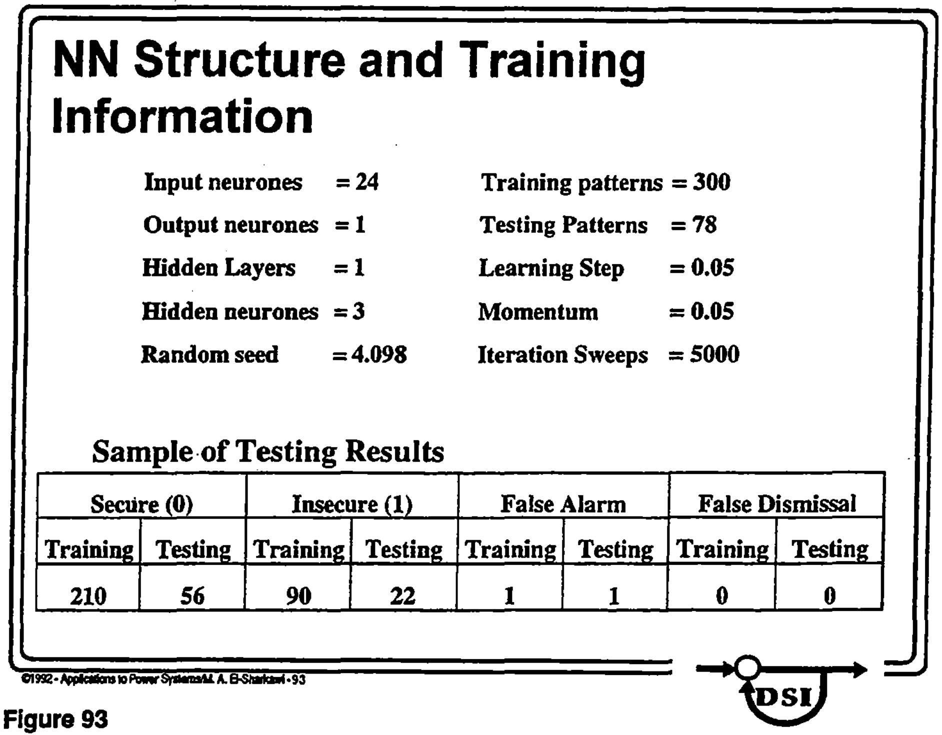


Topology Specific NN With Feature Extraction

- One NN for 6 contingencies
- 378 patterns
- Each pattern is composed of 52 indices
- All patterns are normalized between 0 and 1
- Class-Mean Feature extraction is performed on all indices
- Indices with correlation coefficient greater than 0.9 are eliminated
- Retained indices are 24
- Patterns vectors are randomly shuffled
- Patterns are split into two sets: Training and testing
- Training set has 300 patterns
- Testing set has 78 patterns

DSI

Figure 92



NN Structure and Training Information

Input neurones	= 24	Training patterns	= 300
Output neurones	= 1	Testing Patterns	= 78
Hidden Layers	= 1	Learning Step	= 0.05
Hidden neurones	= 3	Momentum	= 0.05
Random seed	= 4.098	Iteration Sweeps	= 5000

Sample of Testing Results

Secure (0)		Insecure (1)		False Alarm		False Dismissal	
Training	Testing	Training	Testing	Training	Testing	Training	Testing
210	56	90	22	1	1	0	0

DSI

Figure 93

Notes Notes Notes

Notes Notes Notes

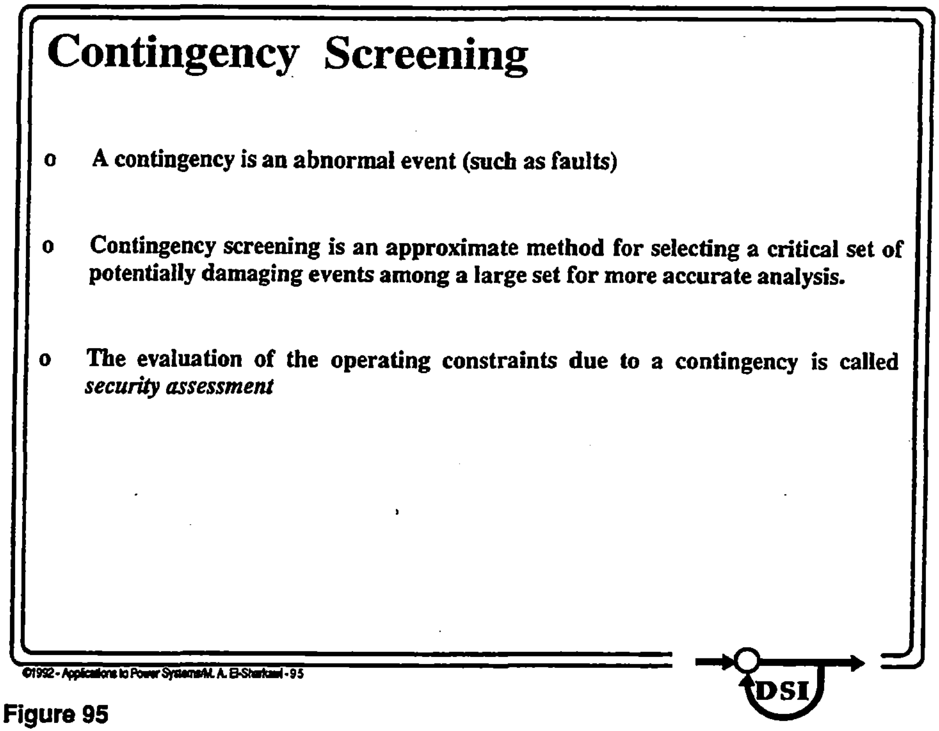
Contingency Screening

DSI

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 94

Figure 94

Notes Notes Notes



Contingency Screening

- A contingency is an abnormal event (such as faults)
- Contingency screening is an approximate method for selecting a critical set of potentially damaging events among a large set for more accurate analysis.
- The evaluation of the operating constraints due to a contingency is called *security assessment*

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 95

Figure 95

A contingency in a power system, is an abnormal event (such as faults) which could be potentially damaging to power system components. Contingency screening is a relatively fast and approximate method of identifying whether a contingency may result in a violation of any of the operating constraints of the power system. The evaluation of the operating constraints due to a contingency is called *security assessment.* Contingency screening helps select a critical set of potentially damaging events for more accurate analysis.

Contingency selection, in its simplest form, is dealing with forming a list of contingencies which may result in steady state voltage or thermal limits violations in the post contingency power flow condition.

Problem Formulation

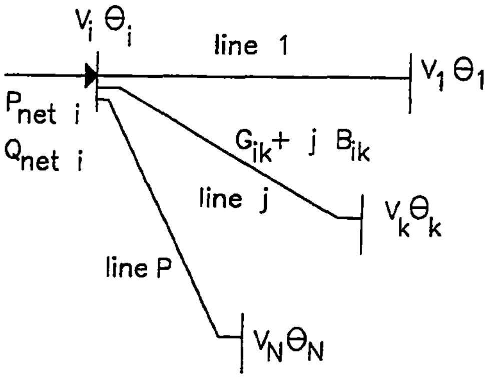


Figure 96

Notes Notes Notes

Basic Equations

$$P_{net\,i} = V_i \sum_k V_k \,[G_{ik} \cos \theta_{ik} + B_{ik} \sin \theta_{ik}] \qquad (1)$$

$$Q_{net\,i} = V_i \sum_k V_k \,[G_{ik} \sin \theta_{ik} - B_{ik} \cos \theta_{ik}] \qquad (2)$$

$$P_{line\,j} = G_{ik} (V_i^2 - V_i V_k \cos \theta_{ik}) - B_{ik} V_i V_k \sin \theta_{ik} \qquad (3)$$

$$Q_{line\,j} = - B_{ik} (V_i^2 - V_i V_k \cos \theta_{ik}) - G_{ik} V_i V_k \sin \theta_{ik} \qquad (4)$$

$$S_{line\,j} = \sqrt{P_{line}{}^2{}_j + P_{line}{}^2{}_j} \qquad (5)$$

$\theta_{ik} = \theta_i - \theta_k$ and $Y = G + jB$

Figure 97

$P_{net\,i}$, $Q_{net\,i}$ are the net real and reactive injections at ith bus. The voltage magnitudes (V_i) obtained by solving equations (1) and (2) and line flows ($S_{line\,j}$) obtained from equation (5) constitute the so called *security variables*, which are the variables that decide the status of the system security. Any magnitude violation of these variables will result in an insecure system.

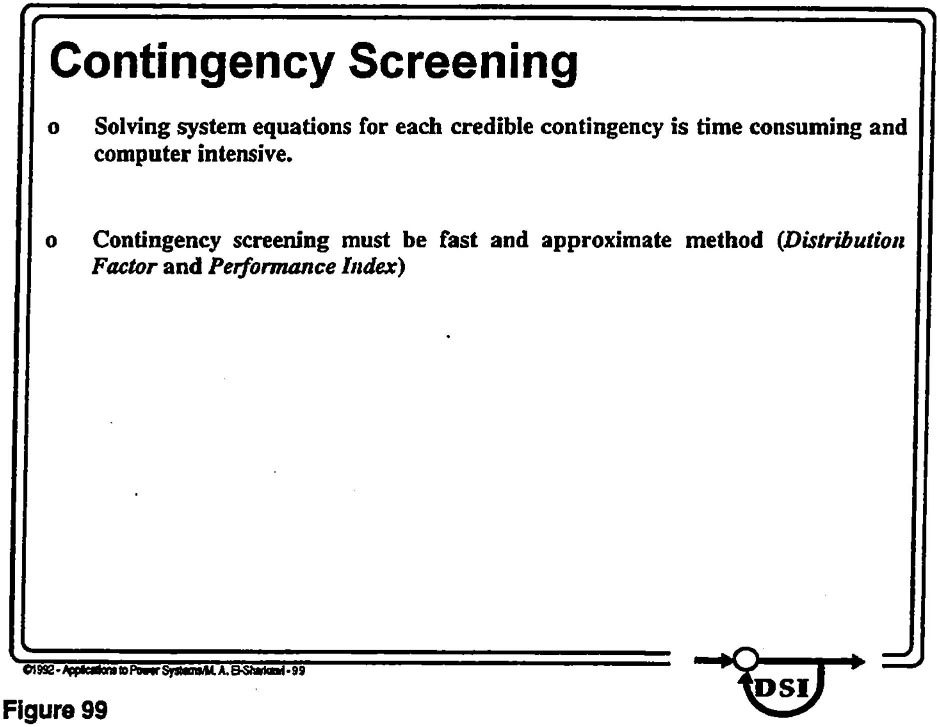
Post-contingency security limits

$$\left.\begin{array}{l} V_U \geq V(\lambda) \geq V_L \\ S_{\max} \geq |S(\lambda)| \end{array}\right\} Z_U \geq Z(\lambda) \geq Z_L$$

- $z_i(\lambda)$ denotes post contingency value of the i^{th} security variable corresponding to λ^{th} contingency.

- If all inequalities are satisfied the system is labelled secure under the λ^{th} contingency.

Figure 98



Contingency Screening

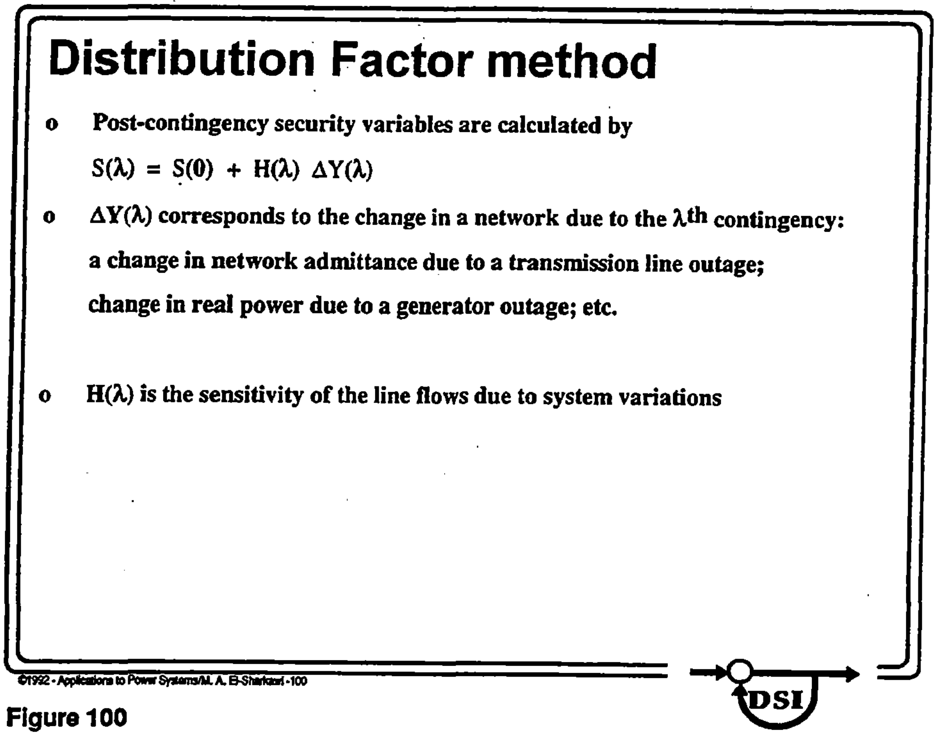
- Solving system equations for each credible contingency is time consuming and computer intensive.
- Contingency screening must be fast and approximate method (*Distribution Factor* and *Performance Index*)

Figure 99

$z_i(\lambda)$ denotes the post contingency value of the i^{th} security variable corresponding to λ^{th} contingency. If all the above inequalities are satisfied the system is labelled as secure under the λ^{th} contingency.

Solving equations (1) through (5) for each credible contingency is time consuming and often computer intensive. To obtain a fast and approximate method for selecting key contingencies is known as *Contingency screening*. Contingency screening can be performed by several methods, among them are the *Distribution Factor* and the *Performance Index*.

