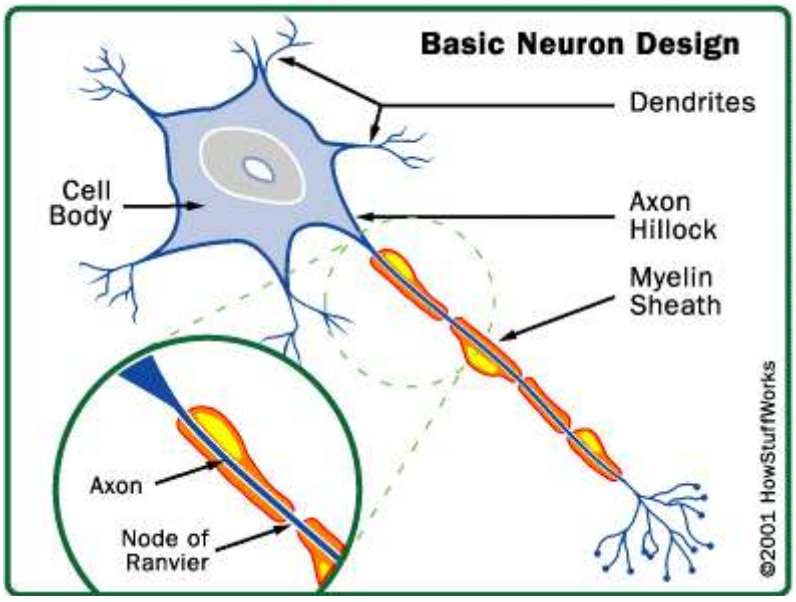
Artificial Neural Networks

Lecture Notes

2018



Dr. Samy Abu Naser
Faculty of Engineering and Information Technology
Al Azhar University-Gaza

Artificial Neural Networks
Lecture Notes

Chapter 1

Contents

Brief Overview

- **Historical Note** McCullloch and Pitts (1943) - developed the first model of artificial neurons.

- When we study artificial neural networks, we need to clarify their relationship to the biological paradigm.

- "Artificial neural networks are an attempt at modeling the information processing capabilities of the nervous system" - from the textbook.

- Animal nervous systems are composed of thousands or millions of interconnected cells. In the case of humans, it is billions of cells.

- Neurons are slow when compared to electronic logic gates. Neuronal delays are on the order of milli-seconds, vs. fractions of a nanosecond in electronic gates.

- Biologists and neurologists understand the mechanisms whereby individual neurons communicate with one another. However, the mechanism whereby massively parallel and hierarchical collections of neurons form functional units eludes us.

- We study the information processing capabilities of complex arrangements of simple computing units (i.e., artificial neurons.)

- **The networks will be adaptive** - Adjustment of parameters is done through a learning algorithm (i.e., there will be no explicit programming.)

Models of Computation

Artificial neural networks can be considered as just another approach to the problem of computation.

Formal Definitions of Computability (1930's & 1940's)
The following lists 5 classic approaches to the study of computability.

- The Mathematical Model
- The Logic Operational Model (Turing Machines)
- Cellular Automata
- The Computer Model
- Biological Model (Neural Networks)

I. The Mathematical Model

- **Notable Contributors:** Hilbert, Ackerman, Church and Kleene.
- **Set of Primitive Functions**

Zero-Function	$Z(x) = 0 \quad \forall x \in N$
Successor Function	$S(x) = x + 1$ Example $S(1) = 2$ $S(2) = 3$, &c.
Projection Function	$U^{I}n(x_1, x_2, x_3, \ldots x_i, \ldots x_n) = x_i$ Example $U^{2}3(x_1, x_2, x_3) = x_2$.
Set of Operations	Composition and Recursion u - minimization operator.

- **A simple Example: Showing that Addition is Recursive**

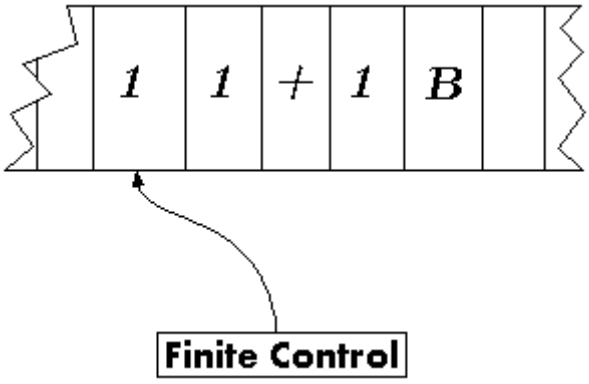
Let us represent the addition of two numbers, m and n, by the function f(m,n).
Thus, we have f(x,0) = x (i.e., x+0 = x - Identity axiom for addition)
f(x,y+1) = S(F(x,y); (i.e., x + (y+1) = successor function of (x+y)

- For example, to add 3+2, we have

$$\begin{aligned} f(3,2) &= S(f(3,1)) \\ &= S(S(f(3,0))) \\ &= S(S(3)) \\ &= S(4) \\ &= 5. \end{aligned}$$

II. The Logical/Operational Model: The Turing Machine

- **Notable Contributors:** Alan Turing 1936, inventor of the Turing Machine; Church.



A Turing Machine "tape" illustrating the unary addition 2+1

- The Turing machine consists of a tape serving as the internal memory of the machine, of unlimited size, and a read/write head which moves along the tape.

- The Turing machine is described by the state, output and direction functions. We can write (state, input) → (state, write, {L, R, N}) where L,R,N mean Left, Right and No movement, respectively.

- In the above figure, we can describe the process of addition in 5 steps. The first step looks like this
- (q0, 1, q0, 1, R) where the first q0 is the current state, followed by the current symbol scanned, followed by the next state, output, and in the end, the direction to move (in this case Rightward). Write the remaining 4 steps.

- The operation of this addition may be described or traced as follows:

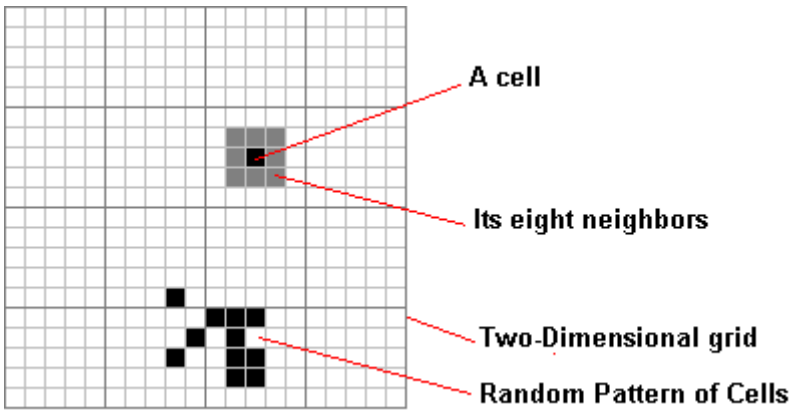
(q_0,11+1) →
1. (1q_0,1+1)
2. (11q_0,+1)
3. (111q_1,1)
4. (111q_1,B)
5. (111$q_2$1)
6. (111q_2B)

- **Unsolvability:** Some problems are undecidable. They are not solvable by the Turing Machine and therefore and not solvable by any other computing machine.

An example of undecideable problems is the **Halting Problem**.

III. The Cellular Automata Model

- **Notable Contributors:** Von Neumann; Conway; Wolfram (1D Cellular automata)



A Two-Dimensional Cellular Automaton

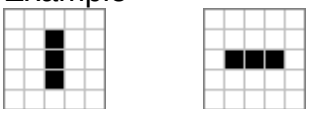
- **The Game of life**

The game of Life is a well-known example of cellular automata, developed by Conway.
It is comprised of a two-dimensional grid of cells. Each cell can be in one of two states, on or off, or alive or dead.
Cells may transmitted from one state to the other, and become dead or alive based on **a set of rules.**

- **Rules of the Game of Life**

Let N be the number of neighbors of a given cell.
If N = 0 or 1, cell dies
If N = 2, the cell maintains its current state (status quo)
If N = 3, the cell becomes alive
If N = 4,5,6,7 or 8, the cell dies.

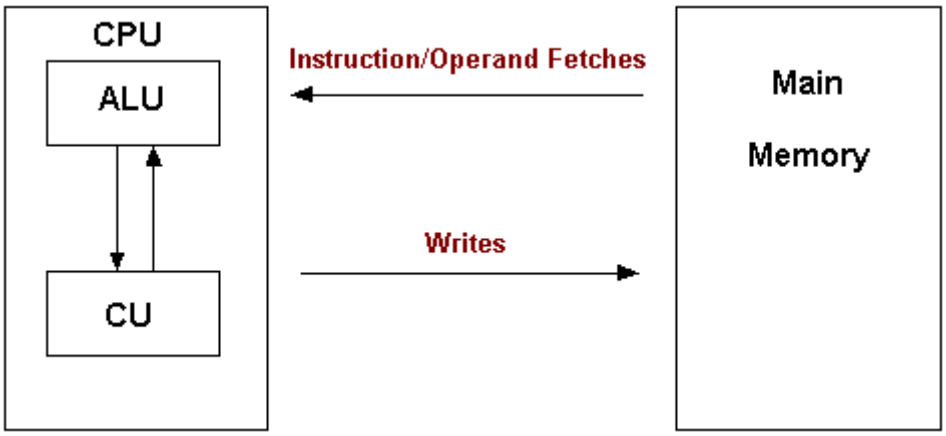
- An Example


at time t at time t+1

- Wolfram studied one-dimensional CA. He enumerated 256 possible rules and 4 complexity classes for such automata.

IV. The Computer Model (Z1, ENIAC, Mark I)

- **Notable Contributors:** Von Neumann; Post; &c



A Von Neumann Machine Schematic

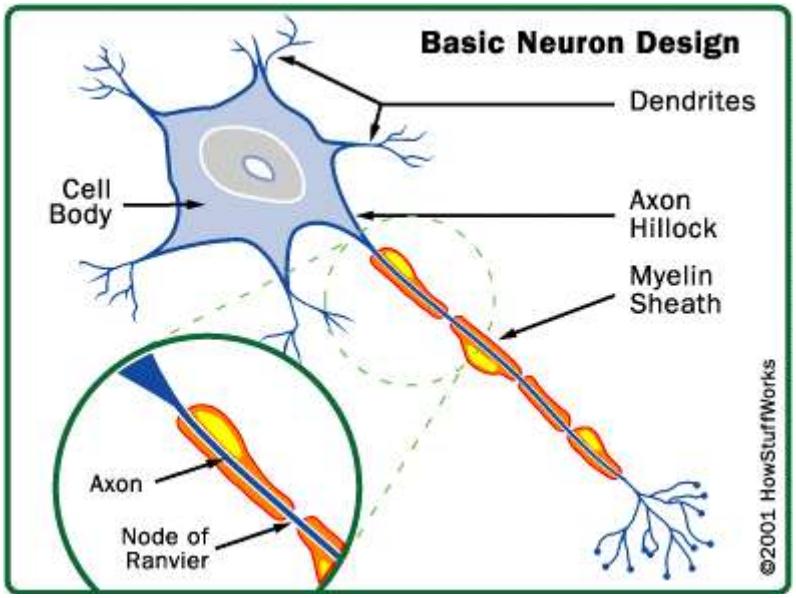
V. The Biological Model (Neural Networks)

- **Notable Contributors:** McCullloch; Pitts; Wiener; Minsky; et al.
- Neural networks as a computing model possess the following **properties:**
  - **Operate in parallel**
  - Computing elements (neurons) work in parallel as opposed to the sequential action of Turing machines.

- **Hierarchical Multilayered Structure**
  Information is transmitted not only to immediate neighbors, but also to more distant units, as opposed to CA.

- **No Program is handed over to the hardware**
  The free parameters of the network have to be found adaptively - as opposed to conventional computers (which require pre-programmed algorithms to execute).

- **Structure of the Neuron**



The cell body (soma)	"Processing" occurs here. ·
Dendrites	Protrude from soma. They conduct signals to the cell body
Axon Hillock	Extends from cell body - initial portion of the axon.
Axon	A long fibre - generally splits into smaller branches.
Synapse	The axon-dendrite (axon-soma; axon-axon) contact between an endbulb and the cell it impinges upon is called a synapse. (End bulbs can be seen at the bottom right corner of the above image.)

Animated close-up of a neuron and the synaptic cleft

- **The signal flow in the neuron** is from the dendrites through the soma converging at the axon hillock and down the axon to the endbulbs.

- **A neuron typically has many dendrites** but only a single axon.
- The four elements
    - dendrites
    - synapses
    - cell body
    - axon

  are the **minimal structure** will be adopted from the biological model.

- Artificial neurons will have
    - input channels
    - cell body
    - output channel
    - synapses

  where the synapses will be simulated by contact points between the cell body and input or output connection.

- A **weight** will be associated with these points.

- **Organizational and Computational Principles of the Brain**

Massive parallelism
A large number of simple, slow units are organized to solve problems independently but collectively.

High degree of connection complexity
Neurons have a large number of connections to other neurons and have complex interconnection patterns.

Trainability (Inter-neuron interaction parameters)
Connection patterns and connection strengths are changeable as a result of accumulated sensory experience.

Binary states and continuous variables
Each neuron has only two states: resting and depolarization. However, the variables of the brain are continuous (potentials, synaptic areas, ion and chemical density, &c. ...) and vary continuously in time and space.

There are many types of neurons and signals.

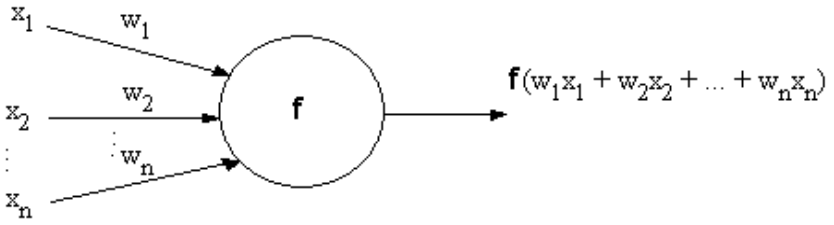
Intricate signal interaction
The interaction of impulses received at a single neuron is highly nonlinear and depends on many factors.

Physical decomposition (Nature's way of dealing with the complexities of the brain itself)
The brain is organized as a mosaic of subnetworks. Each subnetwork consists of several thousand densely connected neurons. These subnetworks are the basic processing modules of the brain. Connections to distant neurons are sparser and with less feedback... Autonomous local collective processing in parallel, followed by a more serial and integrative processing of those local collective outcomes.

Functional decomposition
Each area, or subnetwork is responsible for specific functions.

Artificial Neural Networks

- We may think of artificial neural networks as networks of primitive functions:

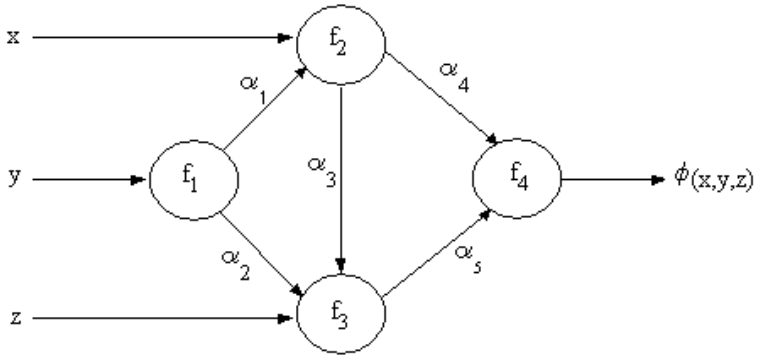


Primitive function f computed in the body of the abstract neuron

- Usually, the input channels have an associated weight

- Different models of ANN's (Artificial Neural Networks) will differ in:

  - Primitive function used
  - Interconnection pattern (topology)
  - Timing of transmission



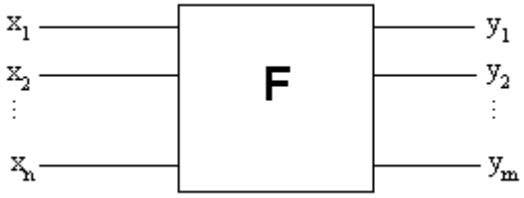
Function model of ANN

- Function ϕ evaluated at the point (x,y,z).

- The nodes implement the primitive functions, f_1, f_2, f_3, f_4, which are combined to form ϕ .
- The function ϕ - is a network function.
- Different weights α_i ... will produce different functions.
- **The key Elements**
  - Structure of the nodes
  - Topology of the network
  - Learning algorithm to find the weights

Threshold Logic

- F: $R^n \rightarrow R^m$
- A neural net as a **black box**:

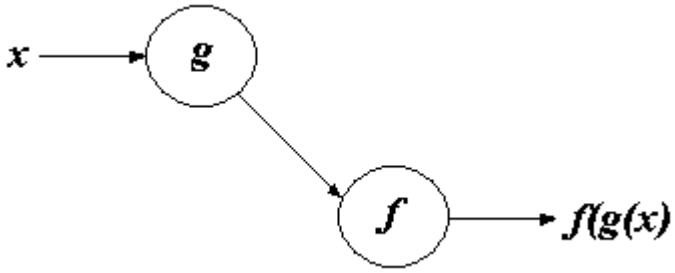


A neural net as a black box

- certain inputs should produce specific outputs.
- How is this done? Through a *self-organizing process*.

Function Composition Vs. Recursion

If no loops, then synchronization is not a problem - Assume no delay.



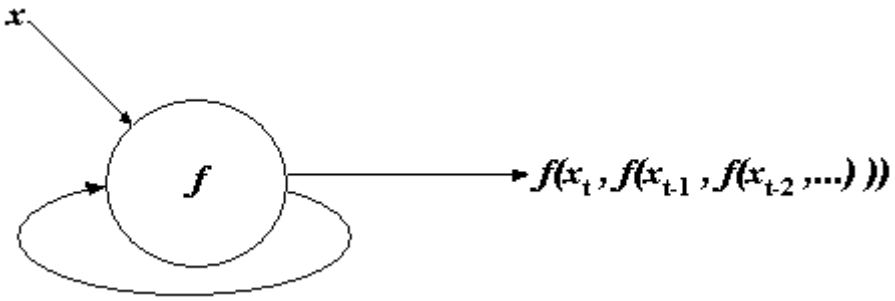
Function Composition

When loops are present, synchronization is required (Δ_t per step)

$f(x_0) = x_1$

$f(x_1) = x_2 = f(f(x_0))$

$f(x_2) = x_3 = f(f(x_1)) = f(f(f(x_0)))$



Recursion

- **Evaluating functions of n arguments**

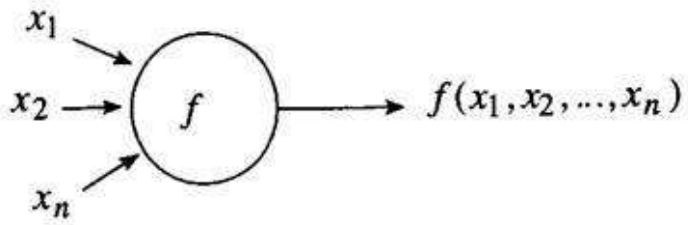


Fig. 2.4. Evaluation of a function of n arguments

  - f is a function of n arguments, which thus has **unlimited fan-in**.

- **Simplifying matters**

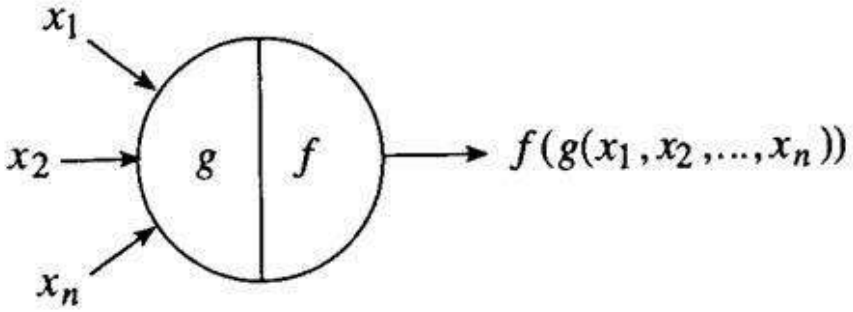


Fig. 2.5. Generic computing unit

  - Here, g is an integration function which reduces the inputs x1, x2, ..., xn to a single argument.

  - f is the output or activation function - it produces the output of this node.

McCulloch-Pitts Networks

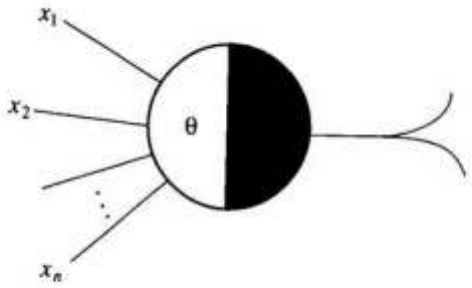


Fig. 2.6. Diagram of a McCulloch–Pitts unit

- Binary Signals - input/output
- Directed, unweighted edges of either an **excitatory** or an **inhibitory** type. (Inhibitory edges are marked with a small circle at the end of the edge that is incident on the unit.)
- Threshold value is θ
- inputs $x_1, x_2, ..., x_n$ come in through n **excitatory** edges.
- Suppose there are also inputs $y_1, y_2, ..., y_m$ coming in through m **inhibitory** edges.

If m >= 1, and at least one of the signals $y_1, y_2, ..., y_m$ is 1, the unit is **inhibited** and thus output = 0.

Otherwise, the total excitation

$\bar{x} = x_1, x_2, ..., x_n$

is computed and compared with the threshold θ .

If $\bar{x} >= \theta$, the unit fires a 1.

- The **Activation Function** for McCullloch-Pitts units, is a step function.

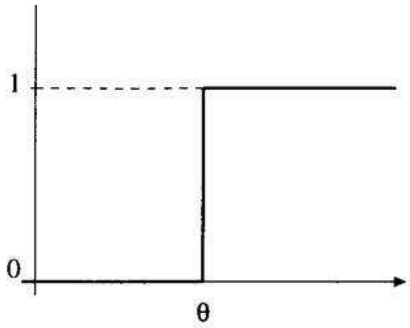


Fig. 2.7. The step function with threshold θ

- McCullloch **-Pitts Units As Boolean Functions**

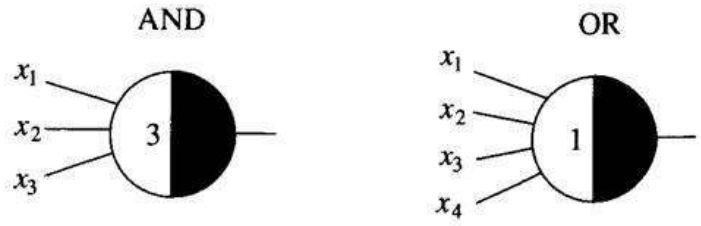


Fig. 2.9. Generalized AND and OR gates

X_1	X_2	$F(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	1
x_1 AND x_2		

X_1	X_2	X*w	$f(x_1,x_2)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1
	x_1 OR x_2		

- What would this picture look like if n=3 in each case?

- **Monotonic Logic Functions**

A monotonic logic function f of n arguments is one whose value at two given n dimensional Points

$x;^{-} = (x_1, x_2, ..., x_n)$ and $y;^{-} = (y_1, y_2, ..., y_n)$

is such that f(x) >= f(y), whenever the number of ones in the input y is a subset of the number of ones in the input x.

An example of a non-monotonic logic function of one argument is
= (1), $y;^{-}$ = (0), and yet f(x) = 0 and f(y) = 1.

- **Proposition**

Uninhibited threshold logic elements of the McCullloch-Pitts type can only implement monotonic logic functions.

- Implementation of some non-monotonic logic functions:

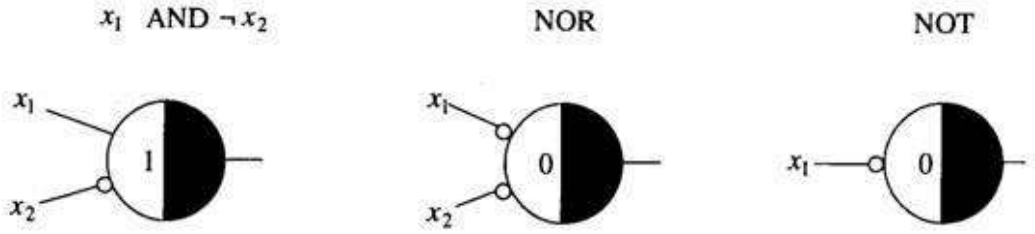


Fig. 2.10. Logical functions and their realization

X_1	X_2	$\sum X_i$	NOR
0	0	0	1
0	1	-	0
1	0	-	0
1	1	-	0

X_1	X_2	$\sum X_i$	NAND
0	0		1
0	1		1
1	0		1
1	1		0

- **Geometric Interpretation**

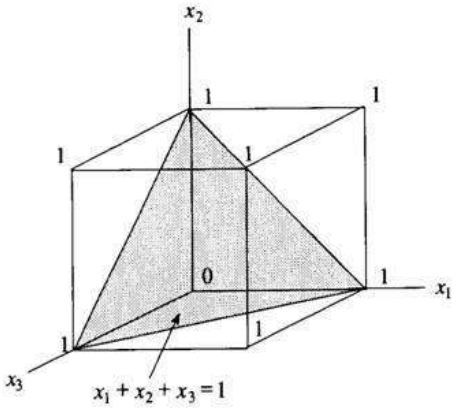


Fig. 2.12. Separation of the input space for the OR function

A McCullloch-Pitts unit divides the input space into two half-spaces.

For a given input (x_1, x_2, x_3), and threshold θ , $x_1 + x_2 + x_3$ is tested *true* for all points to one side of the plane with equation $x_1 + x_2 + x_3 = \theta$.

- **Decoder** for the vector (1,0,1)

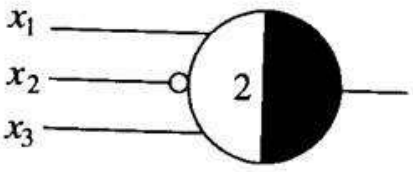
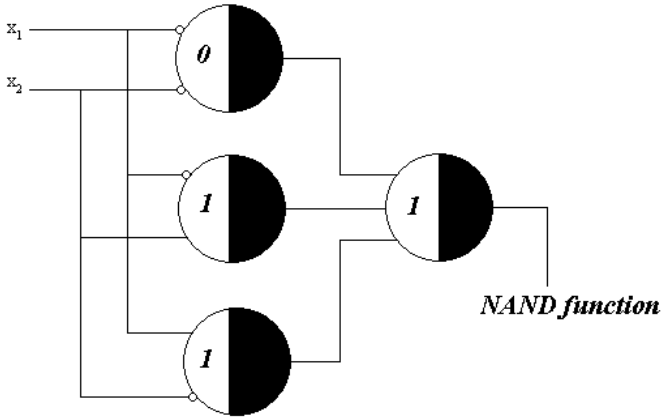


Fig. 2.14. Decoder for the vector $(1, 0, 1)$

Only this input will cause the neuron to fire.

- **Constructing a circuit for the NAND function**

input	F
(0,0)	1
(0,1)	1
(1,0)	1
(1,1)	0



- **Proposition**
  Any logic function F:$\{0,1\}^n$ → {0,1} can be computed with a McCullloch-Pitts network of two layers.
- McCullloch-Pitts networks do *not* use weighted edges.
- **Weighted Edges vs. more Complex Topology**
  Compare the following two figures:
  A unit with weighted edges: $0.2x_1 + 0.4x_2 + 0.3x_3 >= 0.7$

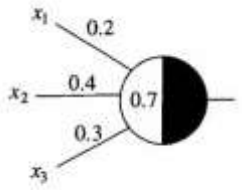


Fig. 2.17. Weighted unit

An equivalent unit with a more complex topology: $2x_1 + 4x_2 + 3x_3 >= 7$.

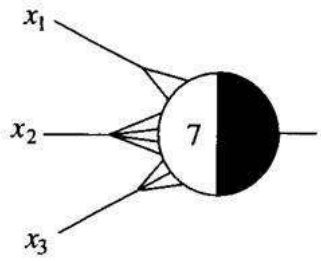
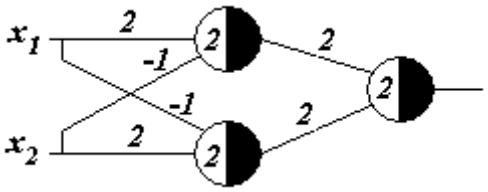


Fig. 2.18. Equivalent computing unit

- Networks with unweighted edges are equivalent to networks with weighted edges.

- **McCullloch-Pitts Networks with Weighted Edges**
  An example of a network for XOR:

X_1	X_2	z_1	z_2	XOR
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	1	1	0



Neural Networks - Application Areas Classification - Mapping input to different classes.

Examples:

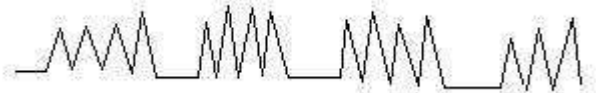
- Two-input Boolean functions

AND (0, 0) 0
(0, 1) 0 0 - class
(1, 0) 0
(1, 1) 1 1 - class

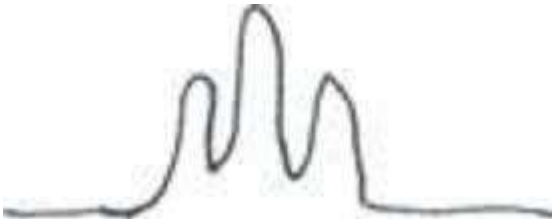
- Recognizing printed or handwritten character:
  { a, a, a } → a - class

{ z, z, z } → z - class

- Analyzing radar data to determine signal source:



→ Helicopter



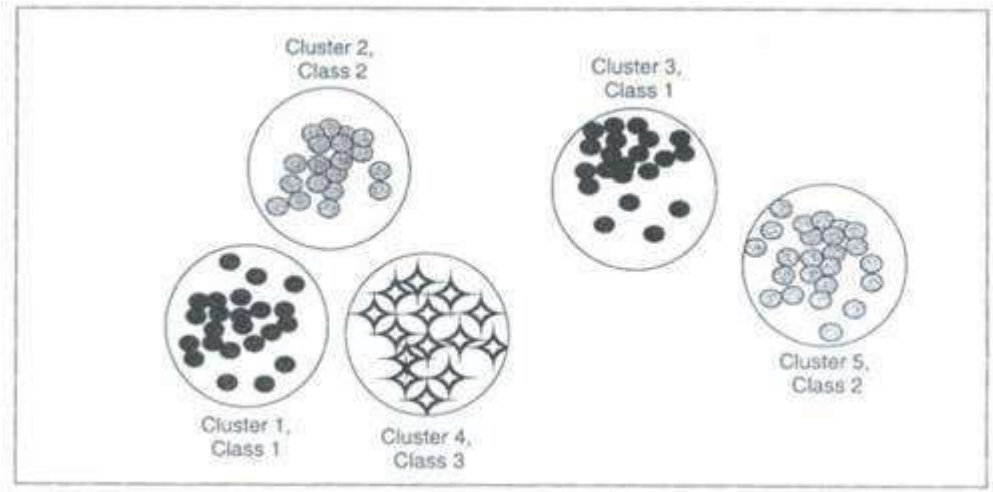
→ Plane

Clustering

Clustering : Grouping together objects that are similar to each other.

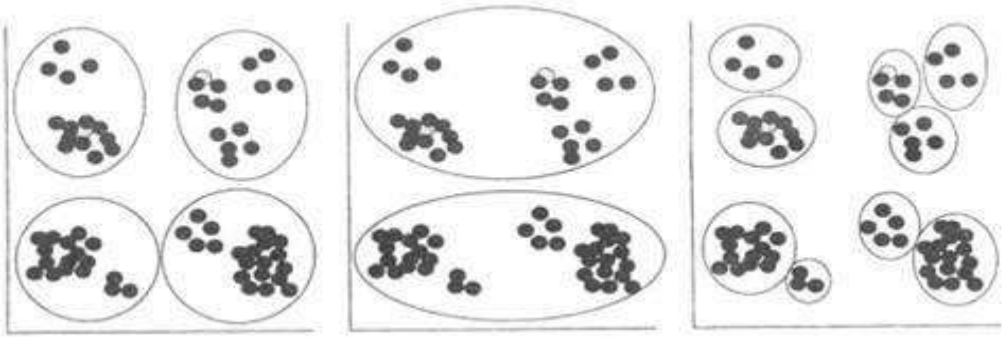
- In clustering problems a priori identification of classes is not known. Instead one has set of samples and distance relationships that can be derived from sample descriptions.

- Samples are to be grouped so as to:
    - minimize intra - cluster distances
    - while maximizing inter- cluster distances



Five clusters, three classes in two-dimensional input space

- The number of clusters depends on the problem but should be as small as possible



Reasonable number of clusters Small number of clusters Too many clusters

Three different ways of clustering the same set of sample points

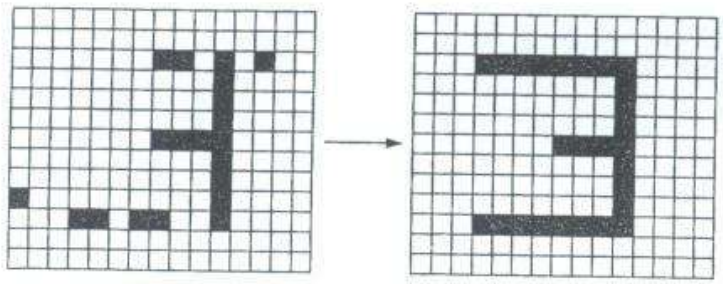
- Initially - each node reacts randomly to input samples.

- Nodes with higher outputs to a particular input sample, learn to react even more strongly to that sample and to other samples that are near it.

Pattern Association

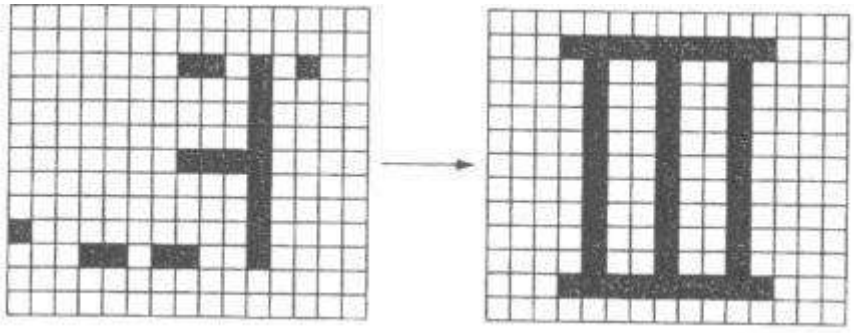
Pattern-Association : The presentation of an input sample triggers the generation of a specific output pattern.

Auto-Association (associative memory tasks) : The input is presumed to be a corrupted, noisy or partial version of the desired output pattern.



Auto-Association

Hetero-Association : The output pattern may be any arbitrary pattern that is to be associated with a set of input patterns

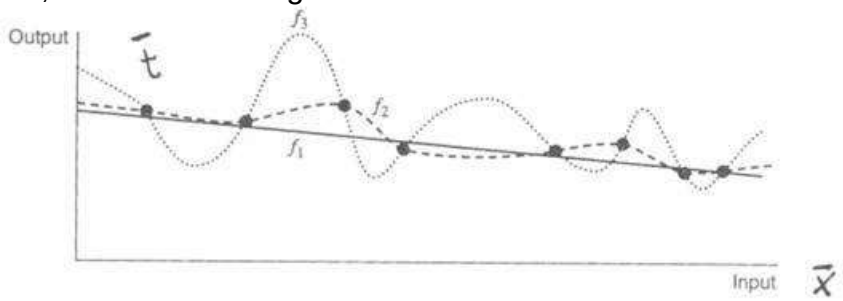


Hetero-Association

Example: The generation of a name when the image of a face is presented.

Function Approximation

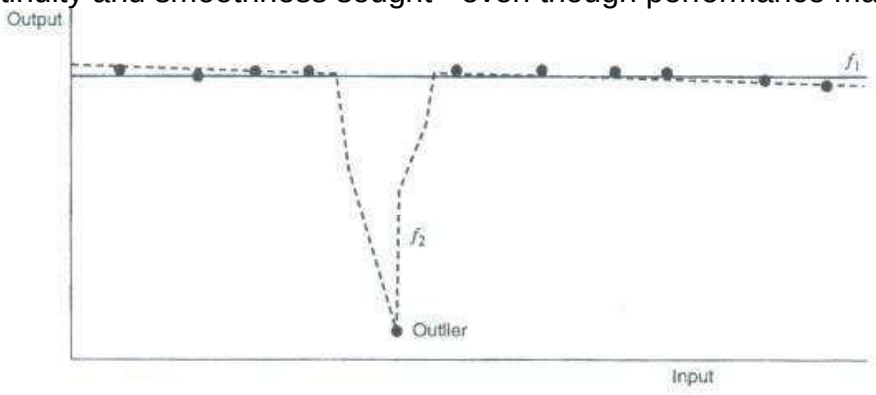
- Function approximation: learning or constructing a function that generates approximately the same outputs from input vectors as the process being modeled, based on training data.



Function approximation: different networks implement function that the same behavior on training samples.

- Many different functions exist that pass through the given pairs x-bar : t-bar in the training set.

- Continuity and smoothness sought - even though performance may suffer.



Presence of an outlier function approximation.

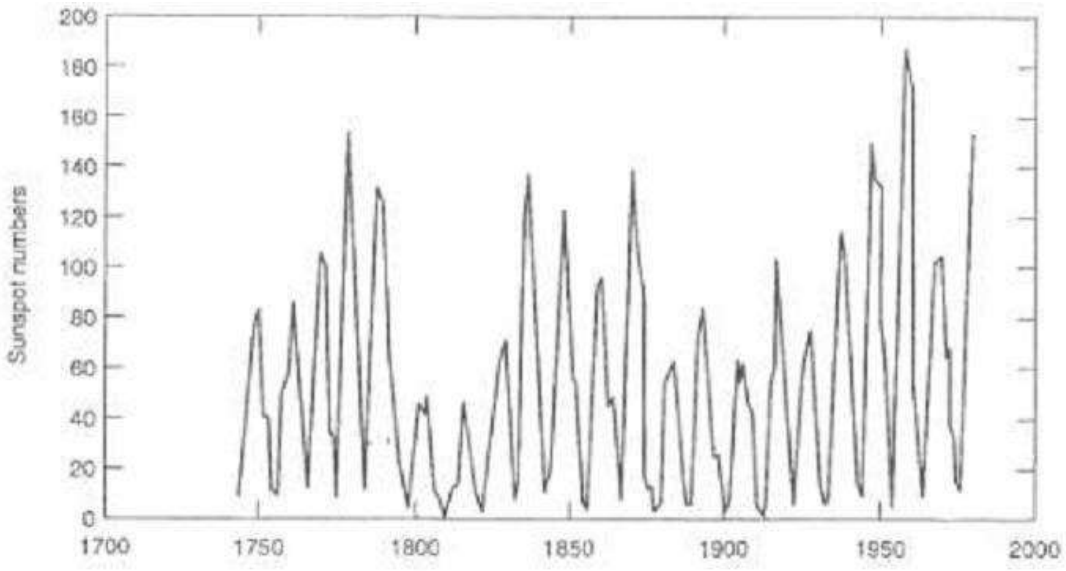
Forecasting

Forecasting: Predicting future events based on past history… the stock market.

Prediction depends on:

- Knowledge of underlying laws… e.g. F = ma.

- Discovery of strong empirical regularities in observations of a given system. However:
    - difficult to discover
    - often masked by noise

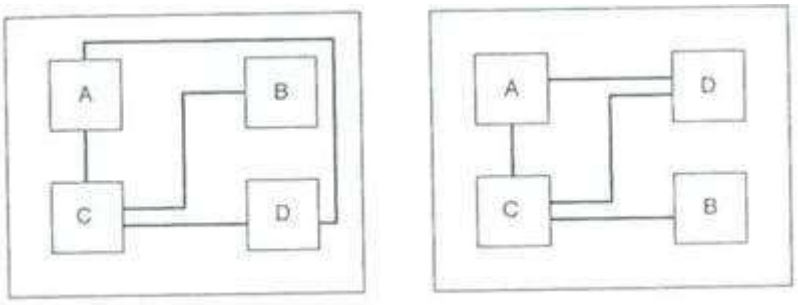


Yearly sunspot numbers, showing cyclic fluctuations.
Discovering the 11- year cycle in sunspot activity.

Time Series employed a sequence of values measured over time. E.g. Dow Jones closing prices from September 11, 2001 to present.

Optimization

Optimization: Maximize or minimization some function subject to some constraints.
Example: Arranging components on a circuit board such that total wire length is minimized. The constraints here are that certain components must be connected to certain others



Two layouts of components on a circuit board with different wire lengths, but with the same inter-component connections.