Notes Notes Notes



Distribution Factor method

- o Post-contingency security variables are calculated by

  $S(\lambda) = S(0) + H(\lambda)\ \Delta Y(\lambda)$

- o $\Delta Y(\lambda)$ corresponds to the change in a network due to the λ^{th} contingency:

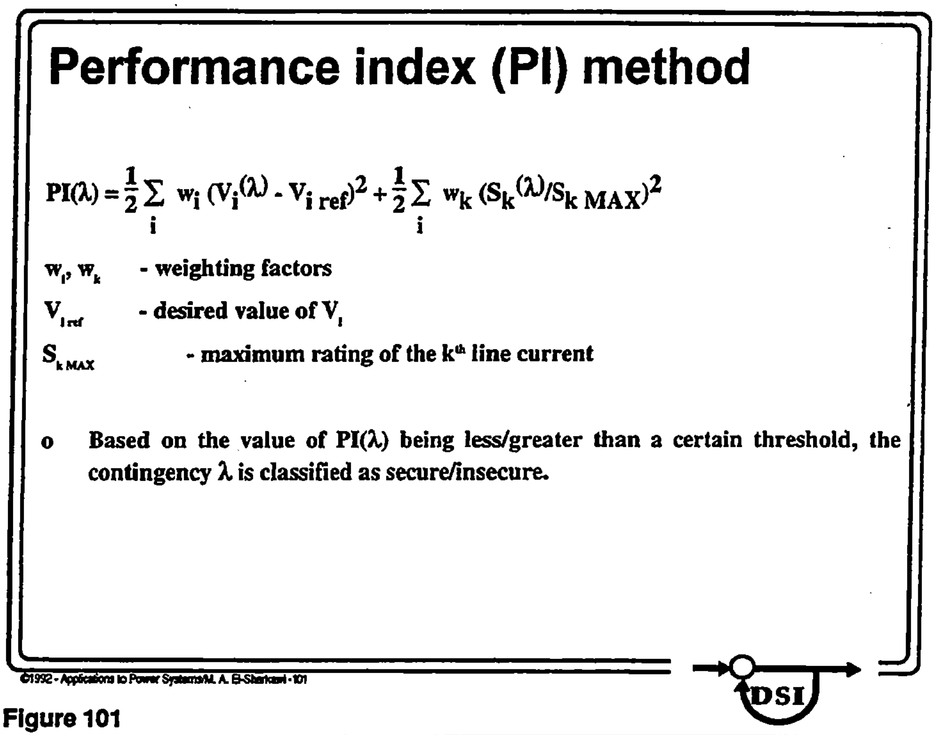
  a change in network admittance due to a transmission line outage;

  change in real power due to a generator outage; etc.

- o $H(\lambda)$ is the sensitivity of the line flows due to system variations

Figure 100

where $\Delta Y(\lambda)$ corresponds to the change in a network due to the λ^{th} contingency. This could be either a change in network admittance due to a transmission line outage or the change in real power due to a generator outage. $H(\lambda)$ is known as the transfer matrix whose elements are a set of factors which represent the sensitivity of the line flows to the above variations. Therefore, these partial derivatives can either be line outage distribution factors or generation shift factors corresponding to the type of the λ^{th} contingency.



Performance index (PI) method

$$PI(\lambda) = \frac{1}{2}\sum_{i} w_i \left(V_i^{(\lambda)} - V_{i\,ref}\right)^2 + \frac{1}{2}\sum_{i} w_k \left(S_k^{(\lambda)}/S_{k\,MAX}\right)^2$$

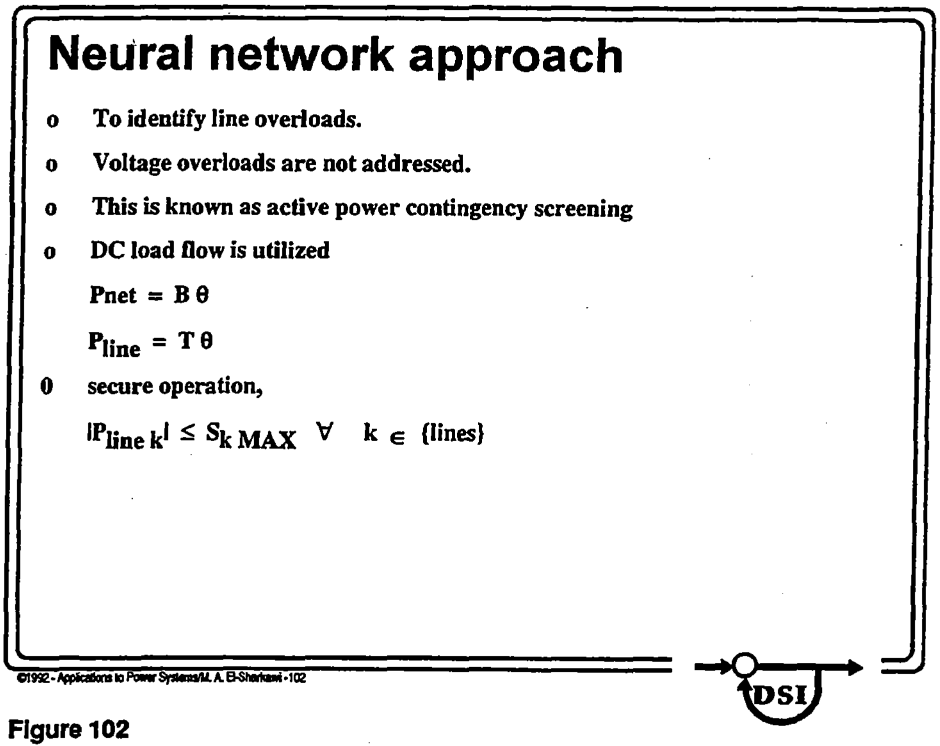
w_i, w_k - weighting factors

$V_{i\,ref}$ - desired value of V_i

$S_{k\,MAX}$ - maximum rating of the kth line current

- o Based on the value of PI(λ) being less/greater than a certain threshold, the contingency λ is classified as secure/insecure.

Figure 101



Neural network approach

- o To identify line overloads.
- o Voltage overloads are not addressed.
- o This is known as active power contingency screening
- o DC load flow is utilized

  $P_{net} = B\,\theta$

  $P_{line} = T\,\theta$

- 0 secure operation,

  $|P_{line\,k}| \leq S_{k\,MAX} \quad \forall \quad k \in \{lines\}$

Figure 102

NN approach is proposed for contingency screening [56]. It is based on identifying the contingent branch overloads. The question of contingent voltages is not addressed in this study. This is known as active power contingency screening which is based on the DC load flow concept: All voltage magnitudes V_i are equal to unity and that all angles θ_i are small ($\sin \theta_i = \theta_i$).

Neural Network Structure

- A collection of NNs are trained
- Each NN is dedicated to a specific line outage.
- Inputs to NN:

  B_{ij} ∀ i, j ∈ {buses} (post-contingency system)

  $P_{net\ i}$ ∀ i ∈ {buses},
- Outputs of NN:

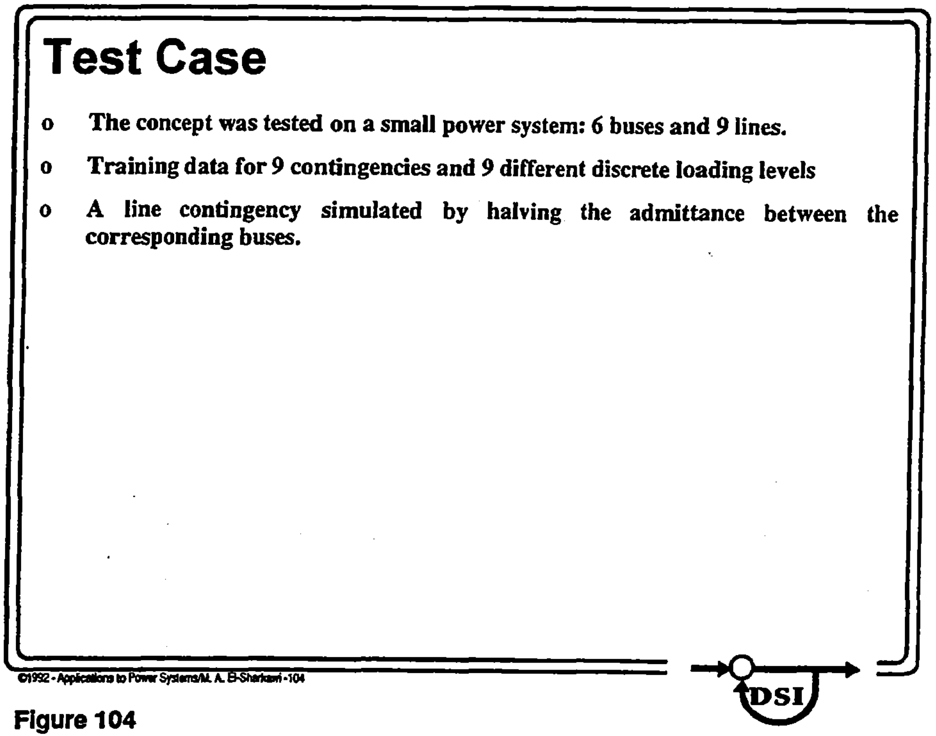
  $P_{line\ k}$ ∀ k ∈ {lines}

  binary security flag ∈ {0, 1}

Figure 103

Notes Notes Notes

Notes Notes Notes

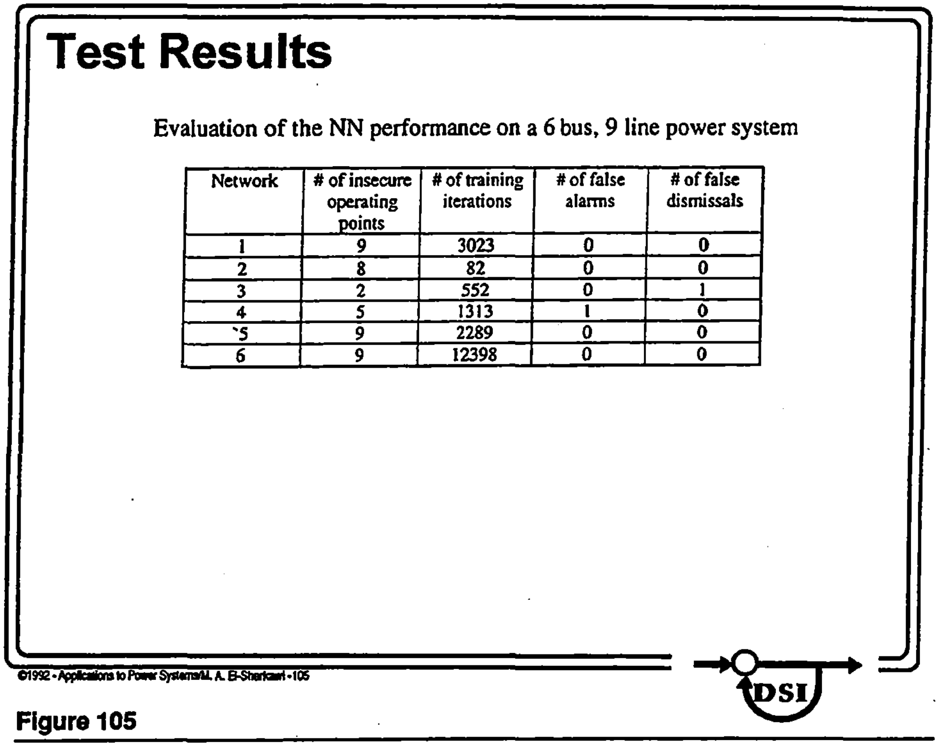


Test Case

- The concept was tested on a small power system: 6 buses and 9 lines.
- Training data for 9 contingencies and 9 different discrete loading levels
- A line contingency simulated by halving the admittance between the corresponding buses.

Figure 104

A collection of NNs are trained where each NN is dedicated to a specific line outage. The concept was tested on a small power system with 6 buses and 9 lines. Training data was generated for 9 contingencies and 9 different discrete loading levels giving 81 different patterns. Only line contingencies were considered. A line contingency was simulated by halving the admittance between the corresponding buses. Each contingency was handled by a separate NN.



Test Results

Evaluation of the NN performance on a 6 bus, 9 line power system

Network	# of insecure operating points	# of training iterations	# of false alarms	# of false dismissals
1	9	3023	0	0
2	8	82	0	0
3	2	552	0	1
4	5	1313	1	0
5	9	2289	0	0
6	9	12398	0	0

Figure 105

Each NN investigates the thermal violations under a single line contingency. Performance of 6 of the 9 NNs are given in the table. Nine different load levels are used. Five are used for training and all 9 patterns are used for testing. The second column indicates the number of insecure operating points out of the selected 9 load levels for any given line contingency.

Comments

- o NN based contingency screening method is effective for a small power system.
- o The number of input nodes is equal to twice the number of buses plus the number of lines.
- o For a larger power system, the input variables can be excessively large.

DSI

Figure 106

Harmonic Evaluation and Identification

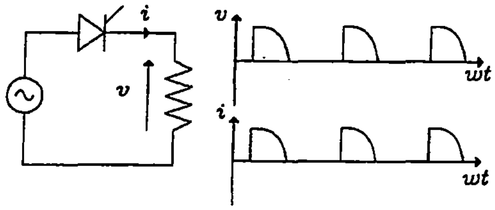
DSI

Figure 107

Notes Notes Notes

Notes Notes Notes

Harmonic Evaluation and Identification

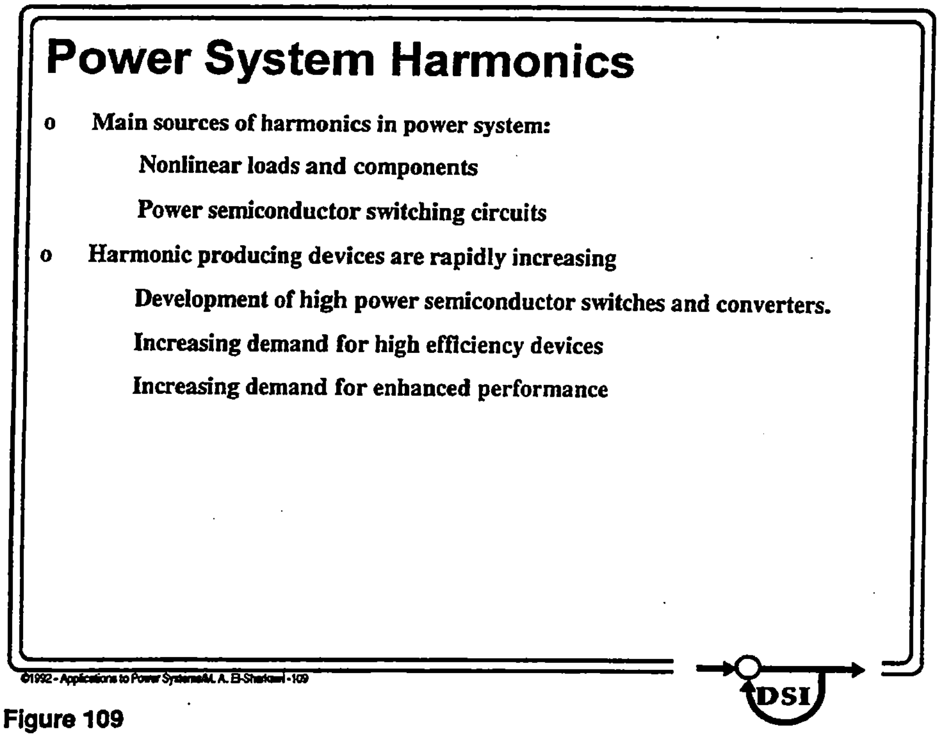


Phase controlled rectifier

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 108

DSI

Figure 108



Power System Harmonics

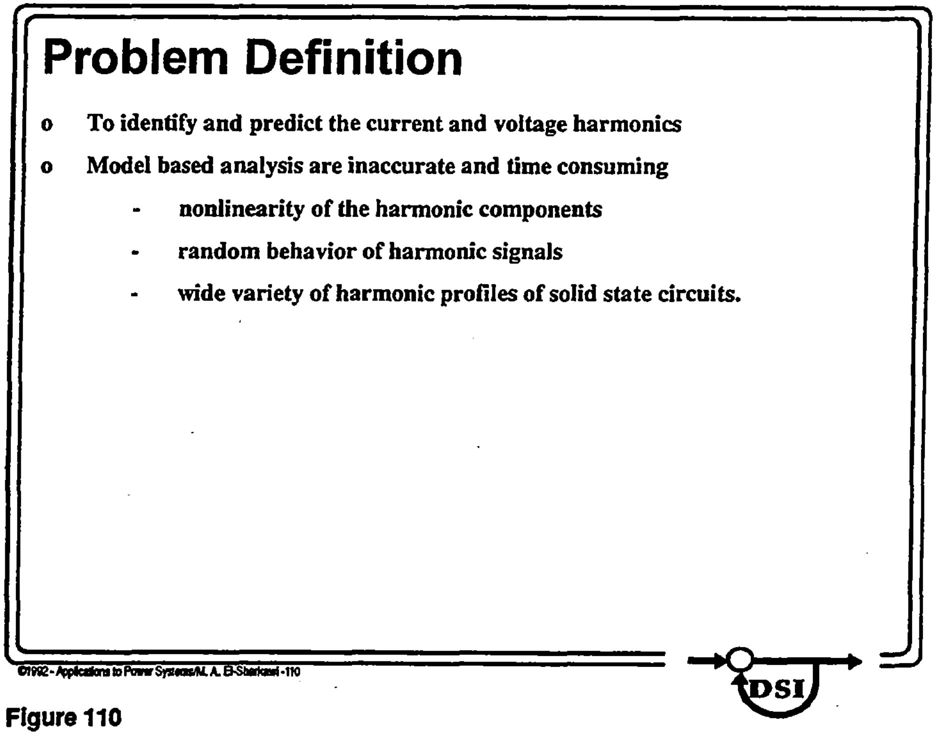
- **Main sources of harmonics in power system:**
  - **Nonlinear loads and components**
  - **Power semiconductor switching circuits**
- **Harmonic producing devices are rapidly increasing**
  - **Development of high power semiconductor switches and converters.**
  - **Increasing demand for high efficiency devices**
  - **Increasing demand for enhanced performance**