Other Application Areas:

- Search… e.g. game playing
- Control
  - Robotic movement
  - Pole balancing
  - Truck backing up

- Vector quantization
  - Clustering

Artificial Neural Networks
Lecture Notes

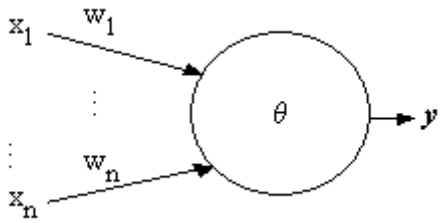
Chapter 2

Contents

- Threshold Logic
    1. Threshold Logic Units
    2. Resilience To Noise and Hardware Failure
    3. Non-Linear Systems
    4. Geometric Interpretation of TLU action
    5. Vectors
    6. TLU's and Linear Separability Revisited

Threshold Logic

- **Threshold Unit or Threshold Logic Unit (TLU)**



TLU

- Activation a = $w_1 x_1 + w_2 x_2 + w_n x_n$

  Example: $\mathbf{w}$ = (0.5, 1.0, 1.2) ,
  $\mathbf{x}$ = (1, 1, 0) ,

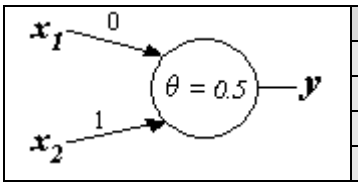
  then $\mathbf{w} * \mathbf{x}$ = (0.5 *1) + (1.0 * 1) + (1.2 * 0)
  = 0.5 + 1.0
  = 1. 5

- To emulate the generation of action potentials, we need a threshold value θ.

$$y = \begin{cases} 1 \text{ if } a >= \theta \\ 0 \text{ if } a < \theta \ . \end{cases}$$

- In the previous example, if $\theta = 1$, then y would equal 1 (because a is >= 1).

- **Resilience To Noise and Hardware Failure**
  - Consider the TLU shown below, and whose behavior is shown in the table to the right. The TLU fires, or outputs one only when x_2 is 1:



X_1	X_2	activation	Output
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

- Now, suppose the hardware which implements the TLU is faulty, thus giving us weights encoded as (0.2, 0.8). The revised table shown above would then be as follows:

X_1	X_2	activation	Output
0	0	0	0
0	1	0.8	1
1	0	0.2	0
1	1	1	1

  - In the revised table above, we see that the activation has changed, but the output is the same. Thus:
  - **Changes in the activation, as long as they don't cross the threshold, produce no change in the output.**

- **Non-Linear Systems**
  - In a linear system, the output is proportionally related to the input: i.e., small/large changes in the input always produce corresponding small/large changes in the output.
  - Non-linear systems do not obey a proportionality restraint so that the magnitude of change in output does not necessarily reflect that of the input.

  - **Example**: In the above TLU, consider a case where an activation change from 0.0 to 0.2 produces no change in the output, whereas an activation change from 0.49 to 0.51

produces change in the output from 0 to 1. This is because 0.51 is larger than the threshold 0.5.

- Similarly, if input signals become degraded in some way, due to noise or a partial loss, once again, correct outputs occur.

- **Example**: instead of input value 1, we have 0.8, and instead of input 0, we have 0.2. Our revised table appears below:

X_1	X_2	activation	Output
0.2	0.2	0.2	0
0.2	0.8	0.8	1
0.8	0.2	0.2	0
0.8	0.8	0.8	1

Non-linear behavior of TLU's

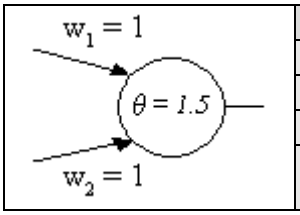
- Thus, TLU's are robust in the presence of noisy or corrupted input signals.

- **Graceful degradation:** In a large network, as the degree of hardware and/or signal degradation increases, the number of TLU's giving incorrect results will gradually increase as well.

  Contrast this with what happens in the case of conventional computers.

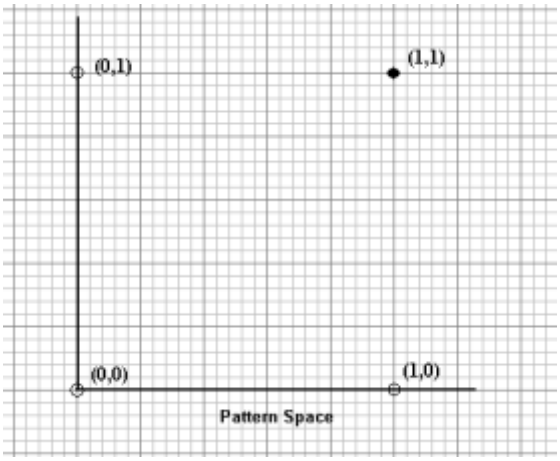
- **Geometric Interpretation of TLU action**
  - A TLU separates its input patterns into two categories according to its binary response (0 or 1) to each pattern.

  - Consider the TLU shown here which classifies its input patterns into two groups: those that give output 0, and those that give output 1.



X_1	X_2	activation	Output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

  - The **pattern space** of this TLU is as follows:



- **The Linear Separation of Classes**

  Inputs: x_1, x_2

  Activation: a

  Threshold: θ.

- The critical condition for classification occurs when the activation equals the threshold
- we have:

  $w_1 x_1 + w_2 x_2 = \theta$

  Subtracting $w_1 x_1$ from both sides

  $w_2 x_2 = - w_1 x_1 + \theta$

  Dividing both sides by w_2 gives

  $x_2 = - (w_1 / w_2) x_1 + (\theta / w_2)$.

  This is of the general form:

  $y = m x + b$

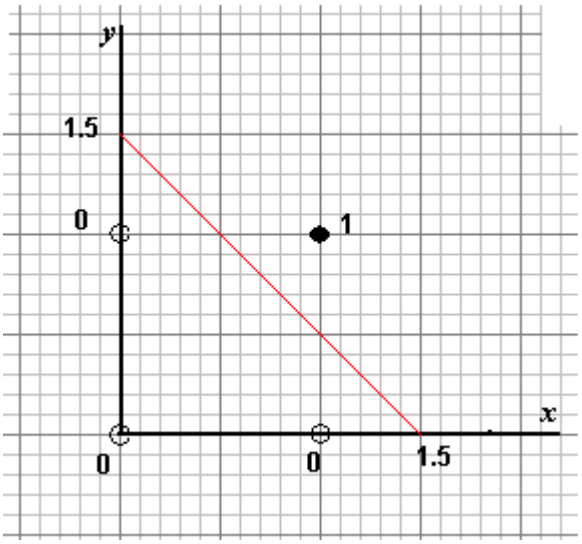
  Recall that this is the linear (line) equation where m is the slope, and b is the y-intercept.

- Our previous example revisited:

  $w_1 = 1$, $w_2 = 1$, $\theta = 1.5$.
  then $m = - (w_1 / w_2) = -1$,
  $b = (\theta / w_2) = 1.5 / 1 = 1.5$.
  $y = -x + 1.5$

x	y

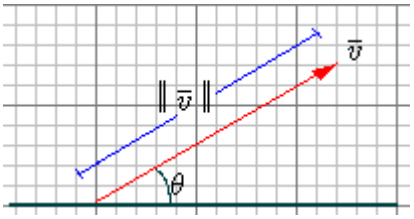
1.5	0
0	1.5



The two **classes** of the TLU output are separated by the red line. This will always be the case - in R^n we will have separating hyperplanes.

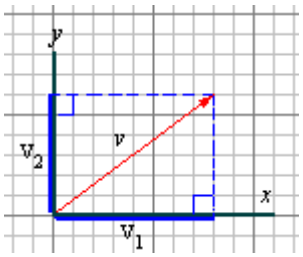
- **TLU's are linear separators** and **their patterns are linearly separable**.

- **Vectors**
  - A vector has a magnitude and a direction.
  - We denote a vector **v** with $\mathbf{v}\bar{}$, (in Europe, **v** .)

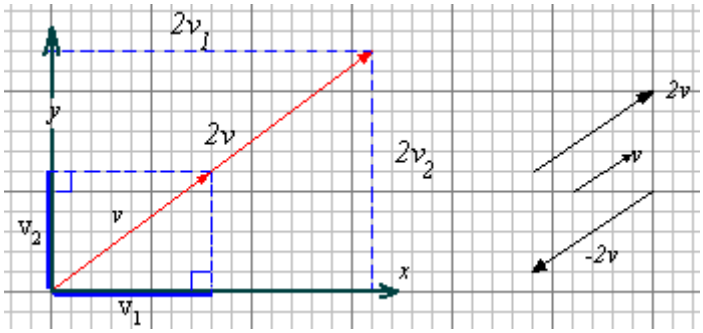


  - We denote a vector's magnitude with $\|\mathbf{v}\bar{}\|$; sometimes with r.
  - A vector is defined by a pair of numbers ($\|\mathbf{v}\bar{}\|$, θ), where θ is the angle the vector makes with some reference direction (e.g., the x-axis).
  - Alternate Representation - Cartesian co-ordinate system.

    For example, $\mathbf{v}\bar{}$ = (v_1, v_2), where v_1, v_2 are the components of the vector.



- With the above representation, the vector can now be considered as an ordered pair, or more generally, an ordered list of numbers - note that the **order** is important. (e.g., in general, $(v_1, v_2) \neq (v_2, v_1)$.)
- Scalar Multiplication of a vector:



- Generalizing to n dimensions, we have $k\mathbf{v}^{-}$
  $= (kv_1, kv_2, \ldots, kv_n)$.
- Vector Addition:
  - Two vectors may be added in 2D by appending one to the end of the other. Note that the vector may be drawn anywhere in the space as long as its magnitude and direction are preserved.
  - In terms of its components, if $\mathbf{w}^{-} = \mathbf{u}^{-} + \mathbf{v}^{-}$, then
    $w_1 = u_1 + v_1$
    $w_2 = u_2 + v_2$.
  - In n-dimensions, we have:
    $\mathbf{w}^{-} = \mathbf{u}^{-} + \mathbf{v}^{-}$
    $\mathbf{w}^{-} = (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n)$.
  - Note: Addition is commutative.
    (i.e., $\mathbf{u}^{-} + \mathbf{v}^{-} = \mathbf{v}^{-} + \mathbf{u}^{-}$)
  - Vector Substraction
  - The Length of a Vector
    - in 2D, $\|\mathbf{v}^{-}\| = \text{sqrt}\,(v_1^2 + v_2^2)$.

- in n-dimensions, $\|\mathbf{v};^{-}\|$ = sqrt($\sum v_i^2$), with i running from 1 to n.

o Comparing Vectors

1. The Inner Product - Geometric Form:

- Useful to have some measure of how well aligned two vectors are. i.e., to know whether they point in the same or opposite direction.
- Inner Product:

  $\mathbf{v};^{-} \cdot \mathbf{w};^{-} = \|\mathbf{v};^{-}\| \, \|\mathbf{w};^{-}\| \cos\theta$

  where θ is the angle between the two.

  This product is also known as the **scalar product**. Note that if θ is > 360° we have equivalence to θ-360 .

2. The Inner Product - Algebraic Form:

- **Given** $\mathbf{v};^{-}$ =(1,1), $\mathbf{w};^{-}$ = (2,0), (note the angle between them is 45°,)
- observe that $\|\mathbf{v};^{-}\|$ = sqrt(1^2+1^2) = sqrt(2), and that

  $\|\mathbf{w};^{-}\|$ = sqrt(2^2+0^2) = 2.

  Hence $\mathbf{v};^{-} \cdot \mathbf{w};^{-}$ = sqrt(2) * 2 * (1/sqrt(2)) = 2.
- Alternatively, we have

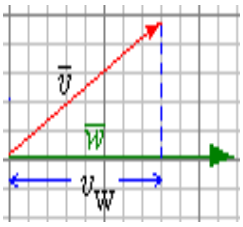
  $\mathbf{v};^{-} \cdot \mathbf{w};^{-} = v_1 w_1 + v_2 w_2$

  = 1*2 + 1*0 = 2.
- This is also called the **dot product**.
- In n-dimensions, we have $\mathbf{v};^{-} \cdot \mathbf{w};^{-} = \sum v_i w_i$, with i running from 1 to n.
- If the sum is positive, then the vectors are lined up to some extent.

  If the sum is zero, then the vectors are at right angles.

  If the sum is negative, then the vectors are pointing away from each other.

Vector Projection



How much of $\mathbf{v};^{-}$ lies in the direction of $\mathbf{w};^{-}$?

Drop a perpendicular projection v_w of $\mathbf{v};^{-}$ onto $\mathbf{w};^{-}$, where

$v_w = \|\mathbf{v};^{-}\| \cos\theta$.

- Alternatively, we may write:

$$v_w = (\|\mathbf{v};^{-}\| \, \|\mathbf{w};^{-}\| \cos\theta) / \|\mathbf{w};^{-}\|$$
$$= (\mathbf{v};^{-} . \mathbf{w};^{-}) / \|\mathbf{w};^{-}\| .$$

○ TLU's and Linear Separability Revisited

- The activation **a** of an n-input TLU is given by $a = \mathbf{w};^{-} . \mathbf{x};^{-}$.
- What happens when the activation equals the threshold?

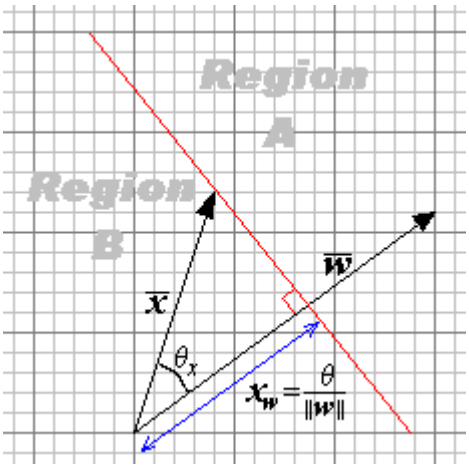
  Let n = 2. We have

$$w_1x_1 + w_2x_2 = \theta$$

  i.e., $\mathbf{w};^{-} . \mathbf{x};^{-} = \theta$ (*)

- For an arbitrary $\mathbf{x};^{-}$, the projection of $\mathbf{x};^{-}$ into $\mathbf{w};^{-}$ is:

$$x_w = \mathbf{w};^{-} . \mathbf{x};^{-} / \|\mathbf{w};^{-}\| .$$



- If the constraint in (*) is imposed, we have: $x_w = \theta / \|\mathbf{w};^{-}\|$.

  (For example: consider when x = 2 , when y = 3)

- So, assuming $\mathbf{w};^{-}$ and θ are constant, the projection x_w is constant and, in the case of 2D, x_w must lie along the perpendicular to the weight vector.
- Therefore, in 2D, the relation $\mathbf{w};^{-} . \mathbf{x};^{-} = \theta$ defines a straight line. Generalizing to n-dimensions, the counterpart is a hyperplane.
- When x lies on the hyperplane: $\mathbf{w};^{-} . \mathbf{x};^{-} = \theta$ and hence y = 1.
- Suppose $x_w > \theta / \|\mathbf{w};^{-}\|$, then x must lie in region A, as $\mathbf{w};^{-} . \mathbf{x};^{-} > \theta$ and hence y = 1.
- If $x_w < \theta / \|\mathbf{w};^{-}\|$, then the projection is shorter, and x must lie in region B.

- TLU is a linear classifier. If patterns cannot be separated by a hyperplane, then they cannot be classified with a single TLU.

Artificial Neural Networks
Lecture Notes

Chapter 3

Contents

Threshold Logic Units (Cont'd)

- **Recurrent Networks**
  - Previously, we spoke of feed-forward networks. By Contrast,
  - **Recurrent Networks** are those whose partial computations are recycled through the network itself.

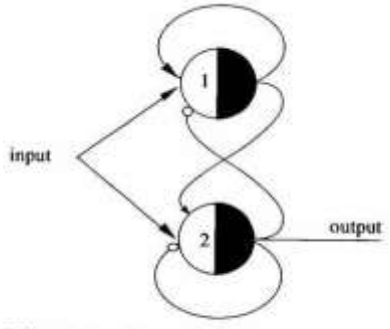


Fig. 2.21. Network for a binary scaler

.

  - Assumption: Computation of the activation of each unit consumes one time unit, Δt.
  - In fig. 2.21, The network takes a string in binary as input, and turns any sequence '11' into '10'. That is, any two consecutive ones are transformed into the sequence 10.

For example: S = 00110110
R = 00100100

- **The Mealy Model of Finite Automata**

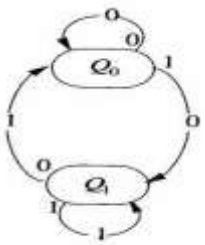


Fig. 2.23. Diagram of a finite automaton

- **The Mealy Model of Finite Automata**
  - Refer to fig. 2.23
  - The finite automaton may be formally described as a structure

    $M_t = <Q, S, R, f, g, q_I>$ where,

Q:	is the set of finite states of the FA
S	is the Input alphabet
R	is the output alphabet
f	is the next-state function f: Q x S → Q
g	is the output function g: Q x S → R
q_I	is the Initial state of the machine (the state it is in, before receiving input)

  - The machine M_t acts as a time-delay machine, that outputs at time t+1 the input given to it at time t.
  - We can write: $x \in S^* \rightarrow \boxed{M_t} \rightarrow y \in R^*$.

The Perceptron

On McCulloch-Pitts Units

- McCulloch-Pitts units are similar to logic gates.
- They have no free parameters that can be adjusted.
- Learning can only be implemented by modifying the network topology and thresholds, which is more complex than merely adjusting weights.

The Perceptron

Frank Rosenblatt developed the perceptron model in 1958, which features

- Numeric weights, and

- a special interconnection pattern.

In the original model

- Units are threshold elements,
- Connectivity is determined stochastically.

In the 1960's, Minsky & Papert refined and perfected this model.

See figure 3.1, The Classical Perceptron.

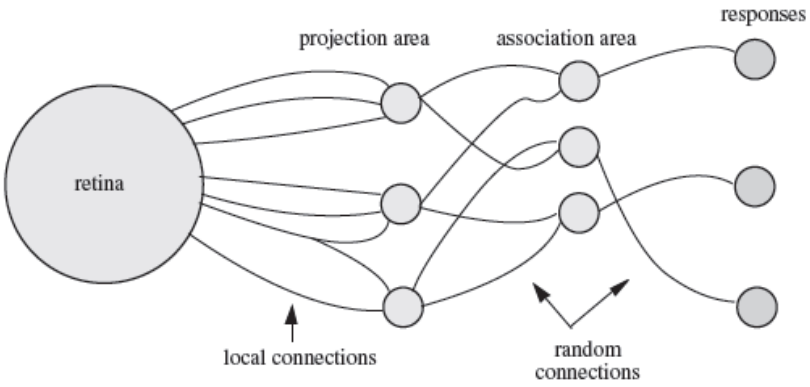
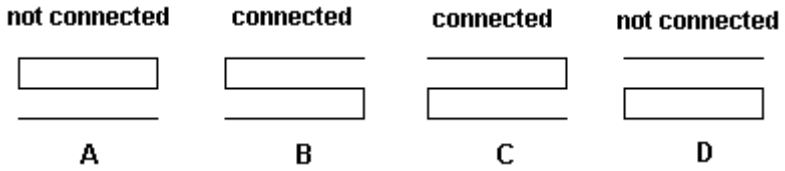


Fig. 3.1. The classical perceptron [after Rosenblatt 1958]

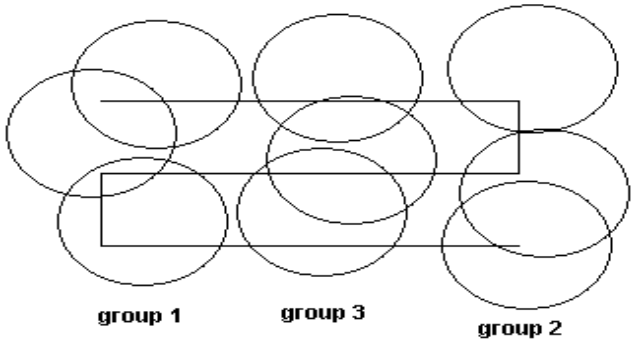
Limitations of the Perceptron Minsky & Papert studied the limitations of various models of the perceptron. Example: Diameter Limited Perceptrons: The receptive field of each predicate has a limited diameter.

Proposition 3.1.1 (Rojas, p. 58) No diameter limited perceptron can decide whether a geometric figure is connected or not.



Proof By Contradiction Diameters of receptive fields are limited - hence no single receptive field contains points from both the left and the right ends of the patterns.

Assume that we have three different groups of predicates



Threshold element performs the computation:

$S = \sum_{Pi \in Group\ 1} w_{1i}P_i + \sum_{Pi \in Group\ 2} w_{2i}P_i + \sum_{Pi \in Group\ 3} w_{3i}P_i - \theta >= 0.$

If S >= 0, then figure is recognized as connected.

If S < 0, then figure is recognized as disconnected.

- If A changed to B - weights in group 2 must increase.
- If A changed to C - weights in group 1 must increase.

However - What if A changed to D?

D is indistinguishable from C in group 1, and

D is indistinguishable from B in group 2

Hence D would be categorized as connected which it is not.

Training TLU's: The Perceptron Rule

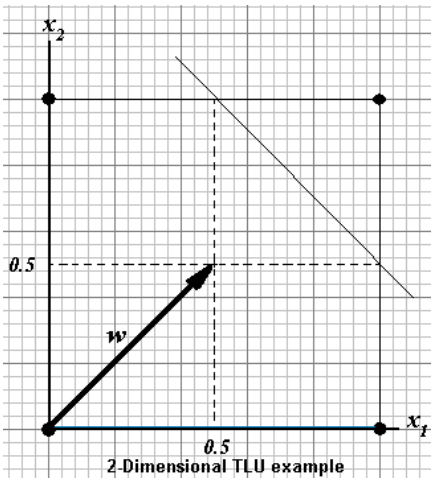
- In order for a TLU to perform a given classification it must have the desired decision surface.
- This is determined by the weight vector and the threshold.
- Hence, it is necessary to adjust the weights and threshold. This is usually done via an iterative procedure.
- At each presentation, small changes are made to weights and thresholds.
- This process is known as **training the net**, and the set of examples is **the training set**.
- From the network's viewpoint, it undergoes a process of learning, or adapting to the training set, and the prescription for how to change the weights at each step is **the learning rule**.

$\mathbf{w}\bar{;} \cdot \mathbf{x}\bar{;} >= \theta$	Condition for an output of 1. i.e. :
$\mathbf{w}\bar{;} \cdot \mathbf{x}\bar{;} - \theta >= 0$	or:
$\mathbf{w}\bar{;} \cdot \mathbf{x}\bar{;} + (-\theta) >= 0$	we may think of the threshold as an extra weight tied to an input set to -1.

- Bias: The negative of the threshold.
- Weight Vector: Originally of dimension n
- Augmented Weight Vector: (n+1)-dimensional vector, w1, w2, w3, ..., wn,θ
- Thus,

$\mathbf{w}\bar{;} \cdot \mathbf{x}\bar{;} >= 0$	→ y = 1
$\mathbf{w}\bar{;} \cdot \mathbf{x}\bar{;} < 0$	→ y = 0

- A two-input TLU that outputs a "1" with input (1,1), and "0" for all other inputs:



Suitable (non-augmented) weight vector (1/2, 1/2) with threshold 3/4.

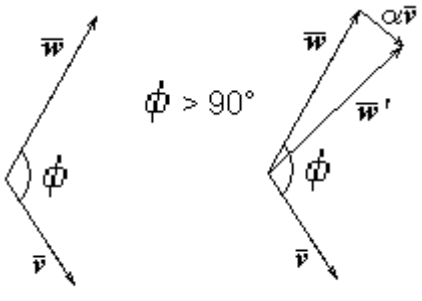
The decision line goes through the poix1 = (1/2 , 1) and x2 =(1, 1/2)

Since w x_1 = w x_2 = 3/4 = θ.

Adjusting The Weight Vector

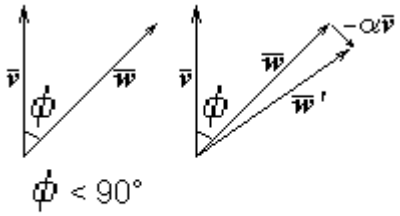
- Training a TLU with augmented weight vector training set: $\bar{v};$ (the input), t (the target).
- Input vector $\bar{v};$ presented to TLU, where the desired response, or target, is t = 1.
- However, it produces an output y = 0).
- TLU has thus **misclassified** - and we must adjust the weights!
- To produce a "0", the activation must have been negative when it should have been positive.

  $\therefore$ the dot product $\bar{\mathbf{w}};\cdot\bar{\mathbf{v}};$ was negative, and the two vectors were pointing away from each other.
- Thus, we need to rotate $\bar{w};$ so that it points more in the direction of $\bar{v};$. How do we do this?
- We add a fraction of $\bar{v};$ to $\bar{w};$ (so as not to upset previous learning).
- i.e., $\bar{w};' = \bar{w};+\alpha\,\bar{v};$. (*)



TLU misclassification 1 - 0

- suppose now that target t = 0, but y = 1
- This means the activation was positive when it should have been negative.
- Thus, we need to rotate $\bar{w}$ away from $\bar{v}$, which we can accomplish by subtracting a fraction of $\bar{v}$ from $\bar{w}$.
- Thus, we have



TLU misclassification 0-1

$$\bar{\mathbf{w}}' = \bar{\mathbf{w}} - \alpha\,\bar{\mathbf{v}} \qquad (**)$$

- Combining (*) and (**) we obtain

$$\bar{\mathbf{w}}' = \bar{\mathbf{w}} + \alpha(t - y)\,\bar{\mathbf{v}} . \qquad (***)$$

or

$$\Delta\bar{\mathbf{w}} = \bar{\mathbf{w}}' - \bar{\mathbf{w}}$$

$$\Delta\bar{\mathbf{w}} = \alpha(t - y)\,\bar{\mathbf{v}}$$

or, in terms of components,

$\Delta w_i = \alpha(t - y)v_i$, i = 1 to n+1, where $w_{n+1} = \theta$ and $v_{n+1} = -1$ (always.)

- This is called the **Perceptron Learning Rule** because historically it was first used with perceptrons.

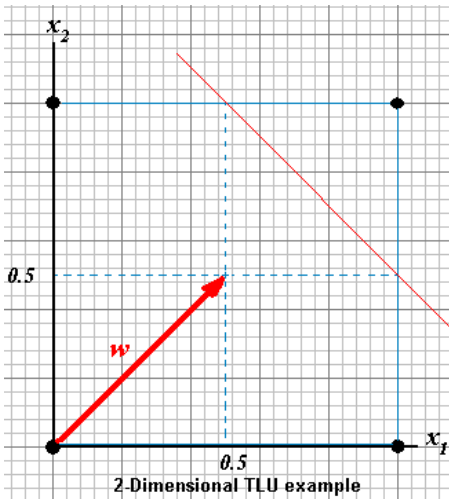
The Perceptron Learning Rule

```
Begin
Repeat
    For each training vector pair (V, t)
        Evaluate the output y when V is input to the TLU
        If y != t Then
            Form a new weight vector W' according to
          (***)
        Else
            Do nothing
        End If
    End For
Until y = t for all vectors
End
```

- The algorithm will generate a valid weight vector for the problem at hand, if one exists.

The Perceptron Convergence Theorem

- If two classes of vectors, X, Y are linearly separable, then application of the perceptron training algorithm will eventually result in a weight vector $\bar{w}_0$, such that $\bar{w}_0$ defines a TLU whose decision hyperplane separates X and Y.
- This theorem was first proven by Rosenblatt in 1962.
- Once $\bar{w}_0$ has been found, it remains stable and no further changes are made to the weights.
- Returning to our previous example,



initial weights are w_1=0 and w_2=0.4

initial threshold θ= 0.3.

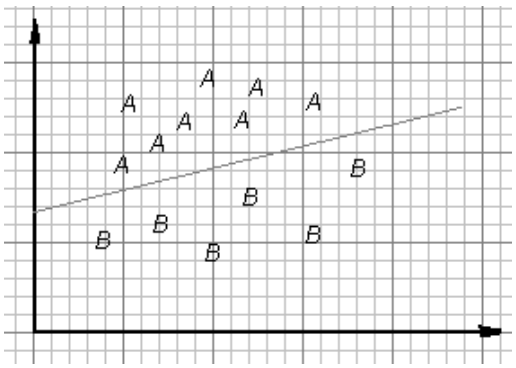
The **learning rate** = 0.25.

- **HOMWORK**: Write a program to implement the Perceptron Learning Rule and solve this problem:

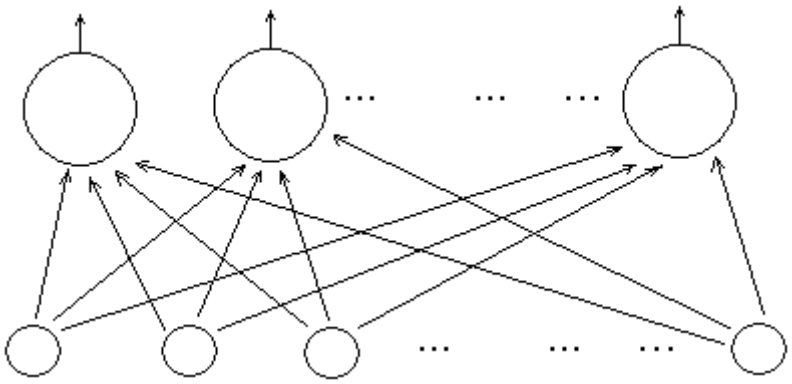
x1	x2	w1	w2	θ	A	Y	t	α(t-y)	Δw1	Δw2	Δθ
0	0	0.00	0.40	0.30	0.00	0	0	0.00	0	0	0
0	1	0.00	0.40	0.30	0.40	1	0	-0.25	0	-0.25	0.25
1	0	0.00	0.15	0.55	0.00	0	0	0.00	0	0	0
1	1	0.00	0.15	0.55	0.15	0	1	0.25	0.25	0.25	-0.25
0	0	0.25	0.40	0.30	0.00	0	0	0.00	0	0	0
0	1	0.25	0.40	0.30	0.40	1	0	-0.25	0	-0.25	0.25
1	0	0.25	0.15	0.55	0.25	0	0	0.00	0	0	0
1	1	0.25	0.15	0.55	0.40	0	1	0.25	0.25	0.25	-0.25
0	0	0.50	0.40	0.30	0.00	0	0	0.00	0	0	0
0	1	0.50	0.40	0.30	0.40	1	0	-0.25	0	-0.25	0.25
1	0	0.50	0.15	0.55	0.50	0	0	0.00	0	0	0
1	1	0.50	0.15	0.55	0.65	1	1	0.00	0	0	0
0	0	0.50	0.15	0.55	0.00	0	0	0.00	0	0	0

0	1	0.50	0.15	0.55	0.15	0	0	0.00	0	0	0
1	0	0.50	0.15	0.55	0.50	0	0	0.00	0	0	0
1	1	0.50	0.15	0.55	0.65	1	1	0.00	0	0	0

Single Layer Nets



- We may use a single perceptron or TLU to classify two linearly separable classes A and B.
- Patterns may have many inputs "cartoon" picture or schematic.
- It is possible to train multiple nodes on the input space to obtain a set of linearly separable dichotomies

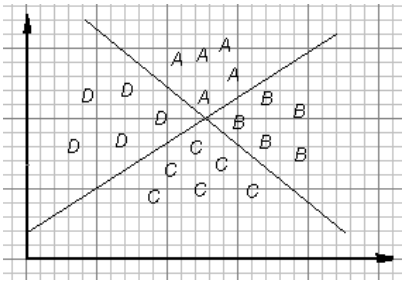


Example: Classifying handwritten alphabetic characters where 26 dichotomies are required, each one separating one letter class from the rest of the alphabet. e.g, A's from Not A's; B's from Not B's; etc.

Non-Linearly Separable Classes

- Four classes, A, B, C and D separated by two planes in pattern space:

  Note: that this is a two-schematic or representation-dimensional space.



- Here, we could not use a single layer net!

  Class A, for example is not linearly separable from the other taken together.

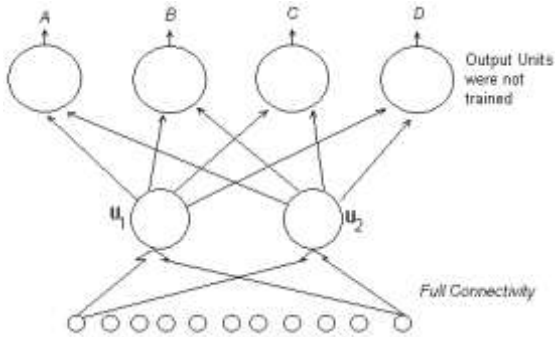
- We can, however "chop" the pattern space into linearly separable regions, and look for particular combinations of overlap within these regions.
- Class of A and B (combined, AB) is linearly separable from C and D together (CD). So are AD and BC.

Class	y_1
AB	1
CD	0
Unit U1	

Class	y_2
AD	1
BC	0
Unit U2	

- If a member of class A is input to each of U_1 and U_2, then $y_1 = 1$, $y_2 = 1$.
- Thus, $y_1 = 1$, $y_2 = 1$ iff input is in A.
- We obtain a unique code for each of the boxes:

y_1	y_2	Class
0	0	C
0	1	D
1	0	B
1	1	A



- Note: The less information we have to supply ourselves, the more useful a network will be.
- Here, we required two pieces of information:

  i) The four classes may be separated by two hyperplanes.

  ii) AB was linearly separable from CD and AD was linearly separable from BC.
- We will need a new training algorithm to accomplish this.

HOMEWORK: Using the Perceptron learning rule, Find the weights required to perform the following classifications:

1. Vectors (1,1,1,1) and (0,1,0,0) are members of the class, and therefore have target value 1.

2. Vectors (1,1,1,0) and (1,0,0,1) are not members of the class, and have target value 0.

Use a learning rate of 1, and starting weights of 0. Using each of the training vectors as input, test the responses of the net.

Artificial Neural Networks
Lecture Notes

Chapter 4

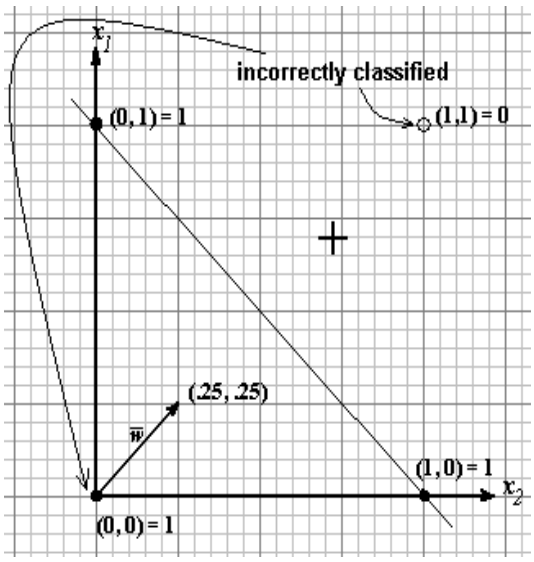
Contents

Perceptron Learning - An example

A Two-Input NAND

x_1	x_2	x_1 NAND x_2
0	0	1
0	1	1
1	0	1
1	1	0

Let $w_1 = w_2 = \theta = 0.25$ to begin.



$\mathbf{w} \cdot \mathbf{x} = \theta$

$w_1 x_1 + w_2 x_2 = \theta$

$x_2 = -(w1 / w_2) x_1 + (\theta / w_2)$

Substituting, we obtain

$x_2 = -(0.25 / 0.25) x_1 + (0.25 / 0.25)$

$x_2 = -x_1 + 1$

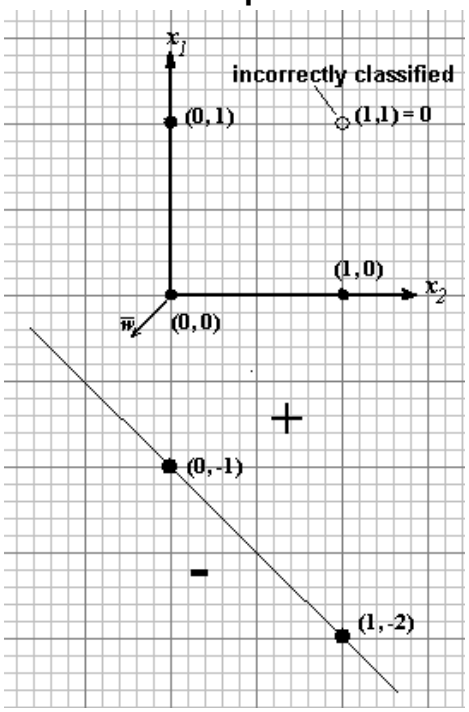
. i.e.,

x_1	x_2
0	1
1	0

First Epoch

w_1	W_2	θ	x_1	x_2	a	y	t	α(t-y)	Δw_1	Δw_2	$\Delta\theta$
.25	.25	.25	0	0	0	0	1	.5(1-0)= .5	0	0	-.5
.25	.25	-.25	0	1	.25	1	1	.5(1-1)= 0	0	0	0
.25	.25	-.25	1	0	.25	1	1	0	0	0	0
.25	.25	-.25	1	1	.5	1	0	.5(0-1)= -.5	-.5	-.5	.5

After the First Epoch



$w_1 = -0.25 = w_2$

$\theta = +0.25$

$x_2 = -(w_1 / w_2)x_1 + (\theta / w_2)$

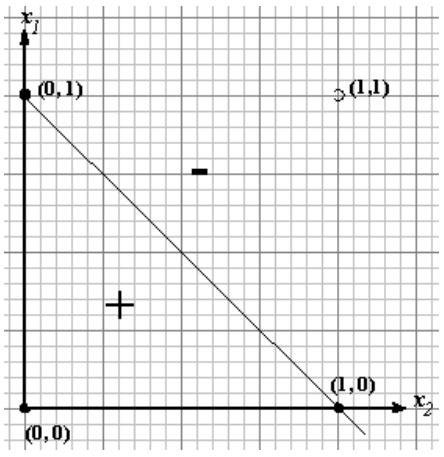
$x_2 = -x_1 - 1$

i.e.

x_1	x_2
0	-1
1	-2

Second Epoch

w_1	w_2	θ	x_1	x_2	a	y	t	α(t-y)	Δw_1	Δw_2	$\Delta\theta$
-.25	-.25	.25	0	0	0	0	1	.5(1-0)=.5	0	0	-.5
-.25	-.25	-.25	0	1	.25	1	1	.5(1-1)=0	0	0	0
-.25	-.25	-.25	1	0	.25	1	1	0	0	0	0
-.25	-.25	-.25	1	1	.5	0	0	.5(0-0)=0	0	0	0



After the second Epoch

$w_1 = -0.25 = w_2$

$\theta = -0.25$

$x_2 = -(w_1 / w_2)x_1 + (\theta / w_2)$

$x_2 = -x_1 + 1$

i.e.

x_1	x_2
0	1
1	0

Third Epoch:

w_1	w_2	θ	x_1	x_2	a	y	t	$\alpha(t-y)$	Δw_1	Δw_2	$\Delta\theta$
-.25	-.25	-.25	0	0	0	1	1	0	0	0	0
-.25	-.25	-.25	0	1	-.25	1	1	0	0	0	0
-.25	-.25	-.25	1	0	-.25	1	1	0	0	0	0
-.25	-.25	-.25	1	1	-.5	0	0	0	0	0	0

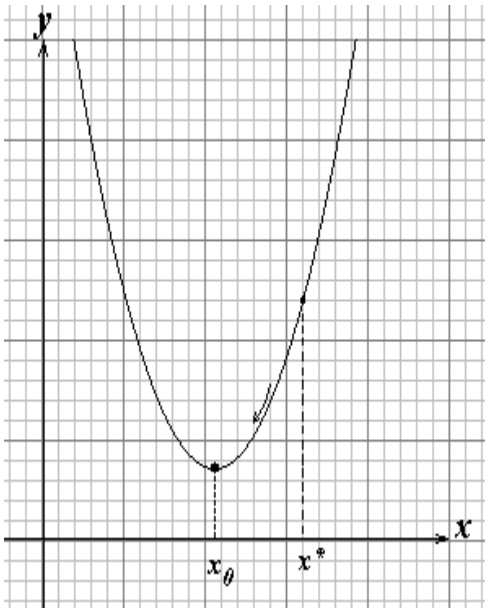
Since there have been no changes, Halt!

The Delta Rule

We desire:

1. Capability to train all the weights in multilayer nets with no a priori knowledge of the training set.
2. Based on defining a measure of the difference between the actual network output and target vector.
3. This difference is then treated as an error to be minimized by adjusting the weights.

Finding the Minimum of a Function: Gradient Descent (informed hillclimbing?)



Suppose that quantity y depends on a single variable x.

i.e., $y = y(x)$.