Figure 109

Notes Notes Notes

Nonlinear loads and other harmonic producing loads have existed in power systems for many years. Today, the number of harmonic producing devices is rapidly rising due to the development of high power semiconductor switches and converters.

The figure indicates a simple phased controlled rectifier connected to a resistive load. The figure shows the load voltage and current. This nonsinusoidal load current, unless filtered, will be drawn from the power system. If a large number of such solid state devices and circuits are used, the nonsinusoidal current will give rise to harmonic voltage drops across system components, thereby distorting the voltage wave form of the system. This can cause potentially damaging problems to the power system such as misoperation of protective relays, overheating of capacitor banks, increased losses in transmission systems, insulation failure in cables, increased losses in transformers and noise in communication circuits.



Problem Definition

- o To identify and predict the current and voltage harmonics
- o Model based analysis are inaccurate and time consuming
  - nonlinearity of the harmonic components
  - random behavior of harmonic signals
  - wide variety of harmonic profiles of solid state circuits.

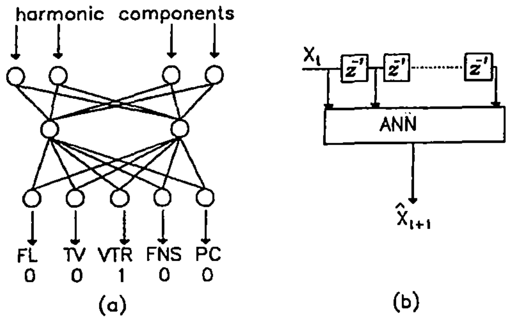
©1992 - Applications to Power Systems/M. A. El-Sharkawi - 110

DSI

Figure 110

The objective is to analyze and predict the behavior of current and voltage harmonics so that appropriate action could be taken to reduce their adverse effects. So far, model based analysis has been inaccurate and time consuming due to the nonlinearity of the harmonic components, the random behavior of harmonic signals and the wide variety of harmonic profiles of all solid state circuits.

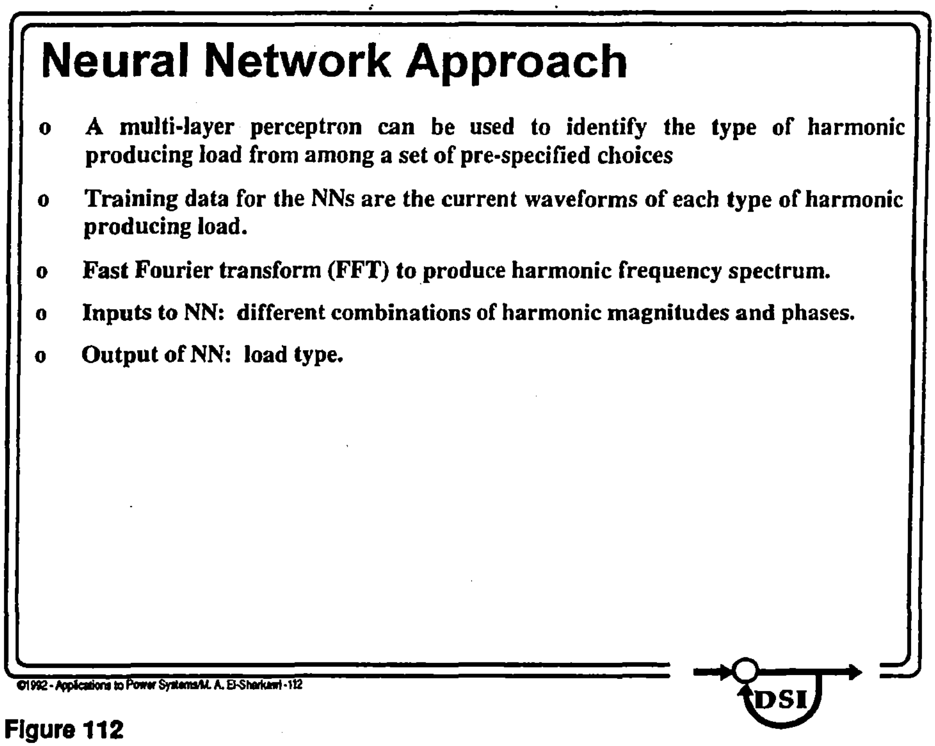
NN Structure For Harmonic Evaluation and Identification



a) Identification of harmonic loads
b) Prediction of Harmonics
From [57] courtesy of IEEE, (C) IEEE,1989

Figure 111

The figure shows the structure of the NN used to learn the harmonic/ load relationship in the example given in reference [57]. The NN input are chosen among 31 harmonic magnitudes and phases. The output is one of 5 load groups, namely Personal Computer (PC), Television Set (TV), Video Tape Recorder (VTR), Fans (FNS) and Fluorescent Lamps (FL). Three different test cases are studied where a NN is trained under each case with different combination of inputs.

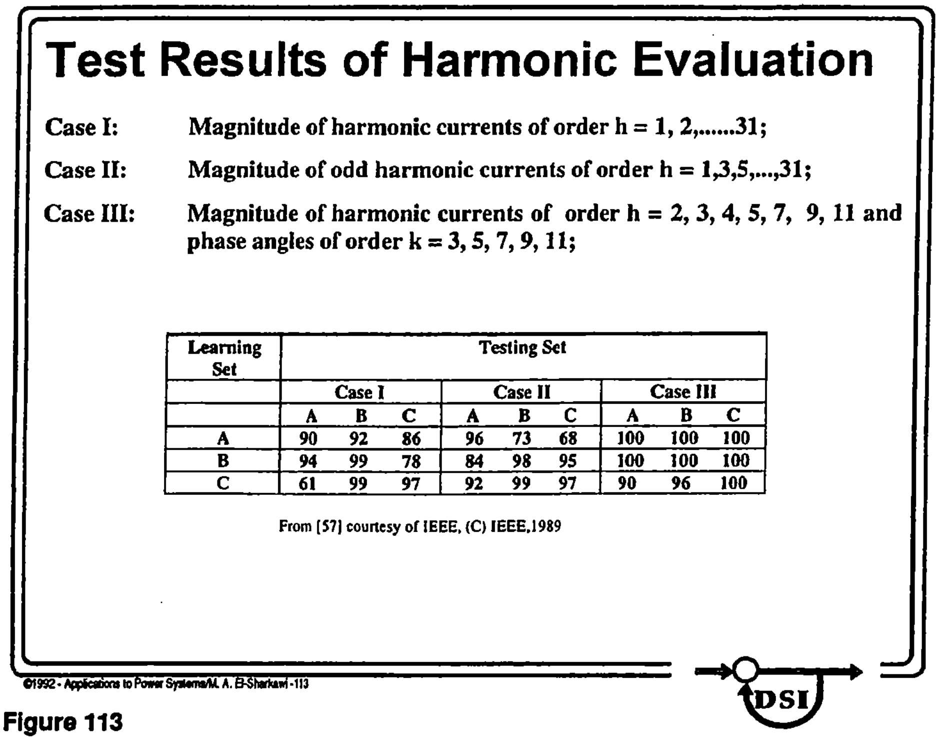


Neural Network Approach

- A multi-layer perceptron can be used to identify the type of harmonic producing load from among a set of pre-specified choices
- Training data for the NNs are the current waveforms of each type of harmonic producing load.
- Fast Fourier transform (FFT) to produce harmonic frequency spectrum.
- Inputs to NN: different combinations of harmonic magnitudes and phases.
- Output of NN: load type.

Figure 112

As a first step to identifying harmonic loads, a multi-layer perceptron was used to identify the type of harmonic load from among a set of pre-specified choices [57]. The training data for the NNs are generated by monitoring the current wave forms corresponding to each specific type of harmonic load. The fast fourier transform (FFT) of the digitized current wave form is used to produce the harmonic frequency spectrum. Different combinations of harmonic magnitudes and phases are then fed to the NN as inputs with the corresponding load type as the output.



Test Results of Harmonic Evaluation

Case I: Magnitude of harmonic currents of order h = 1, 2,......31;

Case II: Magnitude of odd harmonic currents of order h = 1,3,5,...,31;

Case III: Magnitude of harmonic currents of order h = 2, 3, 4, 5, 7, 9, 11 and phase angles of order k = 3, 5, 7, 9, 11;

Learning Set	Testing Set								
	Case I			Case II			Case III		
	A	B	C	A	B	C	A	B	C
A	90	92	86	96	73	68	100	100	100
B	94	99	78	84	98	95	100	100	100
C	61	99	97	92	99	97	90	96	100

From [57] courtesy of IEEE, (C) IEEE,1989

©1992 - Applications to Power Systems/M. A. El-Sharkawi -113

DSI

Figure 113

The ability to correctly classify the load based on the harmonic currents is investigated for three cases. NNs are trained and tested using 3 separate data sets. Several NN architectures with different numbers of hidden layers are used to find the optimal NN design. The NN has six hidden neurons.

It is clearly seen that NN trained under case III configuration has the best classification performance.

Notes Notes Notes

Harmonic Prediction

- o To predict the magnitude of a selected harmonic producing device in a time series form.

$$X^{(i)}(t+1) = f\left(X^{(i)}(t), X^{(i)}(t-1), \ldots\ldots, X^{(i)}(t-k)\right)$$

where,

$X^{(i)}(t)$ - magnitude of the ith harmonic at time t

- o Objective is to predict the magnitude $X^{(i)}(t+1)$ based on a time series of the past magnitudes.

DSI

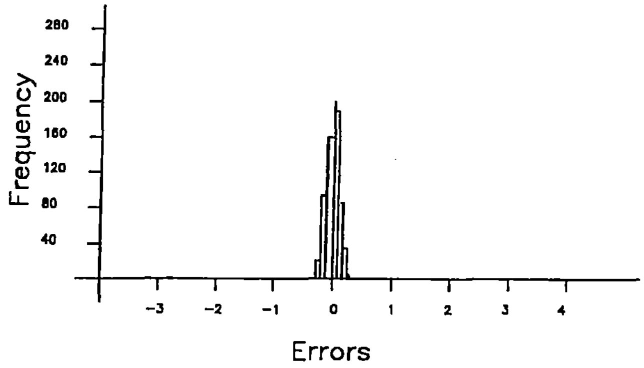
Figure 114

Test Results of Harmonic Prediction

- The performance of the NN is compared to nonlinear system identification algorithms
- The NN identifier was observed to give an error distribution of lower variance compared with the RGMDH algorithm.

Figure 115

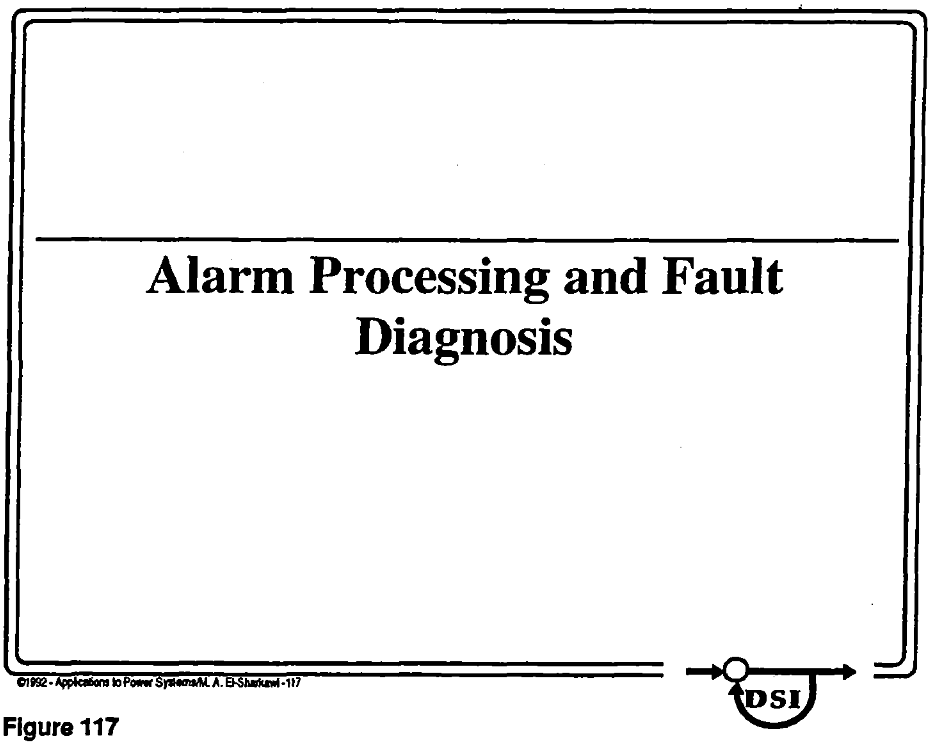
Sample of test results



Reproduced from reference [57]

Figure 116

In subsequent development, a multi-layer perceptron was used to predict the magnitude of a selected harmonic in a time series form [58]. A series of multi-layer perceptrons were trained to predict the magnitude $X^{(i)}(t+1)$ based on a time series of the past magnitudes. The structure of the NN is given in the figure. The performance of the NN was compared with another nonlinear system identification algorithm known as the Revised Group Method of Data Handing (RGMDH). The NN identifier was observed to give an error distribution of lower variance compared with the RGMDH algorithm.