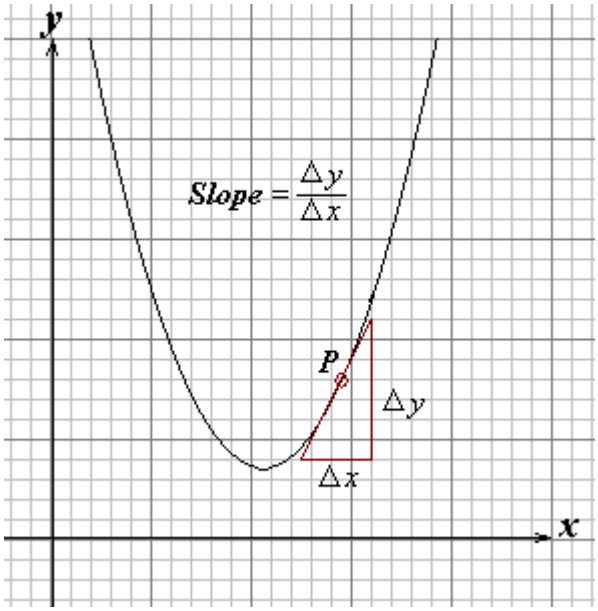
We wish to find x_0 which minimizes x.

i.e.,

$y(x_0) <= y(x)$, $\forall x$.

- Let x^* be current best estimate for x_0.
- To obtain a better estimate for x_0, choose Δx so as to follow the function downhill.
- We need to know the slope of the function at x^*:

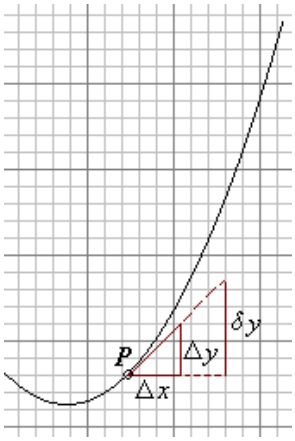
Slope of a Function



The **slope** at any point x is just the slope of a straight line, the **tangent**, which just grazes the curve at that point.

1. Here, one may draw the function on graph paper
2. Draw the tangent at the point P
3. Measure the sides Δx, Δy, or merely calculate:

y'(x = P).



If Δx is small enough, $\delta y = \Delta y$.

Dividing Δy by Δx, and then multiplying by Δx leaves Δy unchanged.

$\Delta y = (\Delta y / \Delta x)\, \Delta x$

Furthermore, $\delta y \approx \Delta y$.

Hence, we may write: δy = slope x. That is,

$\delta y = (dy / dx)\, \Delta x$. (*****)

That is, the derivative of y with respect to x.

Suppose we can evaluate the slope or derivative of y and put

$\Delta x = -\alpha\, (dy/dx)$, where $\alpha > 0$ and is small enough to ensure that $\delta y \approx \Delta y$..

Then, substituting this is in (*****), we get

$dy \approx -\alpha (dy/dx)^2$ (******)

The quantity $(dy/dx)^2$ is positive.

Hence, the quantity $-\alpha\, (dy/dx)^2$ must be negative.

Then $\delta y < 0$.

i.e., we have "traveled down" the curve towards the minimal point.

If we keep repeating steps such as (******), then we should approach the value x_0 associated with the function minimum.

This is **Gradient Descent**.

Its effectiveness hinges on the ability to calculate or make estimates of dy/dx.

Functions of More Than One Variable

Suppose $y = y(x_1, x_2, \ldots, x_n)$.

One may speak of the slope of the function, or its rate of change, with respect to each of these variables independently

The slope or derivative of a function y with respect to the variable x_i is:

$\delta y / \delta x_i$ The partial derivative.

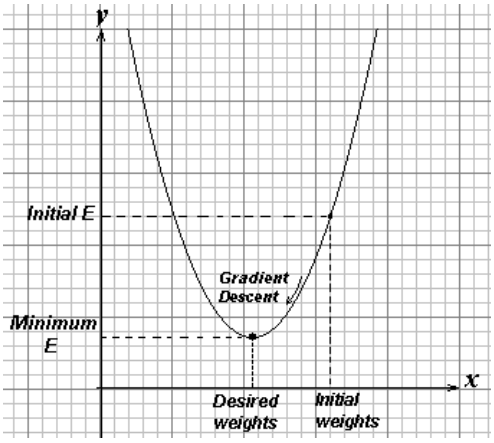
The equivalent is then

$\Delta x_i = -\alpha (\delta y / \delta x_i)$.

There is an equation like this for each variable, and all of them must be used to ensure that $\delta y < 0$ and there is gradient descent.

Gradient Descent on an Error.

- Consider a network consisting of a single TLU.
- Assume supervised learning. i.e., for every input pattern, p, in the training set there is a corresponding target t^p.
- The augmented weight vector, $\bar{w}$, completely characterizes the behavior of the network.
- Any function, E, that expresses the discrepancy between desired and actual network output, may be considered as a function of the weights. i.e., $E = E(w_1, w_2, w_{n+1})$.



The optimal weight vector is found by minimizing this function E by gradient descent.

$\Delta w_i = -\alpha(\delta E / \delta w_i)$.

We need to find a suitable error E.

Suppose we assign equal importance to the error for each pattern, so that if e^p is the error for training pattern p, then the total error E is just the average or mean over all N patterns.

$E = 1/N \ (\sum e^p \ )$ running p = 1 to N

One attempt to define e^p as simply the difference $e^p = t^p - y^p$, where y^p is the TLU output in response to p.

However ... then the error is smaller for $t^{p=0}$, $y^{p=1}$ than for $t^{p=1}$, $y^{p=0}$. <u>They're equally wrong</u>.

We next try

$e^p = (t^p - y^p)^2$.

- <u>A subtle problem remains</u>: With gradient descent, it is assumed that the function to be minimized depends on its variables in a smooth, continuous fashion. First, the activation a^p is simply the weighted sum of inputs. This is smooth and continuous. But, the output depends on a^p via the discontinuous step function.

- One remedy: $e^p = (t^p - a^p)^2$.

  We must be careful how we define the targets. We have used {0,1} heretofore.

  When using the augmented weight vector, the output changes as the activation changes sign i.e.,

  a >= 0 ➔ y = 1.

- hence As long as activation takes on the correct sign, the target output is guaranteed and we are free to choose two arbitrary numbers, one positive, and one negative, as the activation targets.

  {1, -1} are customary.

- One last modification: A factor of 1/2 is added to the error expression - simplifies the resulting slope or derivative.

$\therefore \quad e^p = 1/2 \, (t^p - a^p)^2$.

and thus,

$E = 1/N \; (\sum 1/2 \, (t^p - a^p)^2$ running p = 1 to N

The Delta Rule.

- The Error E depends on all the patterns. So do all its derivatives. Hence, the whole training set needs to be presented in order to evaluate the gradients $\delta E / \delta wi$
- This is **batch training** - results in true gradient descent, but is computationally intensive.
- Instead ... adapt the weights based on the presentation of each pattern individually.

  i.e., we present the net with a pattern p, evaluate e^p / wi, and use this as an estimate of the true gradient $\delta E / \delta w_i$
- Recall that:

  $e^p = 1/2 \, (t^p - a^p)^2$

  and

  $a^p = w_1 x_1^p + w_2 x_2^p + \ldots + w_{n+1} x_{n+1}^p$

  $\delta e^p / \delta w_i = -(t^p - a^p) x_i^p$, where x_i^p is the i^{th} component of pattern p.

  1. The gradient must depend in some way on $(t^p - a^p)$. The larger this is, the larger we expect the gradient to be. If this difference is zero, then the gradient is also zero, since we have found the minimum value of e^p.

  2. The gradient must depend on the input x_i^p, for if this is zero, then the i^{th} input is making no contribution to the activation for the p^{th} pattern - and cannot affect the error. No matter how w_i changes, it makes no difference to e^p.

     Conversely, if x_i^p is large, then the i^{th} input is correspondingly sensitive to the value of w_i.

     $\delta e^p / \delta w_i = -(t^p - a^p) x_i^p$

     use as an estimate ,

     $\Delta w_i = -\alpha(\delta E / \delta w_i)$

     we obtain, $\Delta w_i = -\alpha(t^p - a^p) \, x_i^p$
- <u>Pattern Training Regime</u>: weight changes are made after each vector presentation.

- We are using estimates for the true gradient. The progress in the minimization of E is noisy. i.e., weight changes are sometimes made which increase E.

- This is the **Widrow-Hoff Rule**, now referred to as the Delta Rule (or δ-rule.)

- Widrow and Hoff first proposed this training regime (1960.) They trained ADALINES (ADAptive LINear ElementS,) which is a TLU, except that the input and output signals were bipolar (i.e., {-1,1}.)

- If the learning rate α is sufficiently small, then the delta rule converges. I.e., the weight vector approaches the vector w0, for which the error is a minimum, and E itself approaches a constant value.

- Note: A solution will not exist if the problem is not linearly separable.

- Then w_0 is the best the TLU can do, and some patterns will be incorrectly classified.

- (Note the difference with the Perceptron rule !!!)
- Also note, delta rule will always make changes to weights, no matter how small (because target activation values ± 1 will never be attained exactly.)

The Delta Rule Algorithm

```
Begin
Repeat
    For each training vector pair (V, t)
        Evaluate the activation a when V is input to the TLU
        Adjust each of the weights
    End For
Until the rate of change of the error is sufficiently small
End
```

The Delta Rule - An Example

Train a two-input TLU with initial weights (0, 0.4) and threshold 0.3, using a learning rate α= 0.25. (The AND function)

First Epoch

W_1	W_2	θ	X1	X2	a	t	$\alpha\delta$	δw1	δw2	$\delta\theta$	E
0.00	0.40	0.30	0	0	-0.30	-1.00	-0.17	-0.00	-0.00	**(1)** 0.17	0.24
0.00	0.40	0.48	0	1	-0.08	-1.00	-0.23	-0.00	**(2)** -0.23	**(3)** 0.23	0.43
0.00	0.17	0.71	1	0	-0.71	-1.00	-0.07	**(4)** -0.07	-0.00	**(5)** 0.07	0.04
-0.07	0.17	0.78	1	1	-0.68	1.00	0.42	**(6)** 0.42	**(7)** 0.42	**(8)** -0.42	1.42

After the first epoch, $w_1 = 0.35$, $w_2 = 0.59$, $\theta = 0.36$

We employ $\Delta w_i = + \alpha(t^p - a^p) x_i^p$.

Note the plus sign before : Always travel in the opposite direction of gradient.

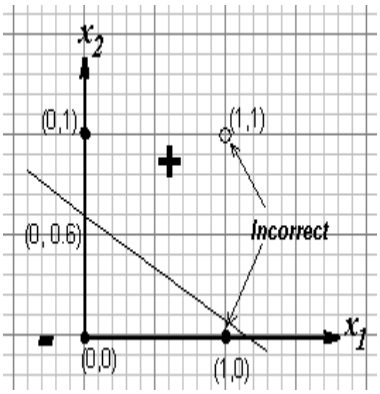
(1) $\delta\theta = +0.25(-1.00 - (-0.30))(-1)$ ← -1 is the input to θ .

$= -0.25(-0.7) = 0.175$ (sign?)!

(2) $\delta w_2 = -0.25(-1.00 - (0.08)) * 1 = -0.25(-0.92) = 0.23$. ((3) will have the opposite sign.)

(4) $\delta w_1 = -0.25(-1.00 - (-0.71)) * 1 = -0.25(-0.29) = 0.07$. ((5) will have the opposite sign.)

(8) $\delta\theta = -0.25(1.00 - (-0.68))(-1) = -0.25\ (1.68) = -0.42$ ((6), (7) opposite sign.)



Since, after the first epoch, we have

$w_1 = 0.35$,

$w_2 = 0.59$,

$\theta = 0.36$,

$x_2 = -(0.35 / 0.59)\ x_1 + (0.36 / 0.59)$

$= -0.59\ x_1 + 0.6$

i.e., slope is -0.59.

Second Epoch

w1	W2	θ	X1	X2	a	t	$\alpha\delta$	**δw1**	**δw2**	**$\delta\theta$**	E
0.35	0.59	0.36	0	0	-0.36	-1.00	-0.16	-0.00	-0.00	0.16	0.21
0.35	0.59	0.52	0	1	-0.08	-1.00	-0.27	-0.00	-0.27	0.27	0.57
0.35	0.32	0.79	1	0	-0.71	-1.00	-0.14	-0.14	-0.00	0.14	0.16
0.21	0.32	0.93	1	1	-0.68	1.00	0.35	0.35	0.35	-0.35	0.98

Third Epoch

w1	W2	θ	X1	X2	a	t	$\alpha\delta$	**δw1**	**δw2**	**$\delta\theta$**	E
0.56	0.59	0.67	0	0	-0.58	-1.00	-0.11	-0.00	-0.00	0.11	0.09
0.56	0.59	0.67	0	1	-0.01	-1.00	-0.25	-0.00	-0.25	0.25	0.49
0.56	0.32	0.42	1	0	-0.37	-1.00	-0.16	-0.16	-0.00	0.16	0.20
0.40	0.32	0.42	1	1	-0.26	1.00	0.32	0.32	0.32	-0.32	0.80

Forth Epoch

w1	W2	θ	X1	X2	a	t	αδ	**δw1**	**δw2**	**δθ**	E
0.72	0.74	0.77	0	0	-0.77	-1.00	-0.06	-0.00	-0.00	0.06	0.03
0.72	0.74	0.83	0	1	-0.09	-1.00	-0.23	-0.00	-0.23	0.23	0.42
0.72	0.51	1.06	1	0	-0.34	-1.00	-0.16	-0.16	-0.00	0.16	0.22
0.55	0.51	1.22	1	1	-0.16	1.00	0.29	0.29	0.29	-0.29	0.67

Fifth Epoch

w1	W2	θ	X1	X2	a	t	αδ	**δw1**	**δw2**	**δθ**	E
0.84	0.80	0.93	0	0	-0.93	-1.00	-0.02	-0.00	-0.00	0.02	0.00
0.84	0.80	0.95	0	1	-0.15	-1.00	-0.21	-0.00	-0.21	0.21	0.36
0.84	0.59	1.16	1	0	-0.32	-1.00	-0.17	-0.17	-0.00	0.17	0.23
0.67	0.59	1.33	1	1	-0.07	1.00	0.27	0.27	0.27	-0.27	0.57

Sixth Epoch

w1	W2	θ	X1	X2	a	t	αδ	**δw1**	**δw2**	**δθ**	E
0.94	0.86	1.06	0	0	-1.06	-1.00	0.02	0.00	0.00	-0.02	0.00
0.94	0.86	1.05	0	1	-0.19	-1.00	-0.20	-0.00	-0.20	0.20	0.33
0.94	0.65	1.25	1	0	-0.31	-1.00	-0.17	-0.17	-0.00	0.17	0.24
0.77	0.65	1.42	1	1	-0.00	1.00	0.25	0.25	0.25	-0.25	0.50

Seventh Epoch

w1	W2	θ	X1	X2	a	t	αδ	**δw1**	**δw2**	**δθ**	E
1.02	0.90	1.17	0	0	-1.17	-1.00	0.04	0.00	0.00	-0.04	0.01
1.02	0.90	1.13	0	1	-0.22	-1.00	-0.19	-0.00	-0.19	0.19	0.30
1.02	0.71	1.32	1	0	-0.31	-1.00	-0.17	-0.17	-0.00	0.17	0.24
0.84	0.71	1.50	1	1	0.06	1.00	0.24	0.24	0.24	-0.24	0.44

First correct pass 'through the training set'. Halt.

Unlike the Perceptron rule, it is possible to generalize the delta rule to train more than a single layer at once. It turns out to be possible to calculate the slope of the error gradient at intermediate network layers. This was our original goal and is fulfilled in the so-called Backpropagation algorithm or generalized delta rule to be dealt with in the next lecture

Artificial Neural Networks
Lecture Notes

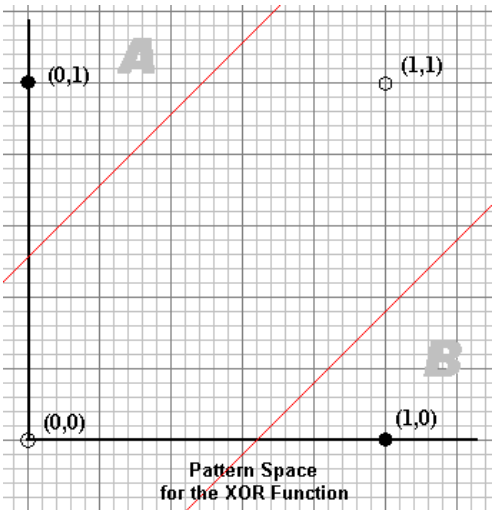
Chapter 5

Contents

Backpropagation

- **Multilayered Networks** Multilayered Networks can compute a wider range of Boolean functions than single-layered networks.

e.g., XOR cannot be computed on a single-layer network (because it is not linearly separable):



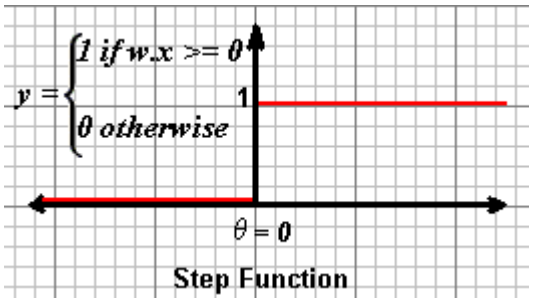
- Pricetag (the trade-off): Learning is slower!

- **The Backpropagation Algorithm** - employs **gradient descent** to find the minimum of the error function in weight space:

  $E(w_1, w_2, ..., w_n)$

- The Error Function must be:
  - Continuous, and
  - Differentiable

- Hence, the Step function is inadequate:



It is neither continuous nor differentiable.

- A popular activation function is the **Sigmoid** - a real-valued function:

S_c: **R** → (0,1) ,

$S_c(x) = 1/ 1 + e^{-cx}$

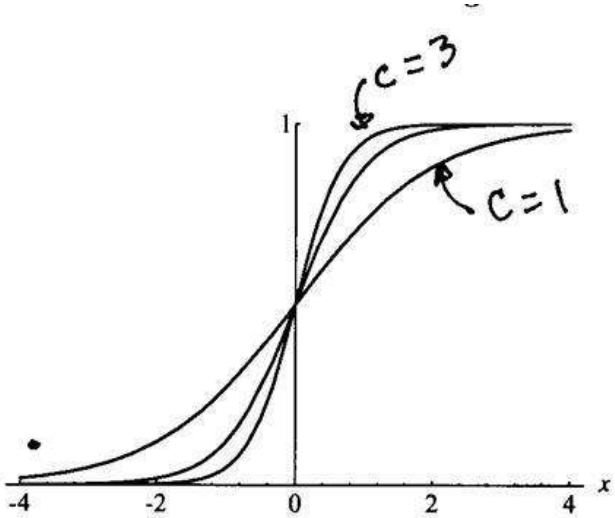


Fig. 7.1. Three sigmoids (for $c = 1$, $c = 2$ and $c = 3$)

- For higher values of c - the sigmoid resembles (or approximates) the step function.

- We let c = 1 (i.e., $S_1(x)$, written s(x))

- **Recalling our calculus**

The Reciprocal Rule

If g is differentiable at x, and $g(x) \neq 0$, then 1/g is differentiable at x, and $(1/g)'(x) = -g'(x) / (g(x))^2$

- Hence, the derivative of the sigmoid with respect to x is

$$d/dx\ s(x) = e^{-x} / (1+ e^{-x})^2$$
$$= s(x)(1 - s(x)) .$$

Alternatives to the Sigmoid *

Classic sigmoid is the upper-left graph in this figure.

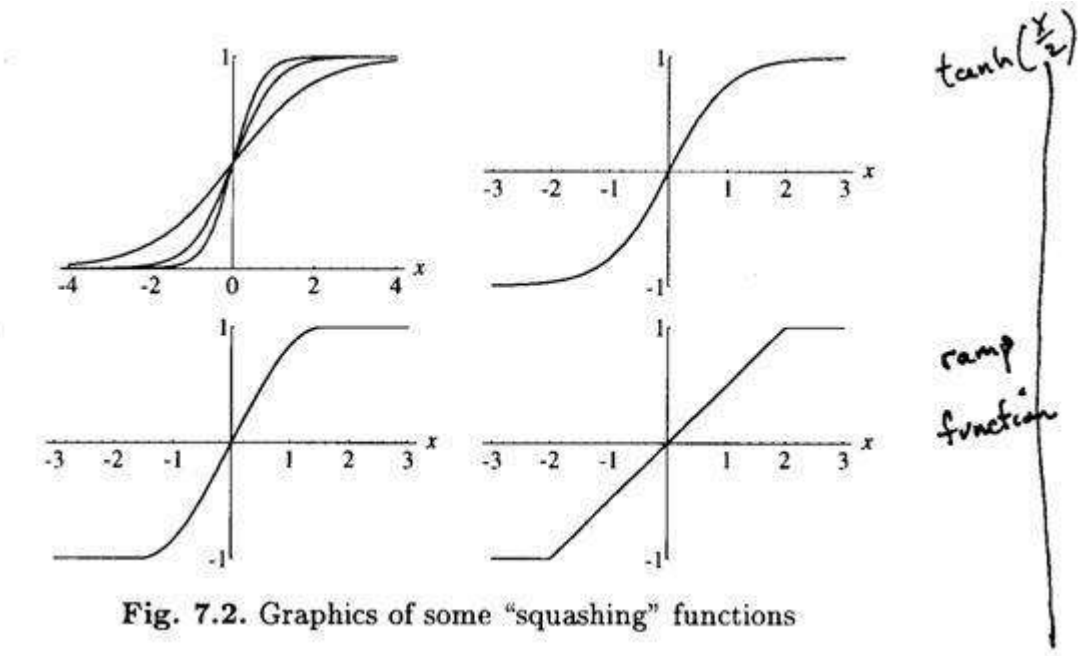


Fig. 7.2. Graphics of some "squashing" functions

- **Symmetrical Sigmoid S(x)**:
  $S(x) = 2s(x) - 1 = (1 - e^{-x}) / (1 + e^{-x})$

  Hyperbolic tangent for x/2, where $\tanh x = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$
  (The upper right graph in figure)

- **Ramp Function**:
  (Lower-right graph in figure)

One must avoid the two points where the derivative is undefined.

- ***Smoothing*** is produced by a sigmoid in a step of the error function.

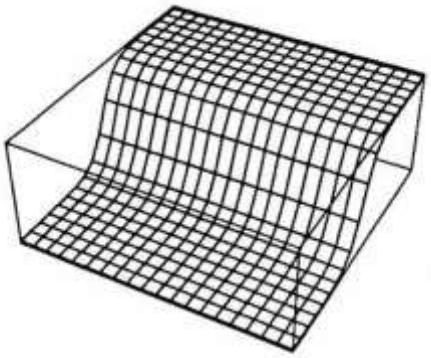


Fig. 7.3. A step of the error function

- We follow the gradient direction (negative gradient direction!) to find the minimum of this function.

- Sigmoid always has a positive derivative; the slope of the error function provides a descent direction which can be followed:

  $e^{-x} / (1+ e^{-x})^2$

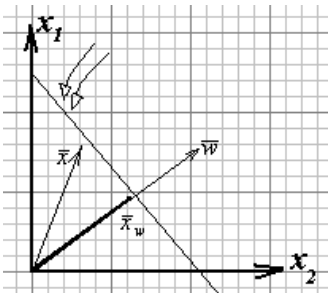
- "We can think of our search algorithm as a physical process in which a small sphere is allowed to roll on the surface of the error function until it reaches the bottom."
  Rojas, p. 151.

- The sigmoid uses the net amount of excitation as its argument.

- Given weights $w_1, ..., w_n$ and a bias $-\theta$, a sigmoidal unit computes for the input $x_1, ..., x_n$, the output

$$\frac{1}{1 + \exp\left(\sum_{i=1}^{n} w_i x_i - \theta\right)}$$

- A higher net amount of excitation brings the unit's output closer to 1.



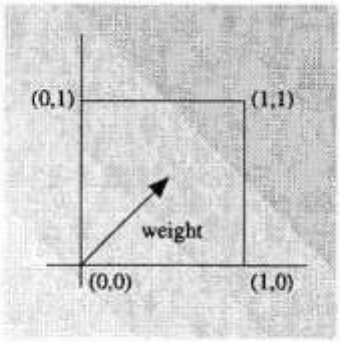


Fig. 7.4. Continuum of classes in input space

- The step of the sigmoid is normal to the vector (w_1, ..., w_n, $-\theta$).
- The weight vector points in the direction in extended input space, in which the output of the sigmoid changes faster.

Drawback for the Sigmoid As Activation Function

- *Local minima* appear in the error function which would not be present if the step function had been used.

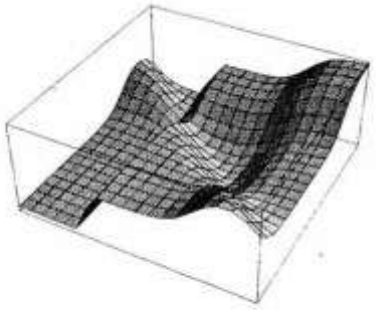


Fig. 7.5. A local minimum of the error function

- There is a local minimum in this figure with a higher error level than in other regions.

- If gradient descent begins in this valley, the algorithm will not converge to the global minimum

The Learning Problem

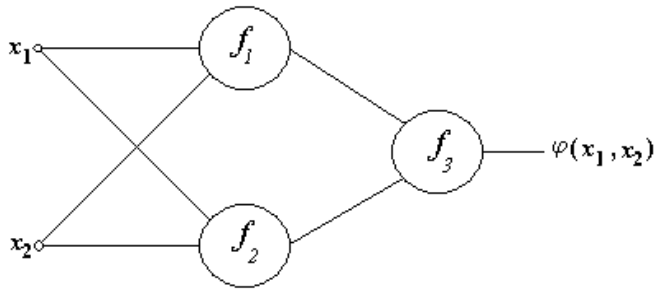
- Each neuron (computing unit) in a neural network evaluates a single primitive function of its input.

- The network represents a chain of function compositions which transform an input (called a *pattern*) to an output vector.

- The network is a particular implementation of a *composite function* from input to output space, which is called the ***network function***.

- The **learning problem** consists of finding the optimal combination of weights so that the network function φ approximates a given function *f* as closely as possible.

HOWEVER, we are not given the function *f* explicitly, but only implicitly through some examples.



- We are given a *feed-forward network* with n inputs and m output units. The number of *hidden units* is arbitrary and any feed-forward connection pattern is permitted.

- We are also given a training set { ($\bar{x}_1$, $\bar{t}_1$), ... , ($\bar{x}_p$, $\bar{t}_p$) }

$$\varphi: \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2^m$$

  Input patterns $\bar{x}_i$ are *n*-dimensional vectors.
  Output patterns $\bar{t}_i$ are *m*-dimensional vectors.

- The *primitive functions* at each node in the network are continuous and differentiable.

- The weights of the edges are Real numbers that are randomly chosen.

- Input pattern ***x;¯***i from the training set is presented to this network. It produces an output ***o;¯***i which may be different from the target ***t;¯***i (the desired output).

- Our goal is to make ***o;¯***i and ***t;¯***i identical for i = 1, ..., p by using a ***learning algorithm***.
  i.e., our goal is to minimize the error function of the network, where

$$E = \frac{1}{2}\sum_{i=1}^{p} \| o_i - t_i \|^2$$

- After minimizing this function for the training set, new unknown input patterns are presented to the network and we expect it to ***interpolate***.

  i.e., the network must recognize whether a new input vector is similar to learned patterns and accordingly produce a similar output.

- The backpropagation algorithm is used to find a local minimum of the error function.

- The gradient of the error function is computed and is used to correct the initial weights.

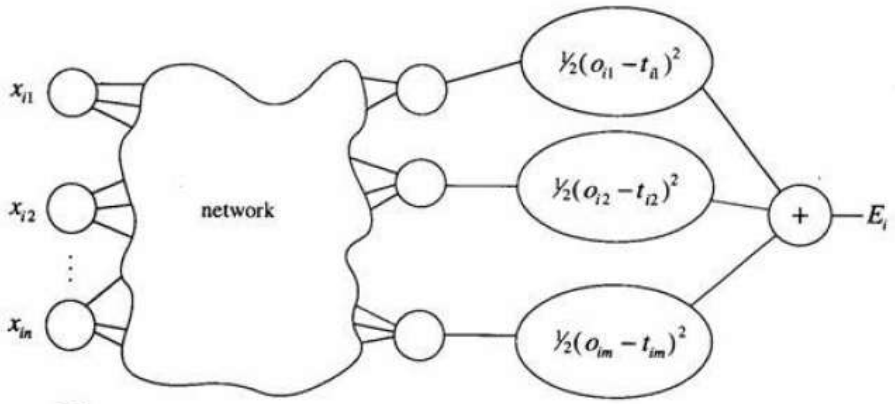


Fig. 7.6. Extended network for the computation of the error function

- The network is extended, so that it computes the error function automatically.

- Every one of the j output units of the network is connected to a node which evaluates the function

$$1/2\ (o_{ij} - t_{ij})^2$$

where o_{ij} and t_{ij} denote the jth component of the output vector ***o;***¯i and the target ***t;***¯i .

- The outputs of these m additional nodes are collected at a node which adds them up and gives the sum E_i as its output.
- The same network extension is built for each pattern t_i or repeated p times !
- A computing unit collects all quadratic errors and outputs their sum E1 + ... + E_p.
- We now have a network capable of calculating the total error for a given training set.
- The weights in the network are the only parameters that can be modified to make the quadratic error E as low as possible.
- We can minimize E by using an iterative process of gradient descent.
- We will need to calculate the gradient

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_L} \right)$$

- Each weight is updated using the increment

  $\Delta w_i = - y (\partial E / \partial \Delta w_i)$ for i = 1, ..., L .

  Where y is the learning rate - i.e., a proportionality parameter which defines the step length of each iteration in the negative gradient direction.
- The learning problem reduces to calculating the gradient of a network function with respect to its weights.

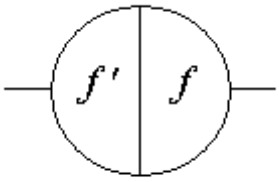
$\nabla E = 0$ at the minimum of the error function.

Derivatives of Network Functions

Calculating the Gradient

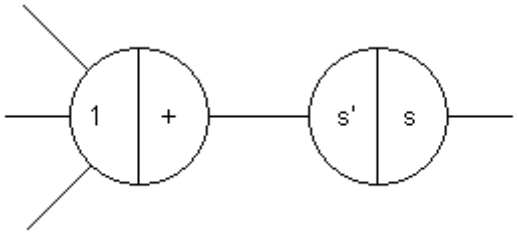
- B-diagram - (Backpropagation diagram):

  Each node in the neural network consists of a left and right side.



Taken from fig 7.7 in Rojas

- The right side computes the primitive function *f* associated with the node.
- The left side computes the derivative *f'* for some input.
- **Note**: The *Intergration* function can be separated from the *Activation* function by splitting each node s.t.
  - The First node computes the sum of its inputs,
  - The second node computes the activation function s.



Taken from fig 7.8 in Rojas

- The derivative of s is s'.
- The partial derivative of the sum of n arguments with respect to any one of them is just 1.

Network Evaluation

Stage 1

Feedforward Step -

Information comes from the left and each unit evaluates its primitive function *f* in its right side, as well as the derivative *f'* in its left side.
Both results are stored in the unit, but only the result from the right side is transmitted to the units connected to the right.

Stage 2
Backpropagation Step -
The whole network is run backwards, and the stored results are now used.

First Case: Function Composition

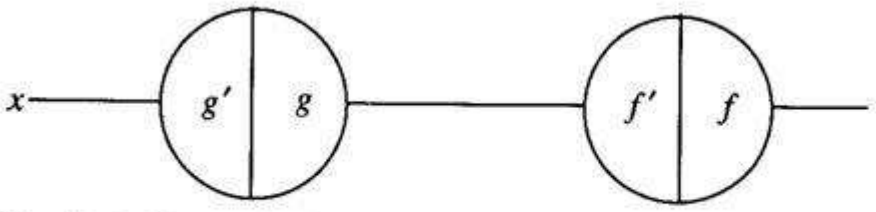


Fig. 7.9. Network for the composition of two functions

- In the feed-forward step, the network computes the composition of the functions *f* and *g*. Incoming information into a unit is used as the argument for the evaluation of the node's primitive function and its derivative.

- After the feedforward step, we have what is shown in the figure here.

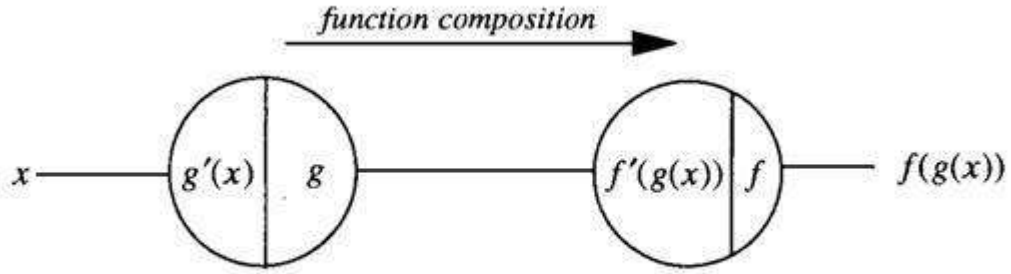


Fig. 7.10. Result of the feed-forward step

- The correct result of the function composition has been produced at the output.
  Each unit has stored some information on its left side.

- In the backpropagation step, the input from the right of the network is the constant 1.

- Incoming information to a node is multiplied by the value stored in its left side.

- The result of the multiplication is transmitted to the next unit to the left. The result at each node is called the *traversing value* at this node:

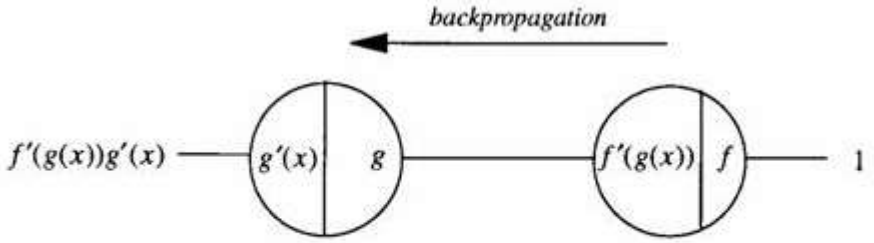


Fig. 7.11. Result of the backpropagation step

- The final result of the backpropagation step is *f'(g(x)) g'(x)* , i.e. , the derivative of the function composition *f(g(x))* implemented by the network.

- The backpropagation step thus provides an implementation of the chain rule.

- One may think of the network as being run *backwards* with the input 1, whereby, at each node, the product with the value stored in the left side is computed.

Second Case: Function Addition

- A network for the computation of the addition of the functions *f1* and *f2*.

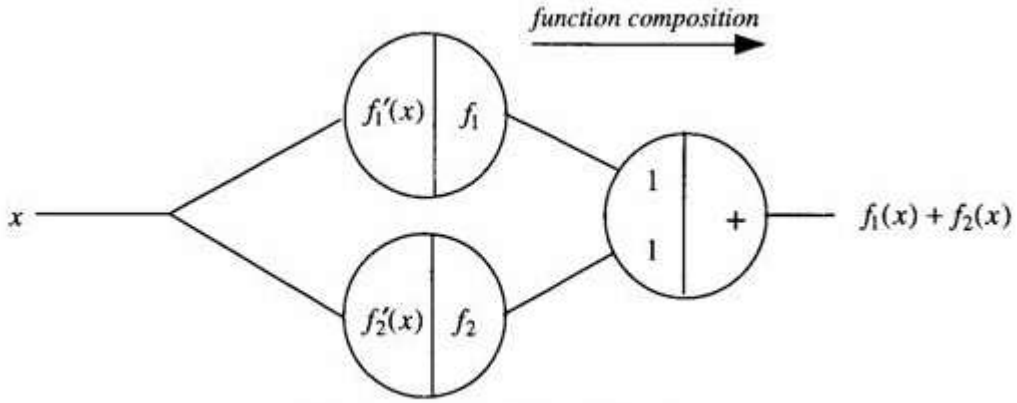


Fig. 7.12. Addition of functions

Note that the extra node handles the addition of the functions.

- In the feed-forward step the network computes *f1(x) + f2(x)* .

- In the backpropagation step the constant 1 is fed from the right side into the network:

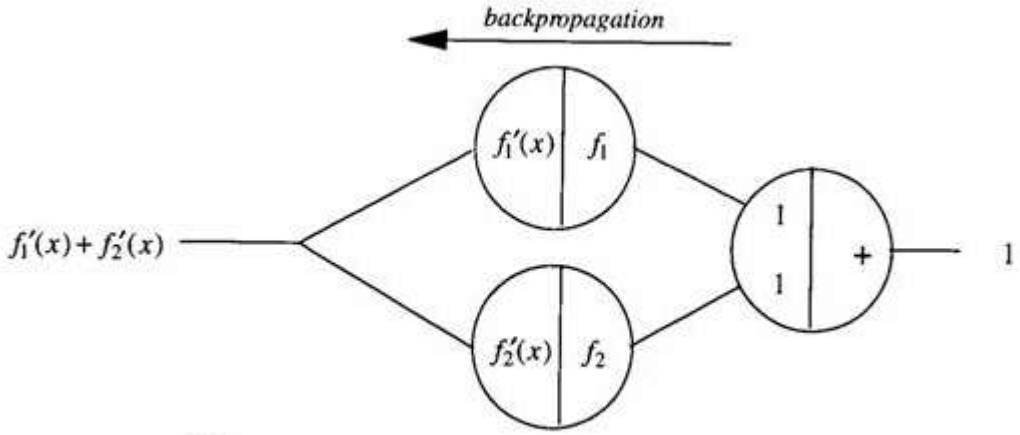


Fig. 7.13. Result of the backpropagation step

- All incoming edges to a unit fan out the *traversing value* at this node and distribute it to the connected units to the left.

- When two right-to-left paths meet, the computed traversing values are added.

Third Case: Weighted Edges

- In the feed-forward step, the incoming information x is multiplied by the edge's weight w.

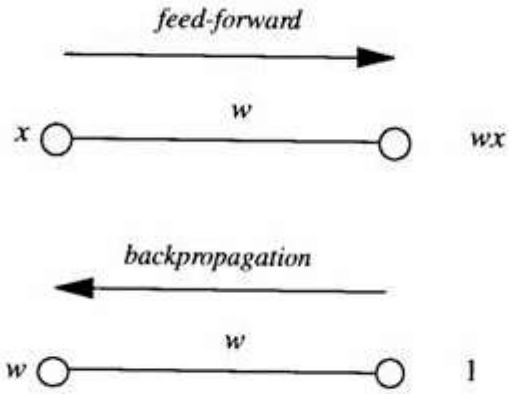


Fig. 7.14. Forward computation and backpropagation at an edge

In the feedforward step, the input is x , and the result is wx.

In the backpropagation step, the input is 1, and the result is d(wx)/dx.

- In the backpropagation step, the traversing value 1 is multiplied by the weight of the edge. The result is w, which is the derivative of wx with respect to x.

Steps of the Backpropagation Algorithm

- We assume we are dealing with a network with a single input and a single output unit.

  This is easily generalized...

- From Rojas:

  Algorithm 7.2.1 Backpropagation algorithm [Computing the derivative of a network function]
  Consider a network with a single real input *x* and network function *F*. The derivative *F'(x)* is computed in two phases:

Feed-forward:	The input *x* is fed into the network. The primitive functions at the nodes and their derivatives are evaluated at each node. The derivatives are stored.
Backpropagation:	The constant 1 is fed into the output unit and the network is run backwards. Incoming information to a node is added and the result is transmitted to the left of the unit. The result collected at the input unit is the derivative of the network function with respect to *x*.

The following proposition shows that the algorithm is correct.

Proposition 7.2.1 *Algorithm 7.2.1 computes the derivative of the network function F with respect to the input x correctly*.

[- End Rojas Quote -]

- The above algorithm generalizes to Activation functions of several variables.(See pp. 160-161)

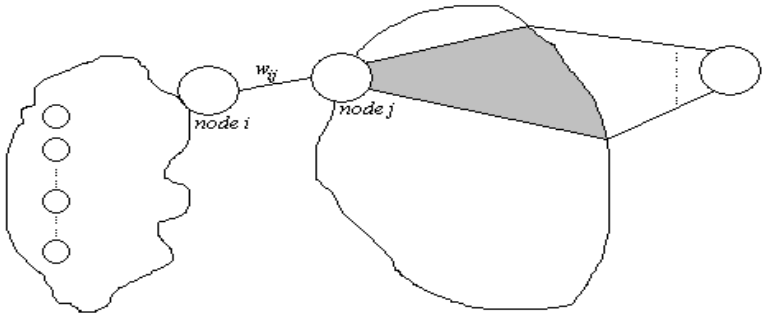
Learning with Backpropagation

- **Goal** - to minimize the error function E
  $E(w_1, w_2, ..., w_{n+1})$

We must deal with all the weights in the network one at a time. We store the output of each unit in its right side.

- We perform the backpropagation step in the extended network that computes the error function.

- We fix our attention on one of the weights, say w_{ij} .



- This weight can be treated as an input channel into the subnetwork made of all paths starting at w_{ij} and ending at the single output unit of the network (shaded region above.)