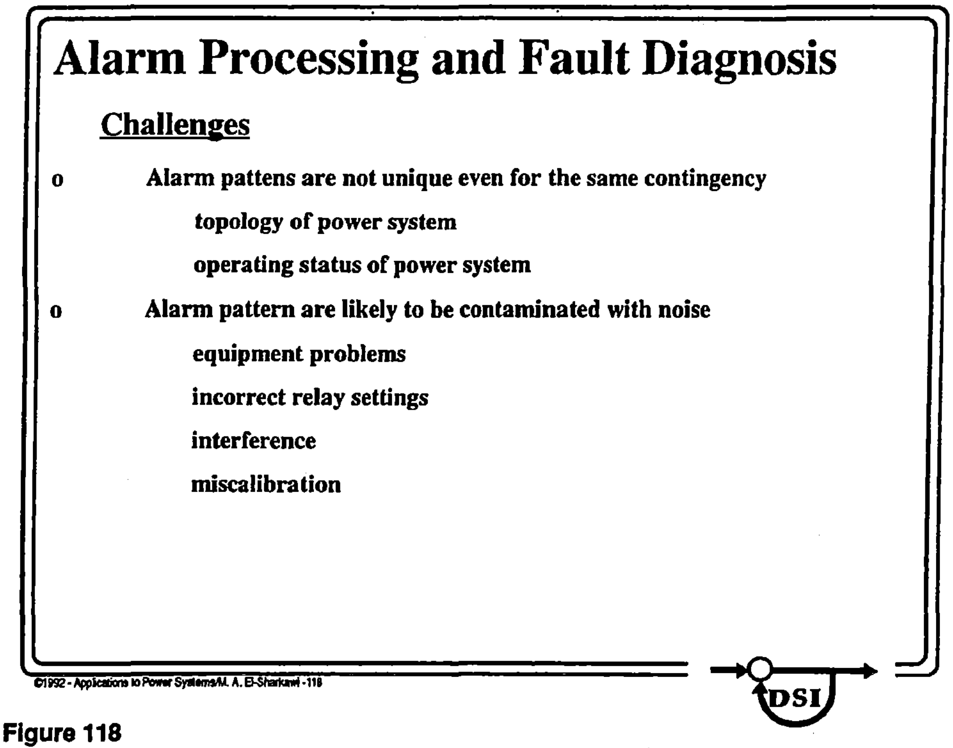


Alarm Processing and Fault Diagnosis

©1992 - Applications to Power Systems/M. A. El-Sharkawi -117

Figure 117

Notes Notes Notes



Alarm Processing and Fault Diagnosis

Challenges

- o Alarm pattens are not unique even for the same contingency
  - topology of power system
  - operating status of power system
- o Alarm pattern are likely to be contaminated with noise
  - equipment problems
  - incorrect relay settings
  - interference
  - miscalibration

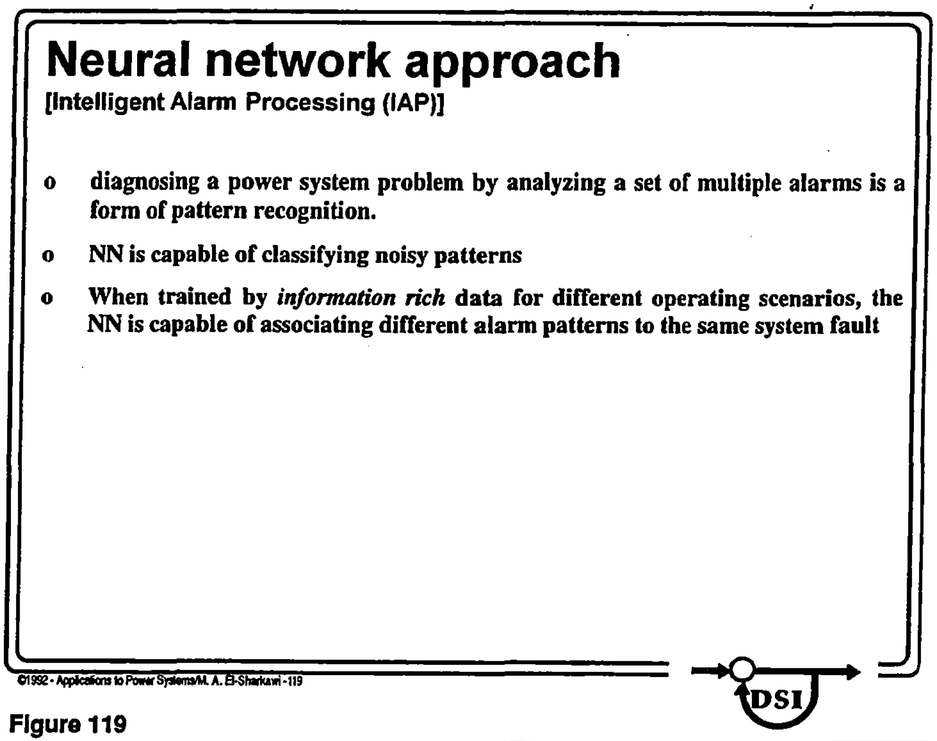
©1992 - Applications to Power Systems/M. A. El-Sharkawi - 118

Figure 118

The control centers of a power system are continuously interpreting large number of alarms signals to determine the status of the system components and to evaluate the power system operation. This process is very complex because of two key reasons:

1. Alarm patterns are not unique to a given power system problem. Same fault may manifest in different alarm patterns based on the current topology and operating status of the power system.
2. Alarm pattern are likely to be contaminated with noise due to equipment problems, incorrect relay settings, interference, or miscalibrated meters.

Expert system techniques have been widely tested for analyzing alarm signals. The formulation of rules, however, requires precise definitions of the power system and its operational strategies which may widely vary depending on the utility. Therefore, expert system technique are known to suffer from a high customization effort.



Neural network approach

[Intelligent Alarm Processing (IAP)]

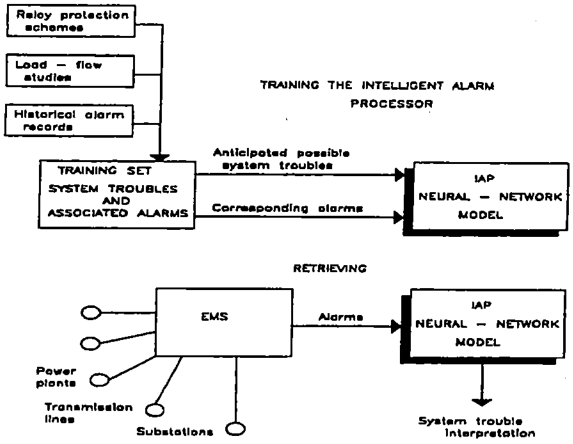
- **diagnosing a power system problem by analyzing a set of multiple alarms is a form of pattern recognition.**
- **NN is capable of classifying noisy patterns**
- **When trained by *information rich* data for different operating scenarios, the NN is capable of associating different alarm patterns to the same system fault**

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 119

Figure 119

The ability of a power system operator to diagnose a system problem by analyzing a set of multiple alarms is a form of pattern recognition. Accurate classification of noisy alarm patterns is also a key shortcoming in most of the conventional techniques. Therefore, NNs with their ability to classify noisy patterns seems a logical choice for alarm processing. The NN is also capable of associating different alarm patterns to the same system fault by training the NN with a set of *information rich* data that represents different operating scenarios [59].

Intelligent Alarm Processing



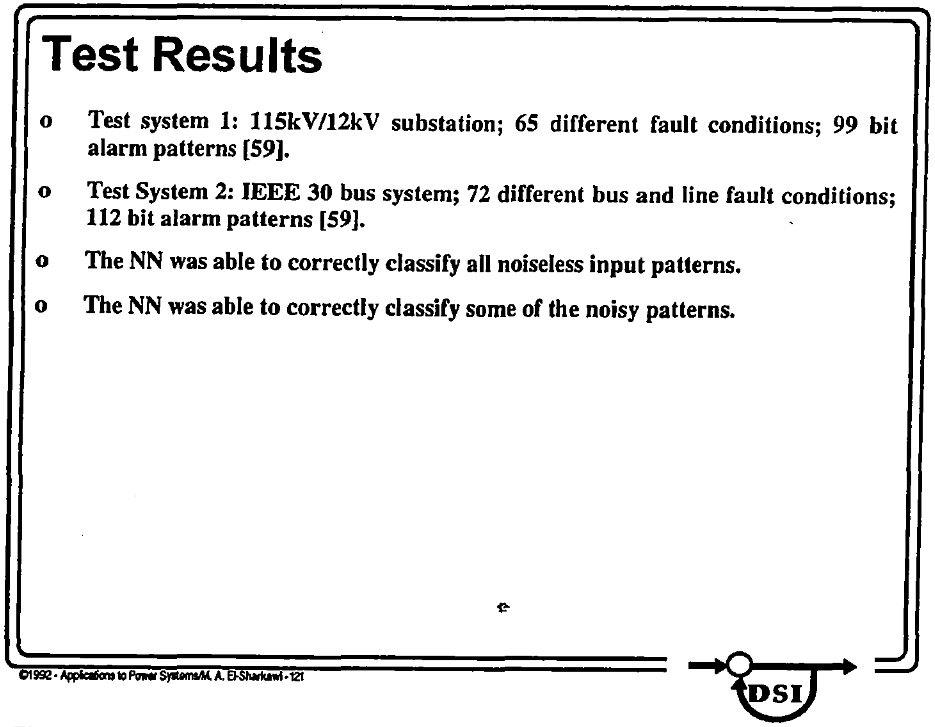
Concept of using NN for IAP

From [59] courtesy of IEEE, (C) IEEE, 1989

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 120

Figure 120

The figure shows a block diagram showing the concept of intelligent alarm processing (IAP) using NNs. Learning and retrieving phases of the IAP NN is presented in the figure. The NN training set is generated by first creating a credible set of contingencies and then deriving the possible alarm patterns under each fault. These patterns are generated by the relay protection schemes and power flow analyses. These patterns are then used to train a multi-layer perceptron using back-propagation [59]. In the retrieving phase, incoming alarm patterns from the energy management system (EMS) are interpreted to predict the possible fault scenario.



Test Results

- Test system 1: 115kV/12kV substation; 65 different fault conditions; 99 bit alarm patterns [59].
- Test System 2: IEEE 30 bus system; 72 different bus and line fault conditions; 112 bit alarm patterns [59].
- The NN was able to correctly classify all noiseless input patterns.
- The NN was able to correctly classify some of the noisy patterns.

Figure 121

The concept was tested on a 115kV/12kV substation for 65 different fault conditions with 99 bit alarm patterns [59]. It was also tested on the IEEE 30 bus system for 72 different bus and line fault conditions with 112 bit alarm patterns [59]. Results showed that the trained NN was able to correctly classify all noiseless input patterns. NN was also able to correctly classify some of the noisy patterns. Noisy patterns were generated by randomly toggling certain bits of the original input pattern. It is also worth mentioning that when noisy patterns were incorrectly classified by the NN, the system operator, given the same noisy pattern, also reached the same wrong conclusion.

Comments

- This is an area where NN seems to have a great potential
- Additional consideration for future work:
  - Order in which alarms are reported
  - Magnitude of the violations
  - Behavior of alarms over a certain time period.

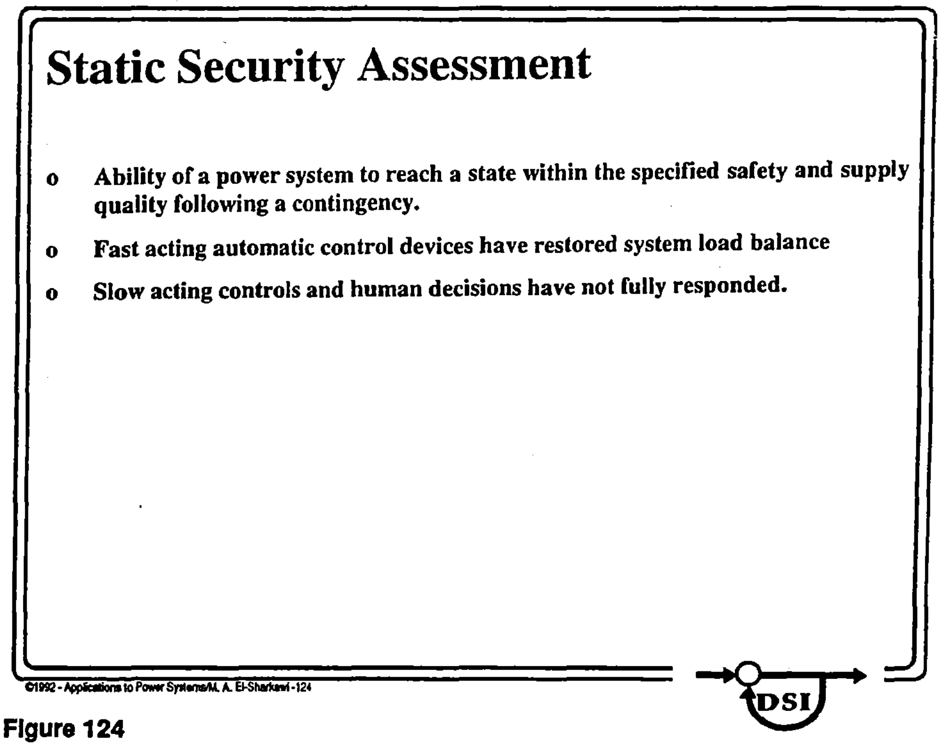
Figure 122

Notes Notes Notes

Static Security Assessment

DSI

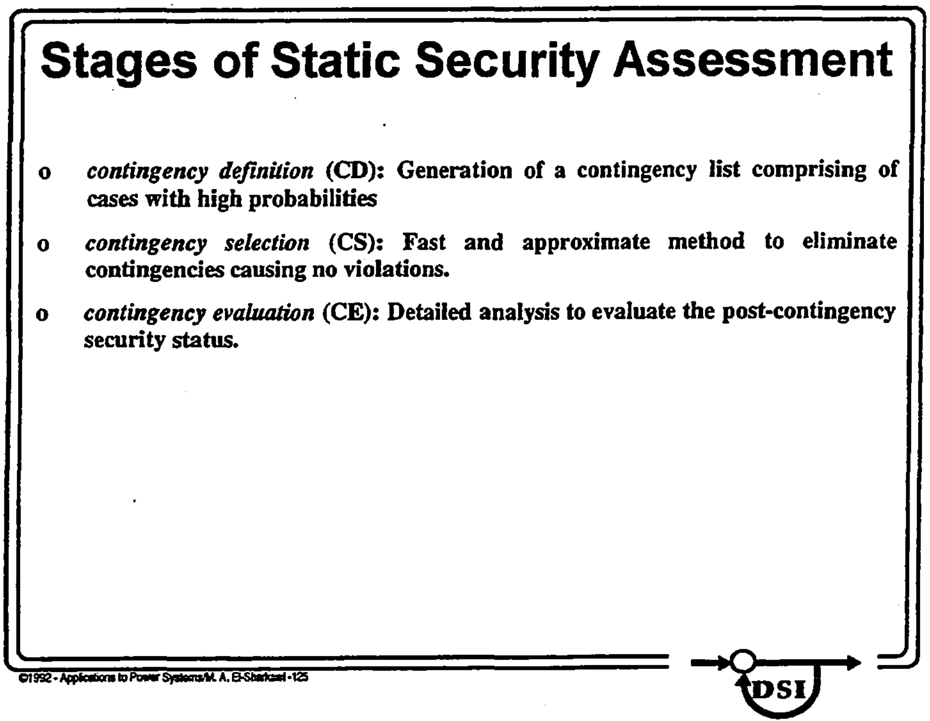
Figure 123



Static Security Assessment

- Ability of a power system to reach a state within the specified safety and supply quality following a contingency.
- Fast acting automatic control devices have restored system load balance
- Slow acting controls and human decisions have not fully responded.