- The information fed into the subnetwork in the feed-forward step was $o_i w_{ij}$, where o_i is the stored output of unit i.
  $x \rightarrow^{w} wx$

- The backpropagation step computes the gradient of E with respect to this input:
  i.e., $\partial E / \partial o_i w_{ij}$

- In the backpropagation step, o_i is treated as a constant. Hence we have,

$$\partial E / \partial w_{ij} = o_i \, (\partial E / \partial o_i w_{ij})$$

- All subnetworks defined by each weight of the network can be handled simultaneously.

- The following additional information must be stored at each node:

  - The output o_i of the node in the feed-forward step.

  - The cumulative result of the backward computation in the backpropagation step up to this node. This quantity is called the *backpropagated error*.

- Let σ_j = the backpropagated error at the jth node. Then,
  $\partial E / \partial w_{ij} = o_i \, \sigma_j$

Once all partial derivatives have been computed, we can perform gradient descent by adding to each weight wij the increment

$$\Delta w_{ij} = -y \, o_i \, \sigma_j$$

where y is the learning rate, and $o_i \sigma_j$ is the gradient. This correction step transforms the backpropagation algorithm into a learning algorithm for neural networks.

Layered Networks

- An important special case of feed-forward networks is that of *layered networks* with one or more hidden layers

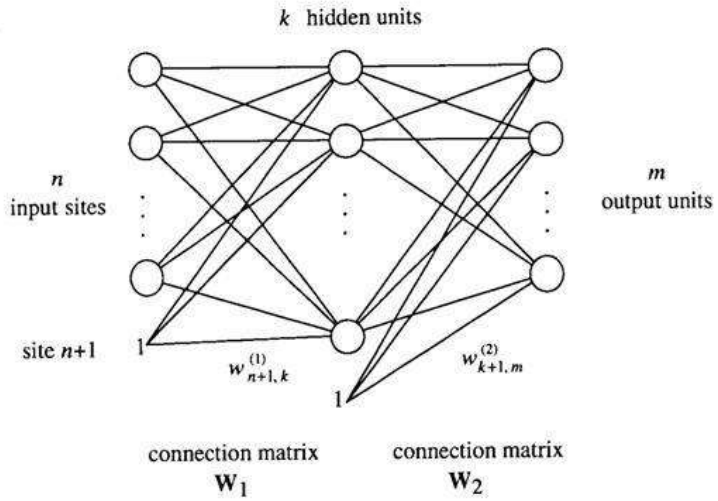


Fig. 7.17. Notation for the three-layered network

n input sites

k hidden units

m output units

$w^{(1)}ij$ the weight between input i and hidden unit j

$w^{(2)}ij$ the weight between hidden unit i and output j

$-\theta$ the bias of each unit implemented as the weight of an additional edge

$w^{(1)}n+1,j$ the weight between the constant 1 and the hidden unit j

$w^{(2)}k+1,j$ the weight between the constant 1 and the output unit j

Artificial Neural Networks
Lecture Notes

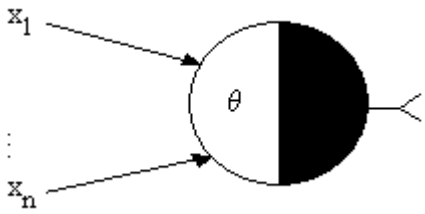
Chapter 6

Contents

Neural Net Paradigms - Review and Clarification

McCulloch-Pitts Networks

- Binary signals - both input and output
- Directed, unweighted edges of excitatory or inhibitory (marked with a small circle) inputs.

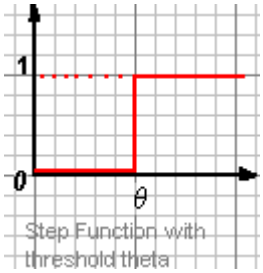


Threshold value θ

inputs $x_1, x_2, \ldots, x_n$ through n excitatory edges and $y_1, y_2, \ldots, y_m$ through m inhibitory edges.

- if m >= 1 and at least one of the signals $y_1, y_2, \ldots, y_m$ is the 1, the unit is inhibited and the output is 0.

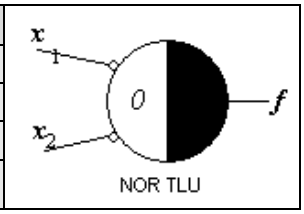
- Otherwise, the total excitation $x;^{-} = x_1 + x_2 + \ldots + x_n$ is computed and compared with the threshold θ.



If $x;^{-} >= \theta$, then unit fires a 1
If $x;^{-} < \theta$, then output = 0

- Example: The NOR Function

X1	X2	$\sum x_i$	NOR
0	0	0	1
0	1	-	0
1	0	-	0
1	1	-	0



Perceptron Learning Rule

- Weight Updates:
  $\boldsymbol{w}_{;}^{\neg\prime} = \boldsymbol{w}_{;}^{\neg} + \alpha(t - y)\, \boldsymbol{v}_{;}^{\neg}$, or
  $\Delta \boldsymbol{w}_{;}^{\neg} = \alpha(t - y)\, \boldsymbol{v}_{;}^{\neg}$

  Where α is the learning rate, t is the target, y is the output, and $\boldsymbol{v}_{;}^{\neg}$ is the input vector (equivalent to $\boldsymbol{x}_{;}^{\neg}$.)

  $\Delta wi = \alpha(t - y)vi$

  with i = 1 to n+1 where $w_{n+1} = \theta$ and $v_{n+1} = -1$, always.

- Example:
  Compute the two-input NOR function using the Perceptron Learning Rule

  Let initial weights be w1 = 0.1, w2 = 0.0, θ= 0.2, and learning rate α= 0.5.

X1	X2	NOR
0	0	1
0	1	0
1	0	0
1	1	0

x1	x2	w1	w2	θ	a	y	t	α(t-y)	Δw1	Δw2	Δθ
0	0	0.1	0.0	0.2	0	0	1	0.5(1-0)=0.5	0.0	0	-0.5
0	1	0.1	0.0	-0.3	0	1	0	0.5(0-1)=-0.5	0.0	-0.5	0.5
1	0	0.1	-0.5	0.2	0.1	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	1	0.1	-0.5	0.2	-0.4	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
0	0	0.1	-0.5	0.2	0	0	1	0.5(1-0)=0.5	0.0	0.0	-0.5
0	1	0.1	-0.5	-0.3	-0.5	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	0	0.1	-0.5	-0.3	0.1	1	0	0.5(0-1)=-0.5	-0.5	0.0	0.5
1	1	-0.4	-0.5	0.2	-0.9	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
0	0	-0.4	-0.5	0.2	0.0	0	1	0.5(1-0)=0.5	0.0	0.0	-0.5
0	1	-0.4	-0.5	-0.3	-0.5	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	0	-0.4	-0.5	-0.3	-0.4	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
1	1	-0.4	-0.5	-0.3	-0.9	0	0	0.5(0-0)=0.0	0.0	0.0	0.0
0	0	-0.4	-0.5	-0.3	0.0	1	1	0.0	0.0	0.0	0.0
0	1	-0.4	-0.5	-0.3	-0.5	0	0	0.0	0.0	0.0	0.0
1	0	-0.4	-0.5	-0.3	-0.4	0	0	0.0	0.0	0.0	0.0

1	1	-0.4	-0.5	-0.3	-0.9	0	0	0.0	0.0	0.0	0.0

No changes in the last epoch; therefore we halt.

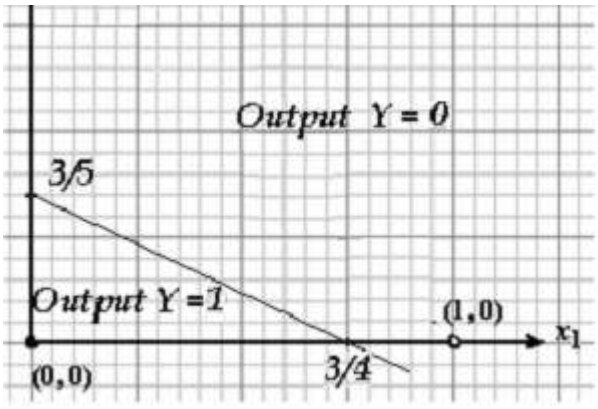
- Final weights:
  $w_1 = -0.4$ $w_2 = -0.5$ $\theta = -0.3$

- Hence

x_2	x_1
3/5	0
0	3/4

slope $m = -w_1/w_2 = -(-0.4 / -0.5) = -4/5$
y-intercept $b = \theta/w_2 = -0.3 / -0.5 = 3/5$
$y = mx + b$ (i.e., $x_2 = mx_1 + b$)

$y = -4/5\ x + 3/5$
$x_2 = -4/5\ x1 + 3/5$



Delta Rule

- Error Function:
  $E = E (w_1, w_2, \ldots , w_{n+1})$

  (where $\bar{w}$ is the augmented weight vector.)

- The optimal weight vector is found by minimizing this function by gradient descent
  $\Delta w_i = -\alpha (\partial E / \partial w_i)$

- One attempt at a suitable error function:

  $e^p = 1/2 (t^p - y^p)^2$

a^p is smooth and continuous. However, output depends on a^p via the *discontinuous* step function.

- One remedy:
  $e^p = 1/2\ (t^p - a^p)^2$

  When using the augmented weighted vector, the output changes as the activation changes sign.
  i.e.,

  $a >= 0$ ➔ $y = 1$

  One choice that works for target values {-1, 1} - *bipolar outputs*.

- If the whole training set is presented, we can obtain the true gradient

  $\partial E / \partial wi$

  (batch training).

- This is computationally intensive.

- Therefore, we adapt the weights based on the presentation of each pattern individually.

- i.e., we present the net with a pattern p, evaluate $\partial e^p / \partial wi$, and we take this as an <u>estimate</u> of the true gradient $\partial E / \partial wi$

  $$e^p = 1/2\ (t^p - a^p)^2$$

  where

  $$a^p = w_1 x^p{}_1 + w_2 x^p{}_2 + \ldots + w_{n+1} x^p{}_{n+1}$$

  $$\partial e^p / \partial w_i = -(t^p - a^p) x_i^p$$

- Hence,

  $$\Delta wi = -\alpha(\partial E / \partial wi)$$
  $$\approx -\alpha(\partial e^p / \partial wi)$$
  $$\Delta w_i = \alpha(t^p - a^p) x_i^p$$

  <u>Careful with the signs !</u>

- <u>Pattern training regime:</u>
  Weight changes are made after each vector presentation.

- **Partial Derivatives**

- Function of two variables.
- Let f be a function of *x* and *y*;

$$f(x, y) = 3x^2 y - 5x \cos \pi y$$

- The partial derivative of f with respect to x is the function $\partial f / \partial x$ obtained by differentiating f with respect to *x*, treating *y* as constant.

$$\partial f(x, y) / \partial x = 6xy - 5\cos \pi y$$

Alternate notation, $f_x (x, y)$.

- The partial derivative of f with respect to *y* is the function $\partial f / \partial y$ obtained by differentiating f with respect to *y*, treating *x* as constant.

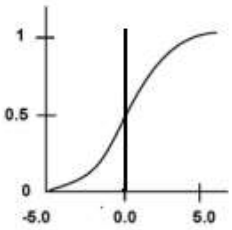
$$f_y (x, y) \text{ or } \partial f (x, y) / \partial y = 3x^2 + 5\pi x \sin \pi y$$

- **Geometric Interpretation**
  $\partial f (x, y) / \partial x$ is the slope of the surface $f (x, y) = 3x^2 y - 5x \cos \pi y$ at the point *P* (*xy*, f(*x, y*)) in the x-direction.
- **An Example Using the NOR Function**
  Train a two-input TLU with initial weights w1 = 0.1, w2 = 0 , θ= 0.2, and learning rate α= 0.25

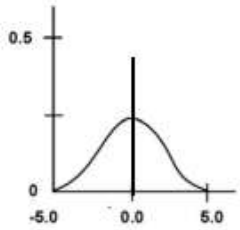
  In the table below note that x3 is always -1 and is input to θ and is used in the activation column a.

  Note also that $(t^p - a^p)$ is referred to as the delta, or δ, so the term $\alpha(t^p - a^p)$ is denoted in the table as $\alpha\delta$.

x1	x2	x3	w1	w2	θ	a	t	$\alpha\delta$	δw1	δw2	$\delta\theta$
0	0	-1	0.1	0.0	0.2	-0.2	1	1.2/4 = 0.3	0.0	0	-0.3
0	1	-1	0.1	0.0	-0.1	0.1	-1	-1.1/4 =-0.275	0.0	-0.275	0.275
1	0	-1	0.1	-0.275	+0.175	-0.075	-1	-0.925/4 =-0.231	-0.231	0.0	0.231
1	1	-1	-0.131	-0.275	0.406	-0.812	-1	-0.047	-0.047	-0.047	0.047
0	0	-1	-0.178	-0.322	0.453	-0.453	1	1.453/4 = 0.363	0	0	



Sigmoid S(a)



Slope of Sigmoid S'(a)

- Suppose first that the activation is very large (or small) so that the output is close to 1 or 0, respectively.

  Graph is flat and hence gradient s'(a) is very small.

  Activation close to the threshold implies that s'(a) is large.

- Alternately - we may use y^p instead of a^p.
  Then,

  $$\Delta wi = \alpha s'(a)(t^p - a^p)\, x_i^p.$$

- It is not surprising that s'(a) appears here - Any changes in the weights alter the output (and hence the error) via the activation.

  The effect of any such changes depends thus on the sensitivity of the output with respect to the activation.

Backpropagation

- Layered Network
  n input sites
  k hidden units
  m output units

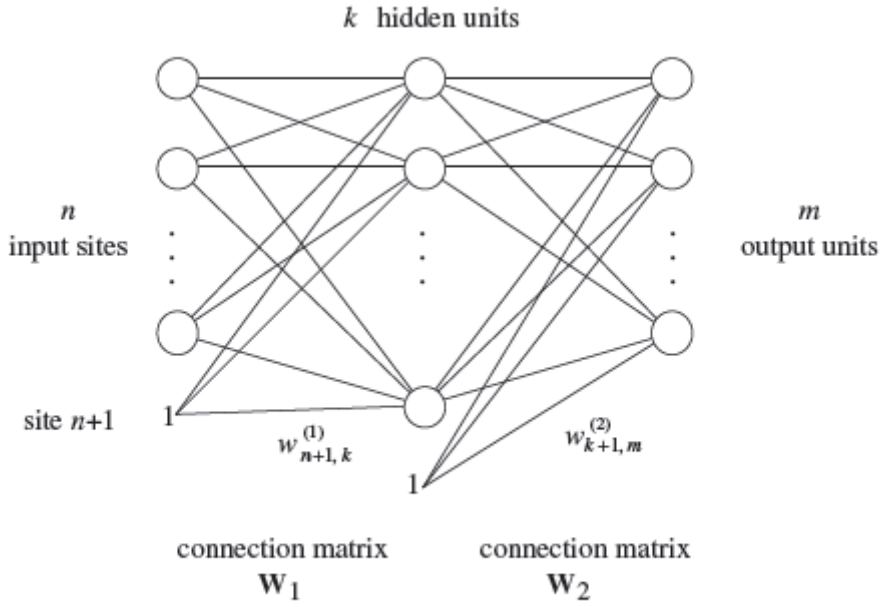


Fig. 7.17. Notation for the three-layered network

- The weight between input site i and hidden unit j: wij $^{(1)}$
- The weight between hidden unit i and output unit j: wij $^{(2)}$
- The bias, -θ is implemented as the weight of an additional edge.
- There are (n+1) * k weights between input sites and hidden units and (k+1) * m between hidden and output units. Thus,

  w;$^{\frown}$1 is the (n+1) * k matrix with component wij $^{(1)}$ at the ith row and jth column.

  w;$^{\frown}$2 the (k+1) * m matrix with components wij $^{(2)}$.

- The n-dimensional input vector ***o;***$^{\frown}$ = (o1, ... , on) is extended to: o;$^{\wedge}$ = (o1, ... , on, 1) .
- The excitation net$_j$ of the jth hidden unit is given by

$$net_j = \sum_{i=1}^{n+1} w_{ij}^{(1)} \hat{o}_i.$$

- The activation function is a sigmoid and the output oj $^{(1)}$ is

$$o_j^{(1)} = s\left(\sum_{i=1}^{n+1} w_{ij}^{(1)} \hat{o}_i\right) .$$

- Excitation of all units in the hidden layer = o;^*w;*¯1

  o;¯ (1) the vector whose components are the outputs of hidden units

  o;¯ (1) = s(o;^*w;*¯1) .

- Excitation of units in the output layer is computed using

  o;^(1) = ($o_1^{(1)}$, ... , $o_k^{(1)}$, 1) .

- The output of the network is the m-dimensional vector

  o;¯ (2) = s(o;^(1)*w;*¯2) .

Steps of the Algorithm

- The following schematic is for a single input-output pair (*o;*¯ , *t;*¯)

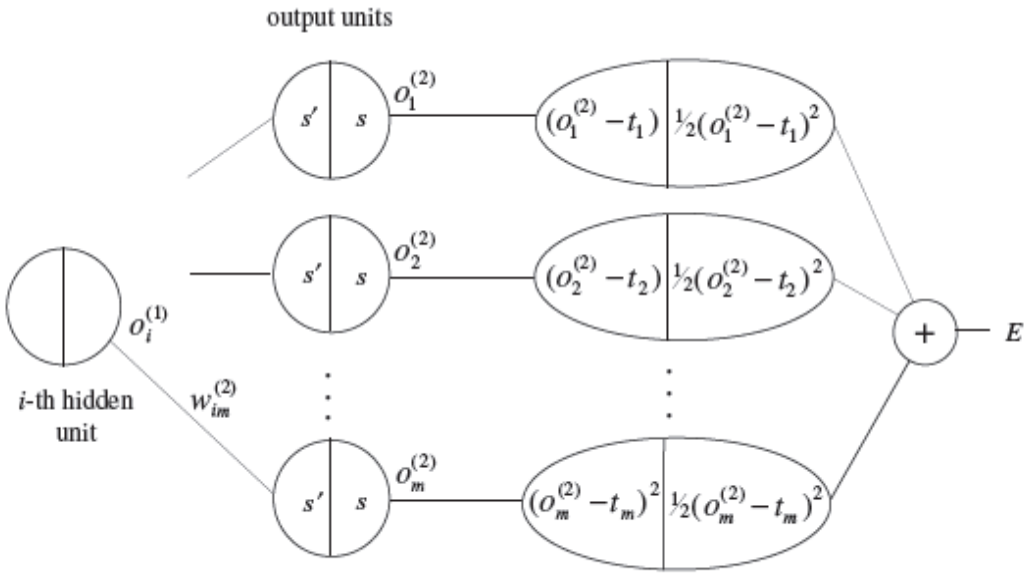


Fig. 7.18. Extended multilayer network for the computation of E

- The error function for p input-output examples - we would need p copies of the above network.

- The weights for the network are chosen randomly.

The Backpropagation Algorithm

i) Feed-forward Computation

ii) Backpropagation to the output layer

iii) Backpropagation to the hidden layer

iv) Weight Updates.

- The algorithm is stopped when the value of the error function has become sufficiently small.

- First Step: *Feedforward Computation*

  The vector $\boldsymbol{o}_{;}^{-}$ is presented to the network.
  The vectors $\boldsymbol{o}_{;}^{-(1)}$ and $\boldsymbol{o}_{;}^{-(2)}$ are computed and stored.
  The evaluated derivatives of the activation functions are also stored at each unit.

- Second Step: Backpropagation to the Output Layer
  We are looking for the first set of partial derivatives $\partial E / \partial w_{ij}^{(2)}$

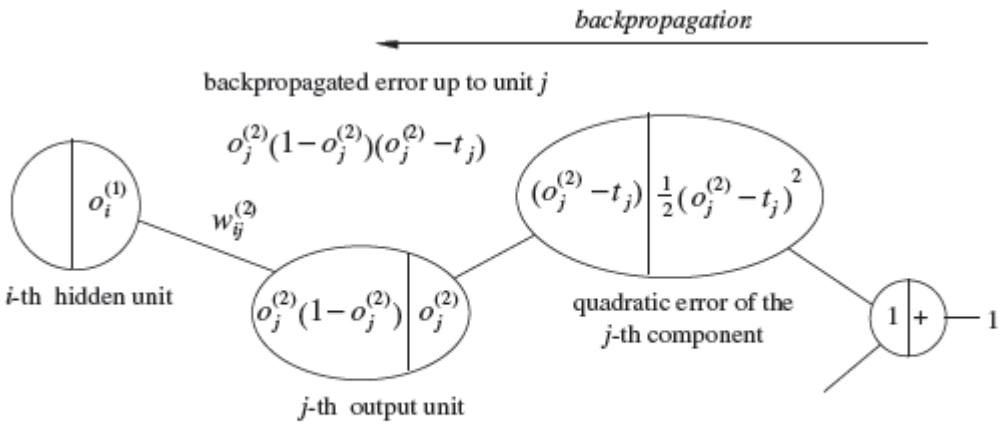


Fig. 7.19. Backpropagation path up to output unit j

From this path, we can collect all the multiplicative terms which define the backpropagated error $\delta j^{(2)}$

Recall that $d/dx\ s(x) = e^{-x} / (1+e^{-x})^2 = s(x)\ (1 - s(x))$. Thus, ...

$$\delta_j^{(2)} = o_j^{(2)} (1 - o_j^{(2)}) (o_j^{(2)} - t_j)$$

and the partial derivative we are looking for is

$$\partial E / \partial w_{ij}^{(2)} = [\ o_j^{(2)} (1 - o_j^{(2)}) (o_j^{(2)} - t_j)\ ]\ o_i^{(1)}$$
$$= \delta_j^{(2)}\ o_i^{(1)}$$

- N.B. For this last step we consider $w_{ij}^{(1)}$ to be a variable and its input $o_i^{(1)}$ a constant.