Figure 124



Stages of Static Security Assessment

- *contingency definition* (CD): Generation of a contingency list comprising of cases with high probabilities
- *contingency selection* (CS): Fast and approximate method to eliminate contingencies causing no violations.
- *contingency evaluation* (CE): Detailed analysis to evaluate the post-contingency security status.

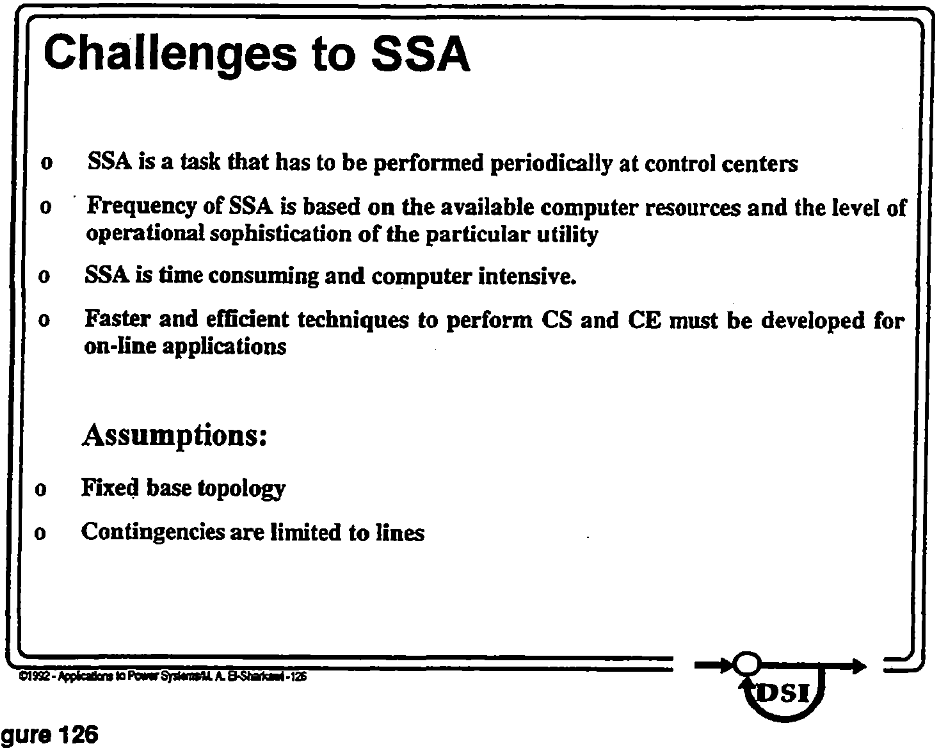
©1992 - Applications to Power Systems/M. A. El-Sharkawi - 125

Figure 125

Static security assessment is defined as the ability of a power system to reach a state within the specified safety and supply quality following a contingency. The time period of consideration is such that the fast acting automatic control devices have restored the system load balance, but the slow acting controls and human decisions have not responded.

Static security assessment consists of three distinct stages. They are *contingency definition* (CD), *contingency selection* (CS), and *contingency evaluation* (CE). CD defines a contingency list to be processed comprising of those cases whose probability of occurrence is deemed sufficiently high. CS is the process that shortens the original long list of contingencies by removing the vast majority of cases having no violations. Two commonly used algorithms for CS are contingency screening contingency ranking. These methods were introduced in a previous section. There has also been an increasing effort towards applying expert systems to augment the analytical CS methods [51]. CE is the process where the selected contingencies are simulated on the power system in order to evaluate the post-contingency security variables. The resulting system attributes are checked for security violations. the calculations are performed on each of the list of ranked contingencies. The number of cases evaluated depends on the amount of time and computer resources available for the task.

Notes Notes Notes



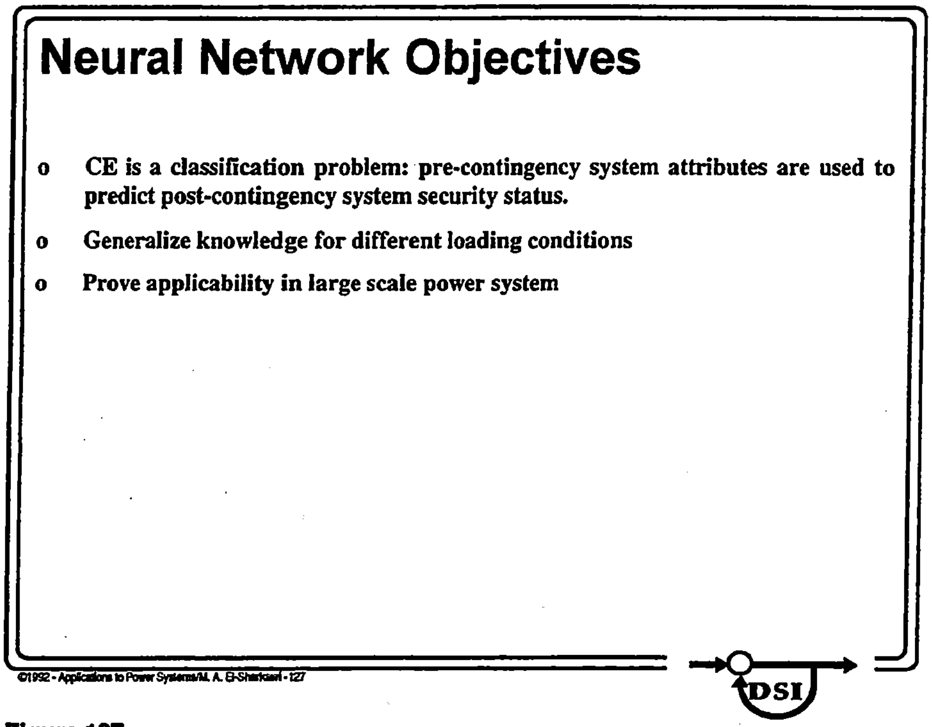
Challenges to SSA

- SSA is a task that has to be performed periodically at control centers
- Frequency of SSA is based on the available computer resources and the level of operational sophistication of the particular utility
- SSA is time consuming and computer intensive.
- Faster and efficient techniques to perform CS and CE must be developed for on-line applications

Assumptions:

- Fixed base topology
- Contingencies are limited to lines

Figure 126



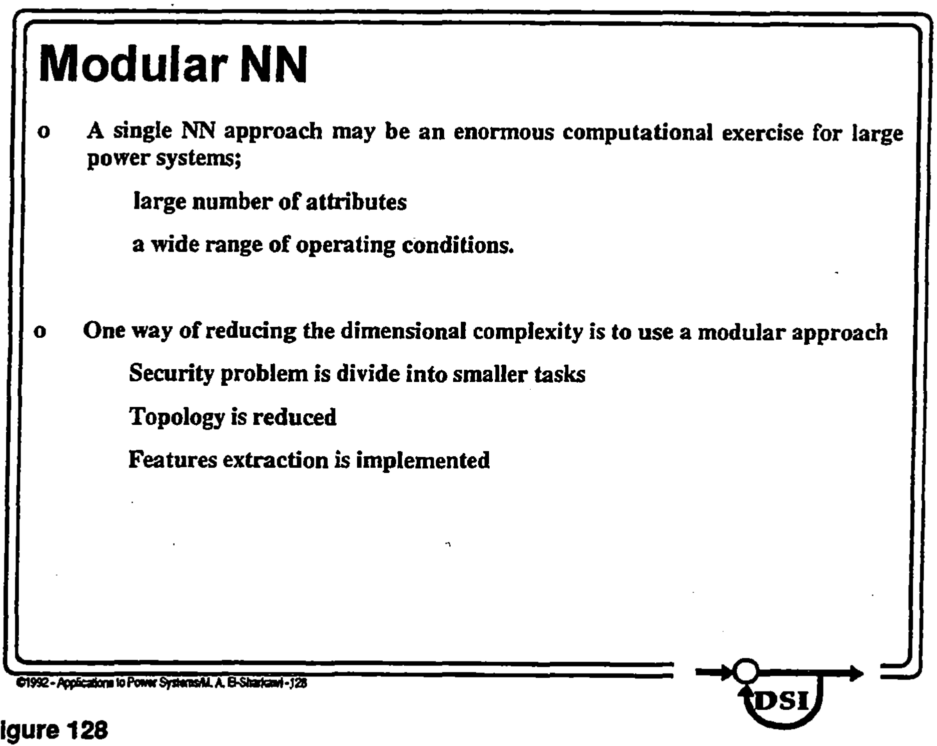
Neural Network Objectives

- CE is a classification problem: pre-contingency system attributes are used to predict post-contingency system security status.
- Generalize knowledge for different loading conditions
- Prove applicability in large scale power system

Figure 127

Notes Notes Notes

Notes Notes Notes



Modular NN

- o A single NN approach may be an enormous computational exercise for large power systems;

  large number of attributes

  a wide range of operating conditions.

- o One way of reducing the dimensional complexity is to use a modular approach

  Security problem is divide into smaller tasks

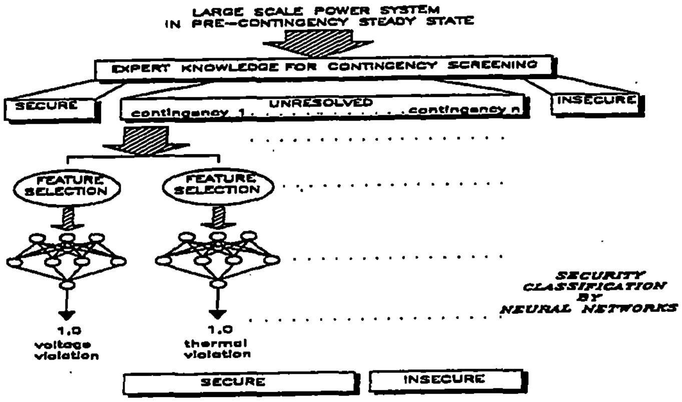
  Topology is reduced

  Features extraction is implemented

Figure 128

From a pattern recognition perspective, CE is a two class classification problem where the pre-contingency system attributes are used to predict post-contingency system security status. A multi-layer perceptron can be trained to perform this pattern classification [51]. But for a large power system, where a large number of attributes and operating conditions are needed to classify the system security, a single NN approach may be an enormous computational exercise. One way of reducing the dimensional complexity is to use a modular approach where the security problem is divide into smaller tasks or reduced topology. A modular NN can then be used to handle each task or topology.

Modular NN Approach with Feature Extraction



LARGE SCALE POWER SYSTEM IN PRE-CONTINGENCY STEADY STATE

EXPERT KNOWLEDGE FOR CONTINGENCY SCREENING

SECURE

UNRESOLVED contingency 1. . . . . .contingency n

INSECURE

FEATURE SELECTION

FEATURE SELECTION

SECURITY CLASSIFICATION BY NEURAL NETWORKS

1.0 voltage violation

1.0 thermal violation

SECURE

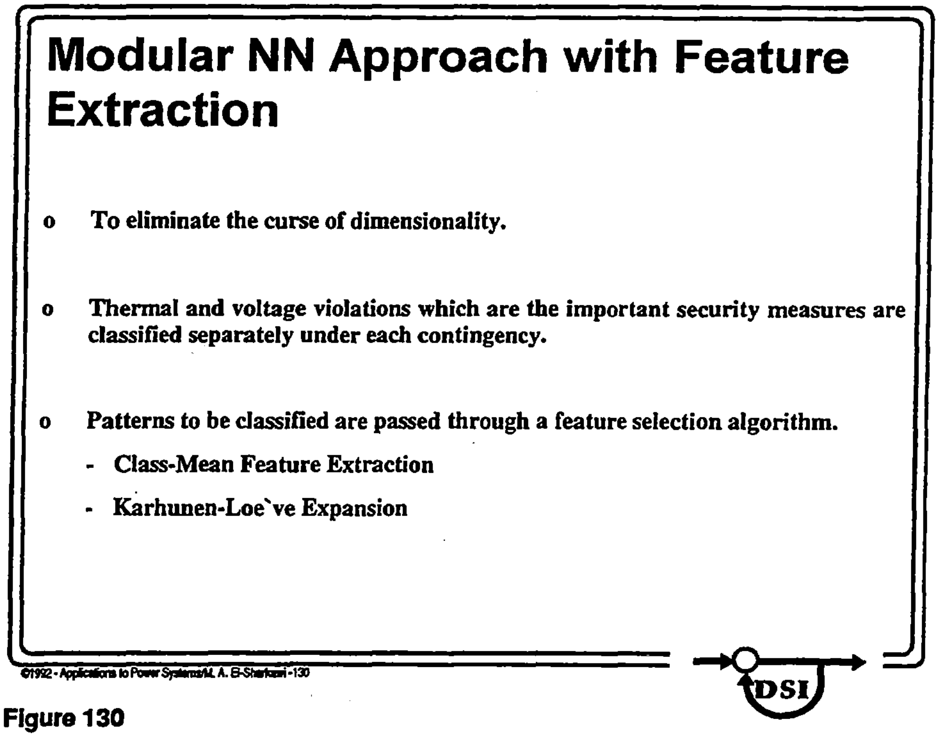
INSECURE

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 129

Figure 129

The Figure shows a possible modular approach to large power system problem. A specific NN for predicting security status under a specific contingency is proposed. This is necessary due to the variations in which a contingency manifests itself based on the nature, location and clearing strategy. Furthermore, for a given contingency, the mechanisms leading to line and voltage violations are fundamentally different. Line violations are brought about by real power overflows, while voltage violations are brought about by an excess or a deficiency of reactive power. Therefore, separate NNs are trained for assessing line and voltage violations under the same contingency.

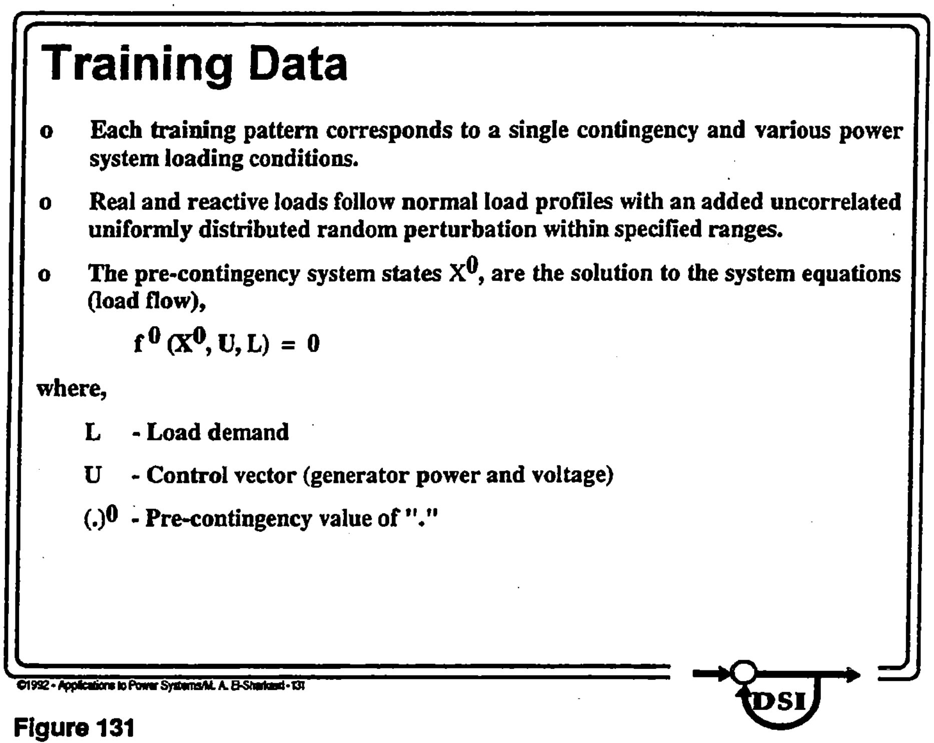
Notes Notes Notes



Modular NN Approach with Feature Extraction

- o To eliminate the curse of dimensionality.
- o Thermal and voltage violations which are the important security measures are classified separately under each contingency.
- o Patterns to be classified are passed through a feature selection algorithm.
  - Class-Mean Feature Extraction
  - Karhunen-Loe`ve Expansion

Figure 130



Training Data

- o Each training pattern corresponds to a single contingency and various power system loading conditions.
- o Real and reactive loads follow normal load profiles with an added uncorrelated uniformly distributed random perturbation within specified ranges.
- o The pre-contingency system states X^0, are the solution to the system equations (load flow),

$$f^0(X^0, U, L) = 0$$

where,

L - Load demand

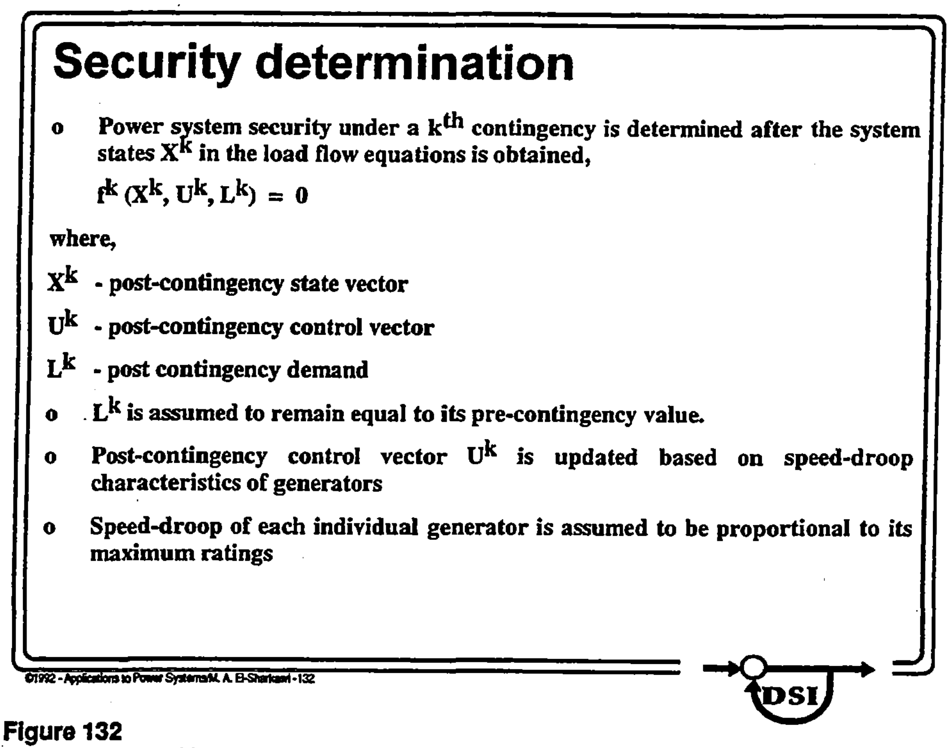
U - Control vector (generator power and voltage)

$(.)^0$ - Pre-contingency value of "."

Figure 131

Notes Notes Notes

Notes Notes Notes



Security determination

- o Power system security under a kth contingency is determined after the system states X^k in the load flow equations is obtained,

  $f^k(X^k, U^k, L^k) = 0$

where,

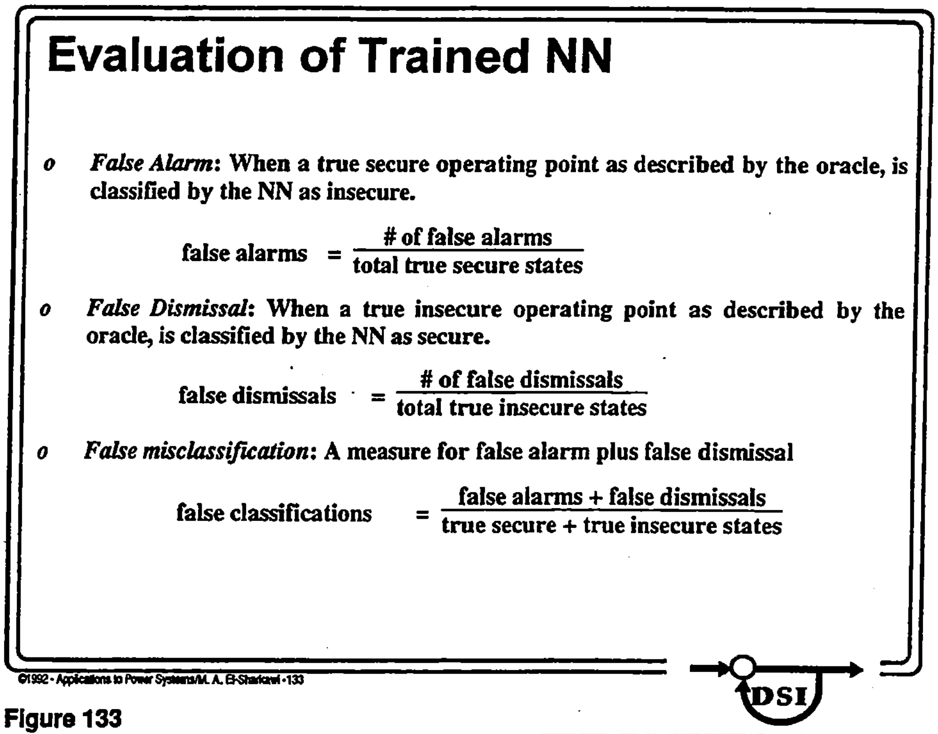
X^k - post-contingency state vector

U^k - post-contingency control vector

L^k - post contingency demand

- o L^k is assumed to remain equal to its pre-contingency value.
- o Post-contingency control vector U^k is updated based on speed-droop characteristics of generators
- o Speed-droop of each individual generator is assumed to be proportional to its maximum ratings

Figure 132



Evaluation of Trained NN

- o *False Alarm*: **When a true secure operating point as described by the oracle, is classified by the NN as insecure.**

$$\text{false alarms} = \frac{\text{\# of false alarms}}{\text{total true secure states}}$$

- o *False Dismissal*: **When a true insecure operating point as described by the oracle, is classified by the NN as secure.**

$$\text{false dismissals} = \frac{\text{\# of false dismissals}}{\text{total true insecure states}}$$

- o *False misclassification*: **A measure for false alarm plus false dismissal**

$$\text{false classifications} = \frac{\text{false alarms + false dismissals}}{\text{true secure + true insecure states}}$$

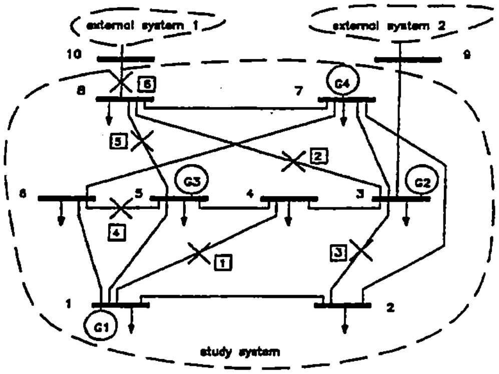
DSI

Figure 133

Notes Notes Notes

Notes Notes Notes

Test Case Extended IEEE-8 Bus System



o 10 buses (b), 16 lines, 2000 unbiased patterns (i), 30 attributes
0 2000 patterns, each with 30 attribute

©1992 - Applications to Power Systems/M. A. El-Sharkawi -134

Figure 134

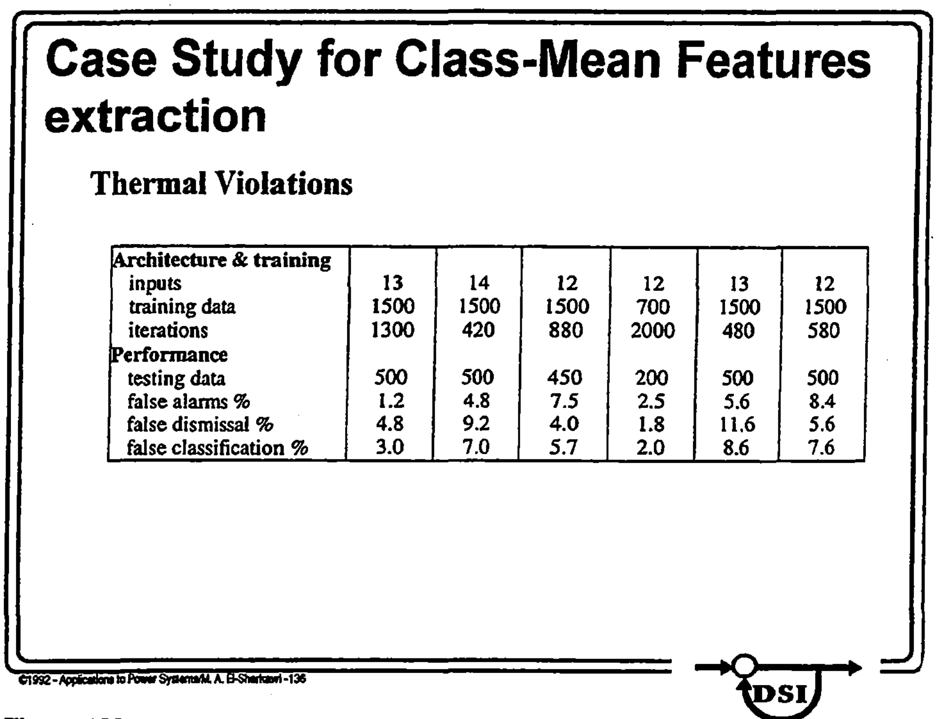
Case Study for Class-Mean Features extraction

Voltage Violations

Neural Network	Cont 1	Cont 2	Cont 3	Cont 4	Cont 5	Cont 6
Architecture & training						
inputs	7	6	7	5	7	7
training data	1500	1500	1500	1500	1500	945
iterations	2000	2000	2000	2000	2000	2000
Performance						
testing data	500	500	500	500	500	315
false alarms %	1.2	2.8	0.8	1.2	2.0	1.9
false dismissal %	0.0	0.4	1.6	2.0	2.8	3.7
false classification %	0.6	1.6	1.2	1.6	2.4	2.8

DSI

Figure 135



Case Study for Class-Mean Features extraction

Thermal Violations

Architecture & training						
inputs	13	14	12	12	13	12
training data	1500	1500	1500	700	1500	1500
iterations	1300	420	880	2000	480	580
Performance						
testing data	500	500	450	200	500	500
false alarms %	1.2	4.8	7.5	2.5	5.6	8.4
false dismissal %	4.8	9.2	4.0	1.8	11.6	5.6
false classification %	3.0	7.0	5.7	2.0	8.6	7.6

DSI

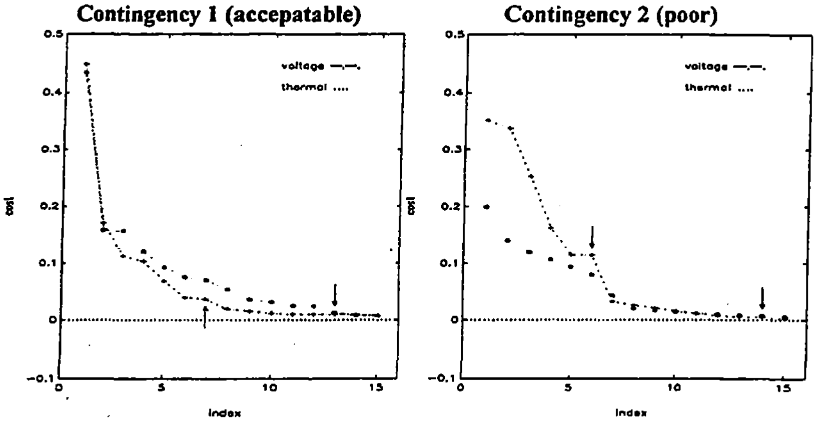
Figure 136

In this test, the tripping of tie line #16 is investigated. A single pre-contingency pattern contains 54 different attributes including all the real and reactive generation (P_{gi},Q_{gi}), real and reactive loads (P_{bj},Q_{bj}), all the bus voltage magnitudes (V_{bj}) and all the line currents (I_{Lk}) in the system. The key features (variables) for training the NN are selected as described earlier. Six features were used for NN training: Q_{b8}, V_{b8}, Q_{g2}, Q_{b10}, I_{L7}, I_{L14}. The training and testing statistics of the NN are given in the Table.

In the second case, the contingency is the tripping of the transmission line between buses #5 and #6. The training data are generated similar to the previous case. The input attributes for the NN are selected by the feature selection algorithm described earlier. The features Q_{b6}, Q_{g1}, Q_{g3}, Q_{g4}, I_{L3}, I_{L11} and I_{L12} are selected. The training and testing statistics for the NN in case II are given in table 6.

Notes Notes Notes

Example



Contingency 1 (accepatable)

Contingency 2 (poor)

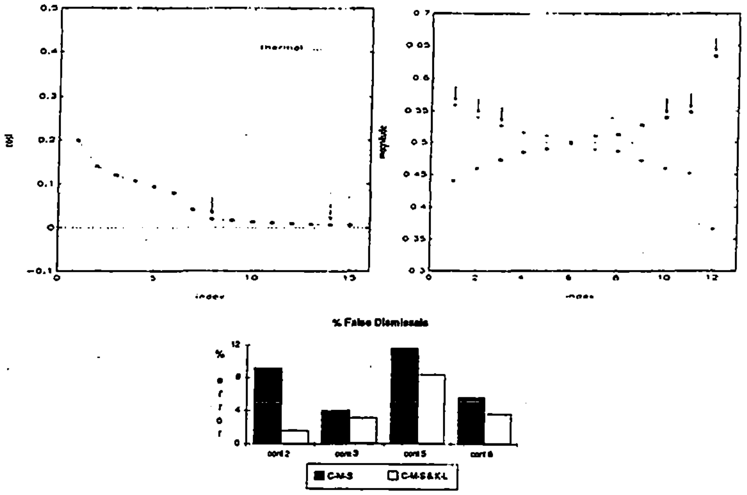
"Lower first order discriminatory information results in poor classifier performance"

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 137

Figure 137

Notes Notes Notes

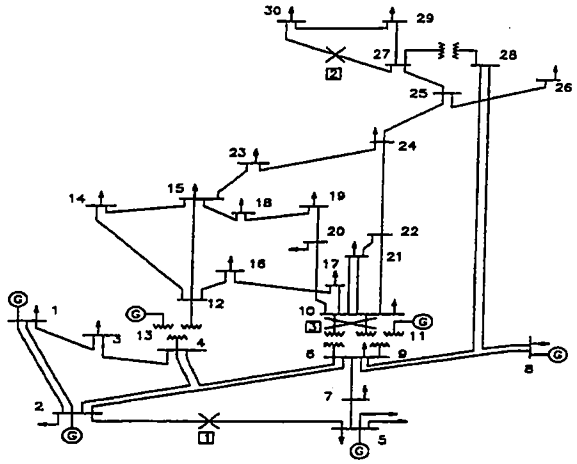
Combined CMS and KL



©1992 - Applications to Power Systems/M. A. El-Sharkawi - 138

Figure 138

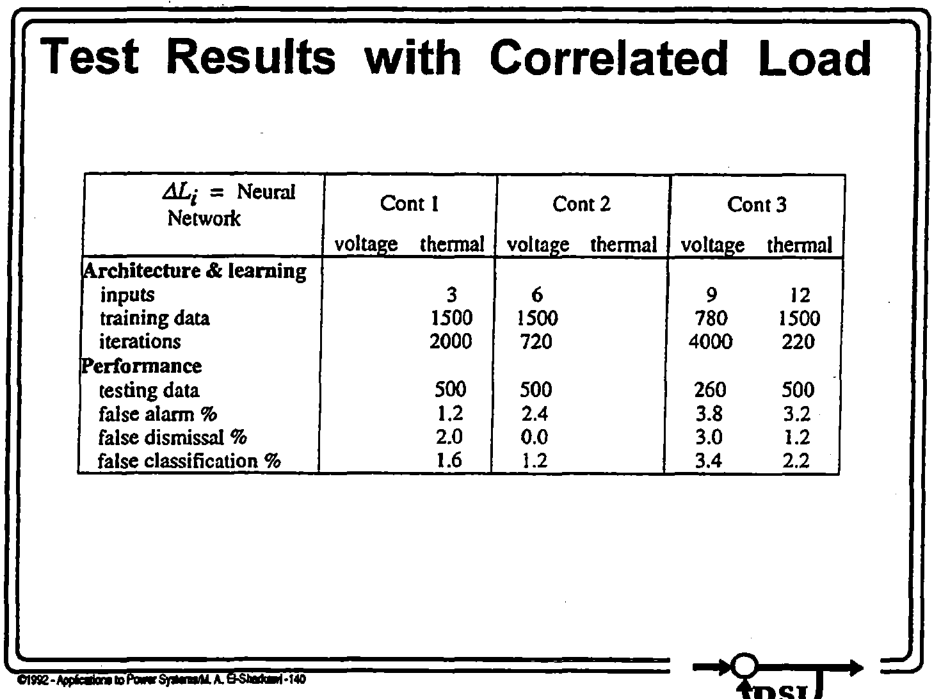
IEEE-30 Bus System



- 30 buses, 41 lines, 2000 patterns, 76 attributes

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 139

Figure 139

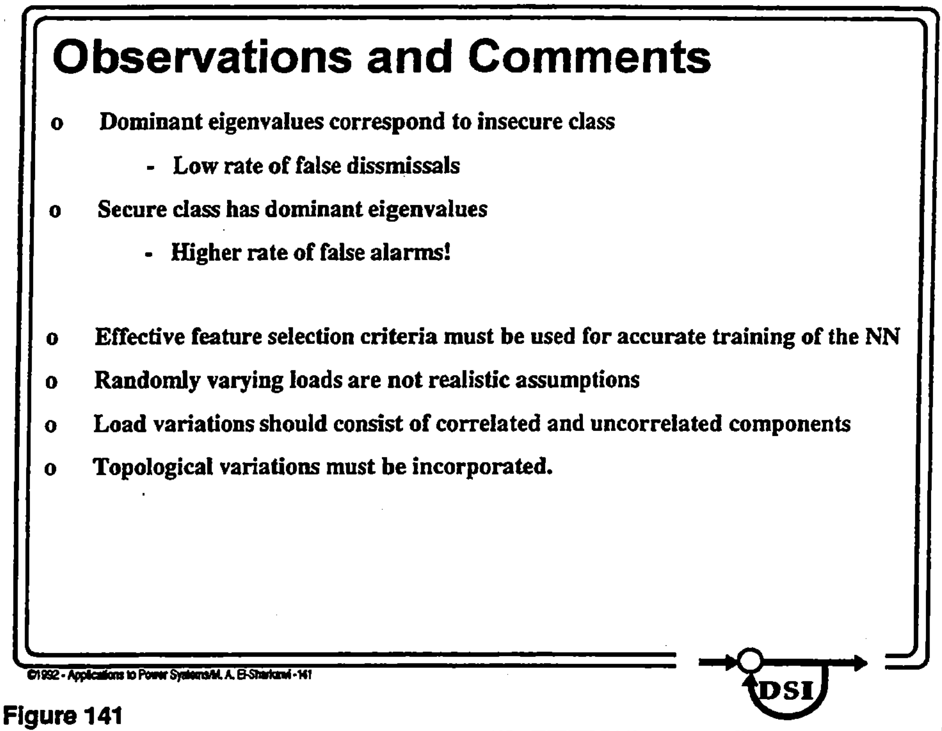


Test Results with Correlated Load

ΔL_i = Neural Network	Cont 1		Cont 2		Cont 3	
	voltage	thermal	voltage	thermal	voltage	thermal
Architecture & learning						
inputs		3	6		9	12
training data		1500	1500		780	1500
iterations		2000	720		4000	220
Performance						
testing data		500	500		260	500
false alarm %		1.2	2.4		3.8	3.2
false dismissal %		2.0	0.0		3.0	1.2
false classification %		1.6	1.2		3.4	2.2

DSI

Figure 140



Observations and Comments

- o Dominant eigenvalues correspond to insecure class
  - Low rate of false dissmissals
- o Secure class has dominant eigenvalues
  - Higher rate of false alarms!
- o Effective feature selection criteria must be used for accurate training of the NN
- o Randomly varying loads are not realistic assumptions
- o Load variations should consist of correlated and uncorrelated components
- o Topological variations must be incorporated.

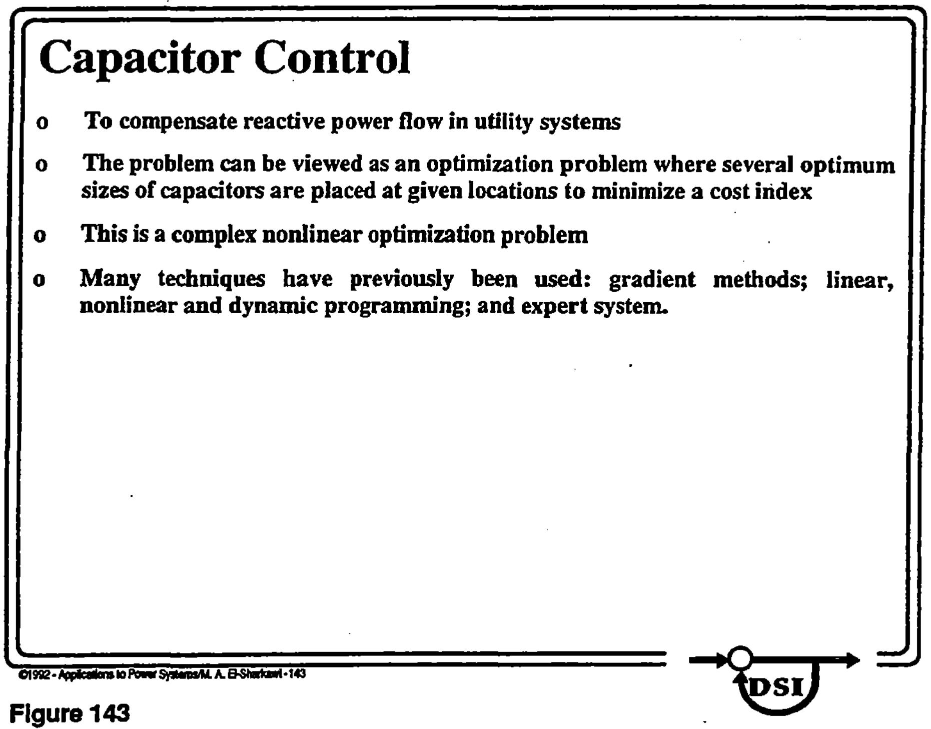
Figure 141

Notes Notes Notes

Capacitor Control

DSI

Figure 142



Capacitor Control

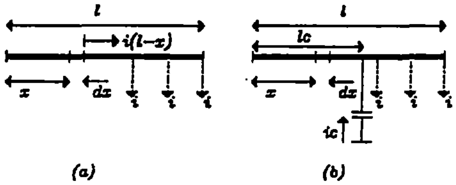
- o To compensate reactive power flow in utility systems
- o The problem can be viewed as an optimization problem where several optimum sizes of capacitors are placed at given locations to minimize a cost index
- o This is a complex nonlinear optimization problem
- o Many techniques have previously been used: gradient methods; linear, nonlinear and dynamic programming; and expert system.

Figure 143

Compensating the reactive power flow in utility systems is an area of continuous development. Reactive power has limiting effect on the operation of the power system due to the line losses and unnecessary equipment load. The reactive power compensation can be viewed as an optimization problem where several optimum sizes of capacitors can be placed at optimum locations to minimize a cost index such as line (or system) losses. This is a complex nonlinear optimization problem. Many techniques have previously been used such as gradient methods, linear, nonlinear and dynamic programming and expert system methods.

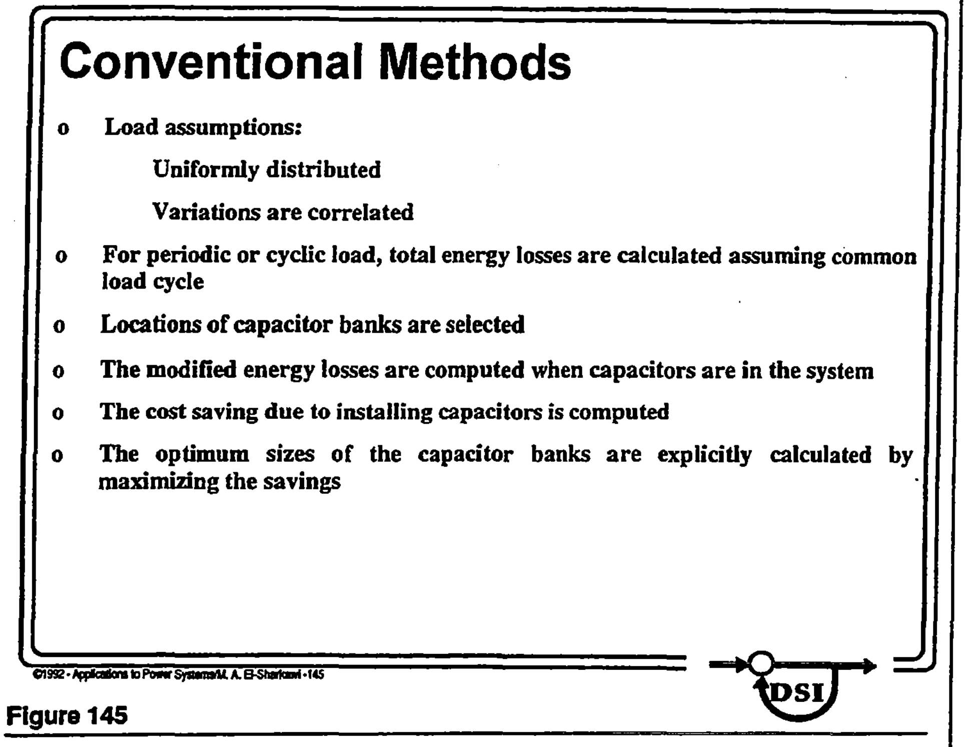
Notes Notes Notes

Problem description



(a) Uniformly loaded cable (b) capacitor compensated cable

Figure 144



Conventional Methods

- Load assumptions:
  - Uniformly distributed
  - Variations are correlated
- For periodic or cyclic load, total energy losses are calculated assuming common load cycle
- Locations of capacitor banks are selected
- The modified energy losses are computed when capacitors are in the system
- The cost saving due to installing capacitors is computed
- The optimum sizes of the capacitor banks are explicitly calculated by maximizing the savings

Figure 145

The 3-phase (3ϕ) power loss in an elemental length dx due to the resistance of the cable is given by

$$d\,L_{3\phi} = 3\,r\,i^2\,(h - x)^2\,dx$$

where

i - current per unit length

r - resistance per unit length

h - length of the cable

The total 3ϕ power loss (w) along the feeder is given by

$$L_{3\phi} = 3ri^2\int_0^h (h-x)^2dx = ri^2h^3 = R_TI_T^2$$

where

$R_T = r\,h$ - the total resistance of the cable

$I_T = i\,h$ - the total load current drawn in to the cable

Assuming that the load is cyclic with a period of T hours, the total energy loss (wh) can be calculated as,

$$E_{3\phi} = \int_0^T L_{3\phi} dt = R_T \int_0^T I_T^2 dt = R_T I_{T\,MAX}^2 L_s T$$

Now consider the installation of a capacitor bank at location h_c as shown in the figure. The 3ϕ power loss (w) can now be modified as,

$$L_{3\phi} = 3r \int_0^{h_c} (i\,(h-x) - i_c)^2 dx + \int_{h_c}^{h} i^2 (h-x)^2 dx$$

$$L_{3\phi} = 3r\,[h^3 i^2/3 + (h_c - 2h_c h)\,i_2 h_c + i_2^2 h_c]$$

where,

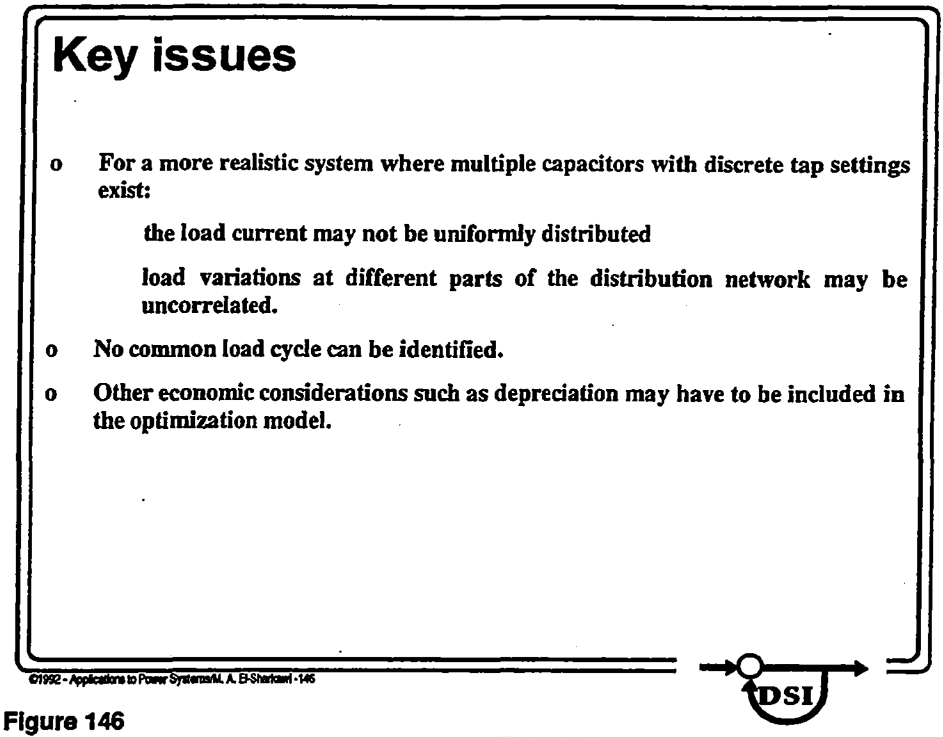
i_2 reactive component of current i

i_c capacitive current provided by the bank

The modified energy loss can be similarly calculated. The cost saving due to installing capacitors to decrease energy and power losses is given by,

$$\Uparrow C = K_1 \Delta E_{3\phi} + K_2\, \Delta L_{3\phi}$$

where K_1 and K_2 are two cost factors. The optimum size and location of the capacitor bank can be explicitly calculated by setting the partial derivatives $(\delta \Delta C/\delta i_c)$ and $(\delta \Delta C/\delta l_c)$ to zero.



Key issues

- For a more realistic system where multiple capacitors with discrete tap settings exist:

  the load current may not be uniformly distributed

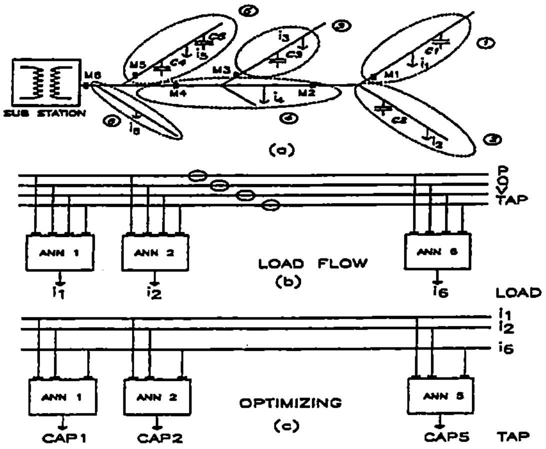
  load variations at different parts of the distribution network may be uncorrelated.
- No common load cycle can be identified.
- Other economic considerations such as depreciation may have to be included in the optimization model.

Figure 146

In a real power system, the conventional method can not be easily applied. The distribution system can have multiple capacitors with discrete tap settings. The load current may not be uniformly distributed and the load variations at different parts of the distribution network may be uncorrelated. Hence, no common load cycle can be identified. Also, other economic considerations such as depreciation, return on investment etc. may have to be included in the optimization model. In order to deal with these constraints, linear and nonlinear programming techniques can be employed. Expert systems also have been looked at as a possible alternative. However, solution accuracy and computational time are a major concern in most of these techniques.

Notes Notes Notes

Neural Network Approach

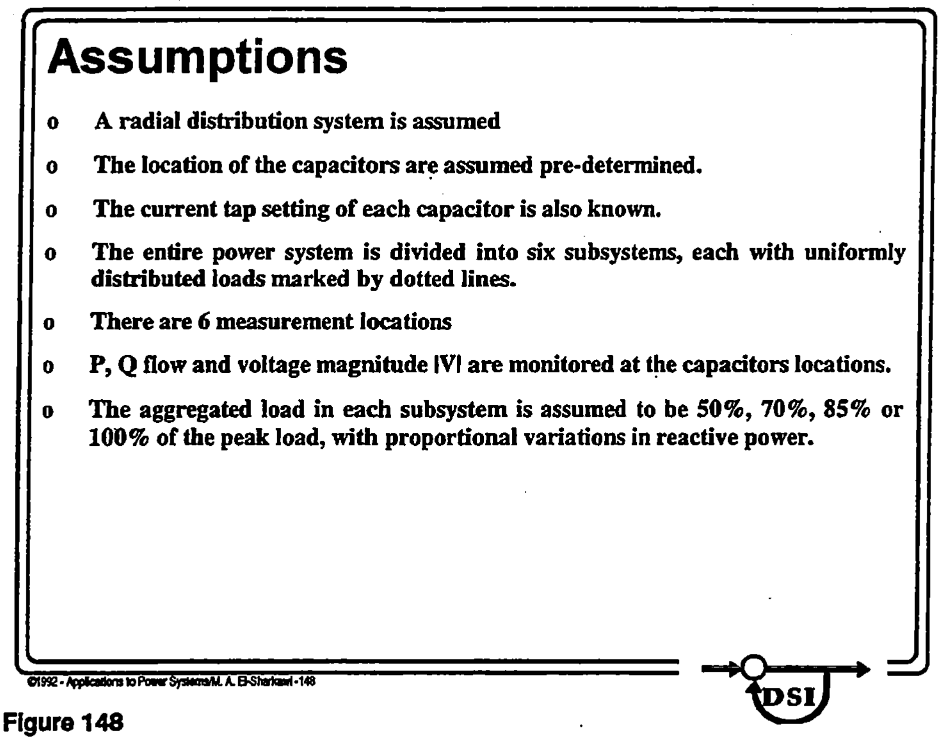


Capacitor control through NNs

From [62] courtesy of IEEE, (C) IEEE, 1989

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 147

Figure 147

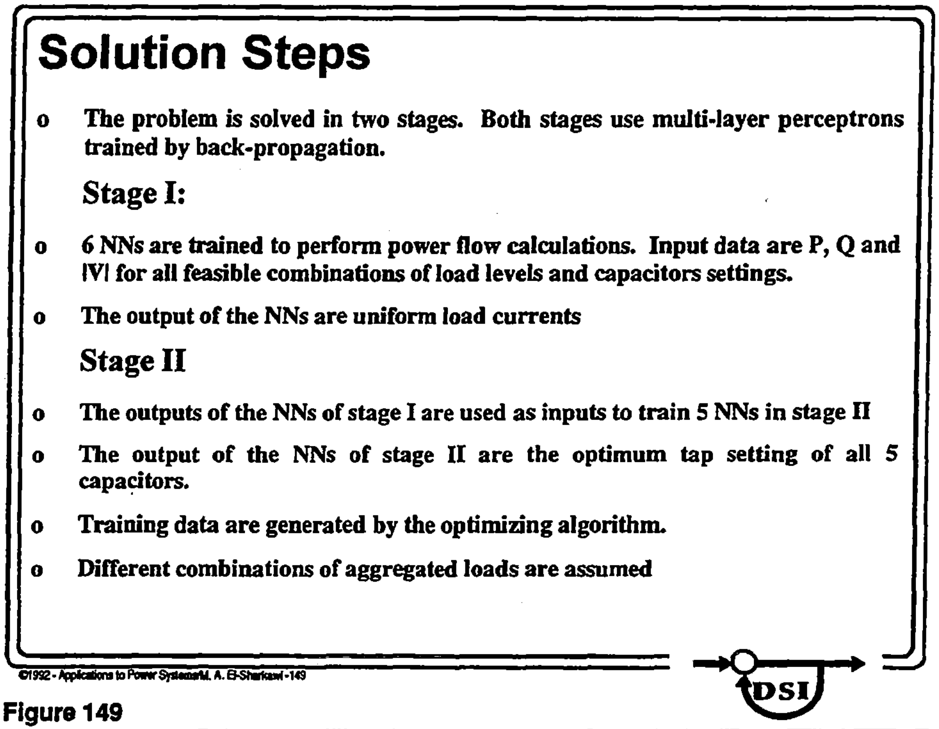


Assumptions

- A radial distribution system is assumed
- The location of the capacitors are assumed pre-determined.
- The current tap setting of each capacitor is also known.
- The entire power system is divided into six subsystems, each with uniformly distributed loads marked by dotted lines.
- There are 6 measurement locations
- P, Q flow and voltage magnitude |V| are monitored at the capacitors locations.
- The aggregated load in each subsystem is assumed to be 50%, 70%, 85% or 100% of the peak load, with proportional variations in reactive power.

DSI

Figure 148



Solution Steps

- o The problem is solved in two stages. Both stages use multi-layer perceptrons trained by back-propagation.

Stage I:

- o 6 NNs are trained to perform power flow calculations. Input data are P, Q and |V| for all feasible combinations of load levels and capacitors settings.
- o The output of the NNs are uniform load currents

Stage II

- o The outputs of the NNs of stage I are used as inputs to train 5 NNs in stage II
- o The output of the NNs of stage II are the optimum tap setting of all 5 capacitors.
- o Training data are generated by the optimizing algorithm.
- o Different combinations of aggregated loads are assumed

©1992 - Applications to Power Systems/M. A. El-Sharkawi - 149

DSI

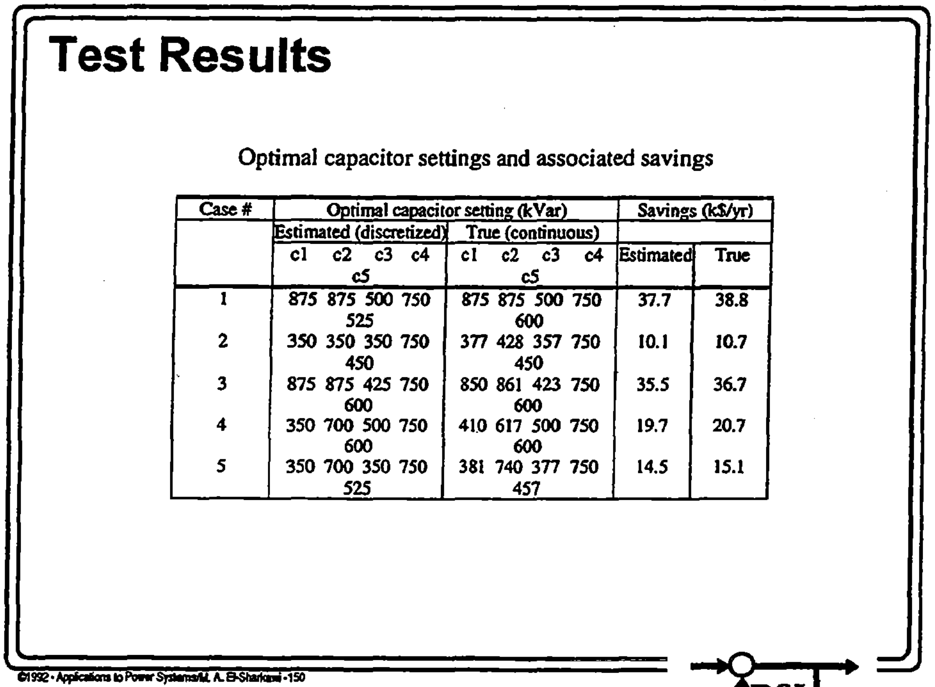
Figure 149

The NN assisted approach to the solution of capacitor control problem is expected to drastically reduce the calculation times and enable on-line adjustments. A specific example in the control of capacitors on a radial distribution system is addressed in [62]. The test power system is given in figure (a). The location of the capacitors are assumed predetermined. The entire power system is divided into six subsystems, each with uniformly distributed loads marked by dotted lines. There are 6 measurement locations marked by M_1 through M_6. P, Q flow and the voltage magnitude |V| are monitored at the capacitor locations. The aggregated load in each subsystem is assumed to take one of 4 feasible levels at 50%, 70%, 85% and 100% of the peak load with proportional variations in reactive power. The current tap setting of each capacitor is also known. The objective is to use 3 measurement quantities (P,Q,|V|) at locations M_1 through M_6 and the current tap settings of the capacitors C1 through C5 in order to calculate the optimum tap settings for the 5 capacitors.

The problem is solved in two stages. Both stages use multi-layer perceptrons trained by back-propagation. In stage I, 6 NNs, shown in figure (b), are trained to perform a power flow calculation. The training data for the this stage are the P, Q, |V| measurements for all feasible combinations of load levels and capacitor settings. The output of the

NNs are uniform load currents i_1 through i_6. In the figure, the circles placed on the lines indicate multiple measurements.

In stage II, the outputs of the NNs of stage I (i_1 through i_6) are used to train 5 NNs as shown in figure (c). In this stage, the NNs are trained to select the optimum tap setting of all 5 capacitors. Training data for stage II are generated by the optimizing algorithm. Different combinations of aggregated loads on the 6 subsystems are assumed. In the retrieving phase, the NN estimated the optimum tap settings.



Test Results

Optimal capacitor settings and associated savings

Case #	Optimal capacitor setting (kVar)		Savings (k$/yr)	
	Estimated (discretized)	True (continuous)		
	c1 c2 c3 c4 c5	c1 c2 c3 c4 c5	Estimated	True
1	875 875 500 750 525	875 875 500 750 600	37.7	38.8
2	350 350 350 750 450	377 428 357 750 450	10.1	10.7
3	875 875 425 750 600	850 861 423 750 600	35.5	36.7
4	350 700 500 750 600	410 617 500 750 600	19.7	20.7
5	350 700 350 750 525	381 740 377 750 457	14.5	15.1

DSI

Figure 150

The estimated descretized capacitor settings as estimated by the NN are compared to the true continuous optimum values. The table also show the corresponding energy savings in k$/yr obtained by the NN predictions and by the optimization method using the true continuous capacitor values. Relatively small difference between the two columns shows the adequacy of this method.

Notes Notes Notes

Comments

- o Partitioning of the overall problem into smaller subproblems is a significant contribution.
- o This modular approach facilitates faster and simpler training of the NN's.

DSI

Figure 151