- The general situation during the backpropagation algorithm:

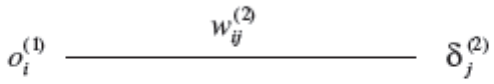


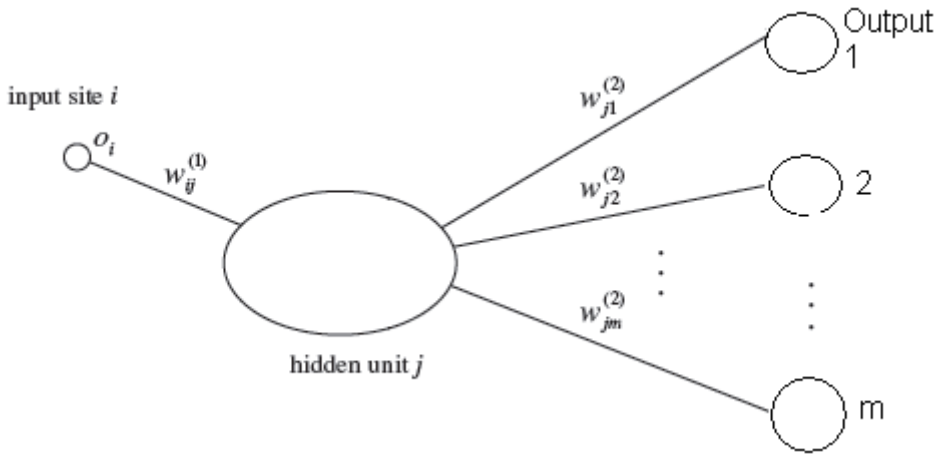
Fig. 7.20. Input and backpropagated error at an edge

At the input side of the edge with weight w_{ij} we have $o_i^{(1)}$ and at the output side , the backpropagated error $\delta_j^{(2)}$

- Third Step: *Backpropagation to the Hidden Layer*

We want to compute $\partial E / \partial w_{ij}^{(1)}$

Each unit j in the hidden layer is connected to each unit q in the output layer with an edge of weight $w_{jq}^{(2)}$ for q = 1, ... , m.



The backpropagated error up to unit j in the hidden layer must be computed taking into account all possible backward paths as shown:

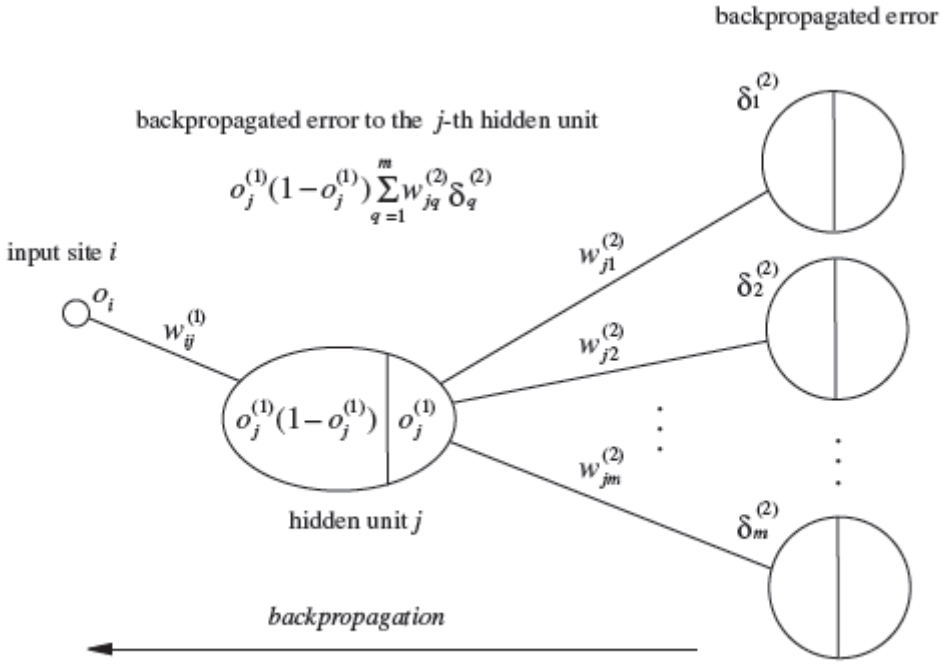


Fig. 7.21. All paths up to input site i

The backpropagated error is

$$\delta j^{(1)} = oj^{(1)}(1 - oj^{(1)}) \sum^{m}_{q=1} wjq^{(2)}\ \delta q^{(2)}$$

The partial derivative we are looking for:

$$\partial E / \partial wij^{(1)} = \delta j^{(2)} o_i.$$

- Fourth Step: *Weight Updates*
  - After computing all partial derivatives, the network weights are updated in the negative gradient direction.
  - A learning constant y defines the step length of the correction.
  - The corrections for the weights are given by:

    $\Delta wij^{(2)} = -yoi^{(1)}\ \delta^{(2)}$, for i = 1, ... , k+1 and j = 1, ... , m

    and

    $\Delta wij^{(1)} = -yoi\ \delta j^{(2)}$, for i = 1, ... , n+1 and j = 1, ... , k

    with the convention that:
    $o_{n+1} = o_{k+1}^{(1)} = 1$.

- Important: Corrections are made to the weights only after the backpropagated error has been computed for all units in the network.

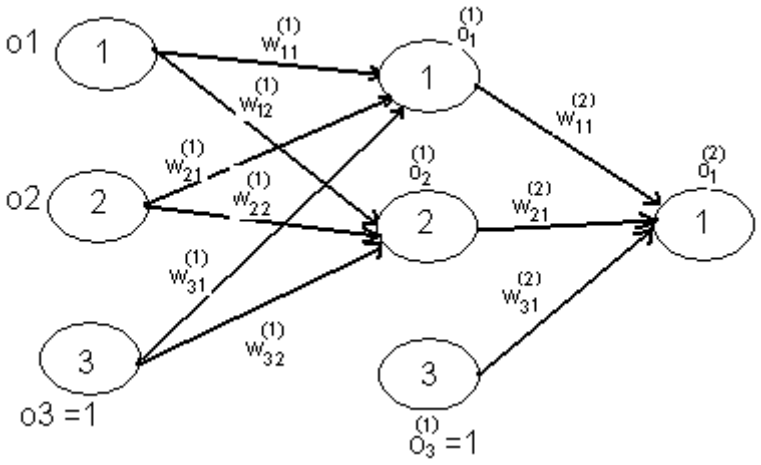
More Than One Training Pattern

- In the case p > 1 input-output patterns, an extended network is used to compute the error function for each of them separately.
- The weight corrections are computed for each pattern yielding for $w_{ij}^{(1)}$ the corrections:

  $\Delta_1 w_{ij}^{(1)}$, $\Delta_2 w_{ij}^{(1)}$, ... , $\Delta_p w_{ij}^{(1)}$
- The necessary updates in the gradient direction are given by:

  $\Delta w_{ij}^{(1)} = \Delta_1 w_{ij}^{(1)} + \Delta_2 w_{ij}^{(1)} + \ldots + \Delta_p w_{ij}^{(1)}$
- (Batch or offline updates are rather expensive as a training set may consist of thousands of patterns.)
- Often, however, the weight updates are made sequentially after each pattern presentation
- (online training) more efficient
- Here, the corrections do not exactly follow the negative gradient direction. This adds some *noise* to the gradient direction - can help to prevent falling into shallow local minima of the error function.
- If the training patterns are selected randomly, the search direction oscillates around the exact gradient direction, and, on average, the algorithm implements a form of descent in the error function.

Three-layer network for solving the X-OR operation



The initial weights and threshold levels are set randomly as follows:

w13 = 0.5	w14 = 0.9	w23 = 0.4	w24 = 1.0	w35 = –1.2	w45 = 1.1	θ3 = 0.8	θ4 = –0.1	θ5 = 0.3
$W_{11}^{(1)}$	$W_{12}^{(1)}$	$W_{21}^{(1)}$	$W_{22}^{(1)}$	$W_{11}^{(2)}$	$W_{21}^{(2)}$	$W_{31}^{(1)}$	$W_{32}^{(1)}$	$W_{31}^{(2)}$

We consider a training set where inputs o_1 and o_2 are equal to 1 and desired target output is 0.

Feed-forward Computation

Calculate the output at the hidden layer $\boldsymbol{o}^{(1)}$:

$$\boldsymbol{o}_1^{(1)} = \text{sigmoid}\,(O_1 W_{11}^{(1)} + O_2 W_{21}^{(1)} - W_{31}^{(1)}) = \frac{1}{1 + e^{-(1 \times 0.5 + 1 \times 0.4 - 0.8)}} = 0.5250$$

$$\boldsymbol{o}_2^{(1)} = \text{sigmoid}\,(O_1 W_{12}^{(1)} + O_2 W_{22}^{(1)} - W_{32}^{(1)}) = \frac{1}{1 + e^{-(1 \times 0.9 + 1 \times 1 - 0.1)}} = 0.8808$$

Next, we calculate the output of the neural network itself:

$$\boldsymbol{o}_1^{(2)} = \text{sigmoid}\,(O_1^{(1)} W_{11}^{(2)} + O_2^{(1)} W_{21}^{(2)} - W_{31}^{(2)}) = \frac{1}{1 + e^{-(0.5250 \times -1.2 + 0.8808 \times 1.1 - 0.3)}} = 0.5097$$

Thus, the following error is obtained:

$e = o_1^{(2)} - t1 = 0.5097 - 0 = 0.5097$

The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, *e*, from the output layer backward to the input layer.

First, we calculate the error gradient in the output layer:

$\delta_1^{(2)} = o_1^{(2)}(1 - o_1^{(2)})\, e$

$= 0.5097\,(1 - 0.5097) \times -0.5097 = 0.5097 \times (1 - 0.5097) \times 0.5097 = 0.1275$

Then we determine the weight corrections at the output unit assuming that the learning rate parameter, y, is equal to **0.1**:

$\Delta wij^{(2)} = -yoi^{(1)}\, \delta^{(2)}$, for i = 1, ... , 3 and j = 1 // only one output

$\Delta w_{11}^{(2)} = -0.1 \times 0.5250 \times 0.1275 = -0.0067$

$\Delta w_{21}^{(2)} = -0.1 \times 0.8581 \times 0.1275 = -0.0112$

$\Delta w_{31}^{(2)} = -0.1 \times 1 \times (-1) \times 0.1275 = 0.0127$

Next, we calculate the backpropagated error at the hidden layer

$\delta j^{(1)} = oj^{(1)}(1 - oj^{(1)}) \sum_{q=1}^{m} wjq^{(2)}\, \delta q^{(2)}$

$\delta_1^{(1)} = o_1^{(1)}(1 - o_1^{(1)})\, w_{11}^{(2)}\, \delta_1^{(2)}$

$= 0.5250 \times (1 - 0.5250) \times (-1.2) \times (0.1275) = -0.0381$

$\delta_2^{(1)} = o_2^{(1)}(1 - o_2^{(1)})\, w_{21}^{(2)}\, \delta_1^{(2)}$

$= 0.8808 \times (1 - 0.8808) \times (1.1) \times (0.1275) = 0.0147$

We then determine the weight corrections:

$\Delta w_{11}^{(1)} = -y\, o_1\, \delta_1^{(1)} = (-0.1) \times 1 \times (-0.0381) = 0.0038$

$\Delta w_{12}^{(1)} = -y\, o_1\, \delta_2^{(1)} = (-0.1) \times 1 \times (0.0147) = -0.0015$

$\Delta w_{21}^{(1)} = -y\, o_2\, \delta_1^{(1)} = (-0.1) \times 1 \times (-0.0381) = 0.0038$

$\Delta w_{22}^{(1)} = -y\ o_2\ \delta_2^{(1)} = (-0.1) \times 1 \times (0.0147) = -0.0015$

$(\theta 3)\quad \Delta w_{31} = -y\ (1)\ \delta_1^{(1)} = (-0.1) \times -1 \times (-0.0381) = -0.0038$

$(\theta 4)\quad \Delta w_{32}^{(1)} = -y\ (1)\ \delta_2^{(1)} = (-0.1) \times (-1) \times (0.0147) = 0.0015$

At last, we update all weights and threshold:

$w_{11}^{(1)} = w_{11}^{(1)} + \Delta w_{11}^{(1)} = 0.5 + 0.0038 = 0.5038$

$w_{12}^{(1)} = w_{12}^{(1)} + \Delta w_{12}^{(1)} = 0.9 + -0015 = 0.8985$

$w_{21}^{(1)} = w_{21}^{(1)} + \Delta w_{21}^{(1)} = 0.4 + 0.0038 = 0.4038$

$w_{22}^{(1)} = w_{22}^{(1)} + \Delta w_{22}^{(1)} = 1.0 + -0015 = 0.9985$

$W_{11}^{(2)} = W_{11}^{(2)} + \Delta W_{11}^{(2)} = -1.2 + -0.067 = -1.25067$

$W_{21}^{(2)} = W_{21}^{(2)} + \Delta W_{21}^{(2)} = 1.1 + -0.0112 = 1.0888$

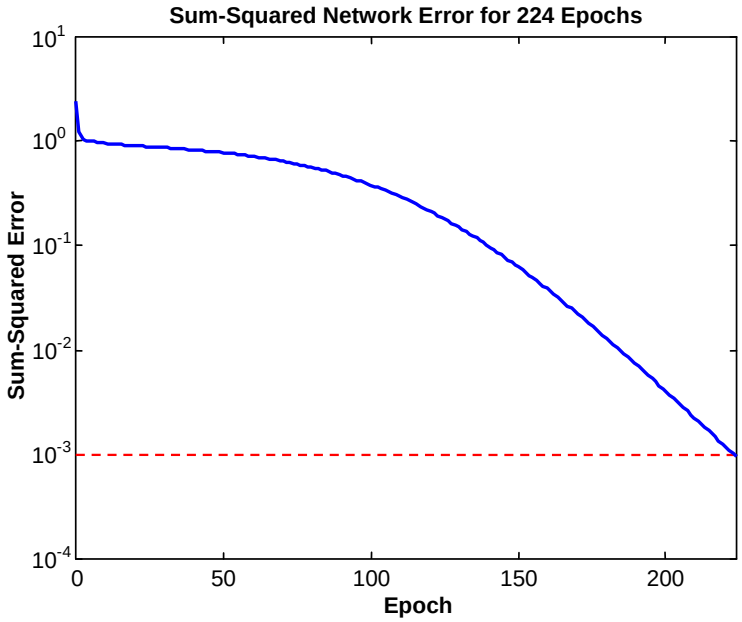
$(\theta 3)\quad w_{31}^{(1)} = W_{31}^{(1)} + \Delta W_{31}^{(1)} = 0.8 + -0.0038 = 0.7962$

$(\theta 4)\quad w_{32}^{(1)} = W_{32}^{(1)} + \Delta W_{32}^{(1)} = -0.1 + 0015 = -0.10985$

$(\theta 5)\quad W_{31}^{(2)} = W_{31}^{(2)} + \Delta W_{31}^{(2)} = 0.3 + 0.0127 = 0.3127$

The training process is repeated until the sum of squared errors is less than 0.001.

Learning curve for operation *Exclusive-OR*



Final results of three-layer network learning

Input		Target output	Actual Output	Error	Sum of Squared errors
O1	O2	(t)	$o_1^{(2)}$	e	
1	1	0	0.0155	-0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	09849	0.0151	
0	0	0	0.0175	-0.0175	

Artificial Neural Networks
Lecture Notes

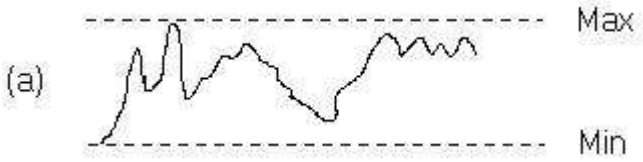
Chapter 7

Application Design

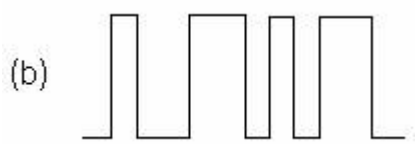
- Neural networks (ANN) perform some useful information-processing function that can be exploited to solve other problems.
    - Knowing how ANN work is only half the story…topology, activation functions, learning rules.
- To build useful ANN applications one needs to understand:
    - The process of acquiring and modeling application data.
    - Selecting the most appropriate network model.
    - Training the network for the application.

Data Representation

- ANN process information in the form of input pattern vectors.
- BPN (and other models) produces an output pattern that is completely independent of the input pattern different processing elements (PE) produce the output than those that accept the inputs.
- Signals propagating through a neural networks are:
    - Analog – continuously variable
    - Discrete – quantized



Analog: Vary continuously from max to min values.



Discrete: This signal may assume only one of two values.

This diagram depicts the two different types of information-bearing signals that can be used by a neural model. Notice that the signals do not need to be periodic, nor does the time base need to be consistent between different signals. (a) Analog signals vary in amplitude smoothly, allowing the signal to

take on any value between its minimum and maximum amplitude. (b) A discrete signal behaves as a step function. A signal of this type can take on only one of two values: its minimum and maximum amplitudes. Values between the minimum and maximum are never valid.

- ANN's "designed to detect, and respond to, the presence of features in an input pattern vector that is presented as a static pattern to the network."

- "How can we capture a myriad of time-varying signals and represent them so that a neural network model can process them as spatial pattern vectors?"

- How does one capture the application data and properly format it prior to presentation to the networks.

Internal Representation Issues

- Signals are limited in amplitude and are generally continuous from 0 to 1 (sometimes from -1 to 1).

- If the amplitude of signals were not limited then those pattern components with very large amplitude swings would dominate the behavior of the network.

External Interpretation Issues

- How does one interpret the outputs from each layer of a network?

- Individually, the processing elements within a network have little or no understanding of the application that the network is being asked to learn.

- Each processing element knows how to compute an input simulation and produce a corresponding output signal…

    - The patterns that are propagated through a neural network consist of a set of component signals that individually reveal very little about the entire pattern.

- Even when viewed collectively, the patterns often do not take on a meaning until we define how the information is to be interpreted externally.

    - Example: How different are these binary patterns?

      x = 00010001011111000100011001011001001
      y = 10000100001000010000100001000011110

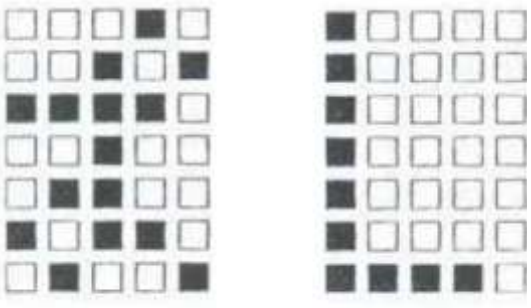
The answer hinges on how one chooses to interpret x and y.

i. x and y construed as binary numbers – then Hamming distance might be the appropriate measure
H (110, 001) =3 - the number of bits in which x and y differ

ii. Alternately, x and y considered as pattern vectors with bit position indicating the status of individual components in the pattern. Here one might employ the inner product (dot product)

x and y appear to be quite dissimilar! And yet….

If x and y are viewed as 2-dimentional patterns, the story is quite different!



This diagram illustrates how the character L can be represented as a Bitmapped vector. We interpret the pattern vectors by first chopping each 35-element vector into a 7 row * 5 column matrix. Then, by replacing each 1 in the resulting matrix with an opaque dot, and each 0 with a transparent dot, we obtain a pattern that resembles the symbol “L” as might be written (a) cursively, and (b) in block form.

Note the impact that our interpretation scheme has on the degree of similarity.

Pattern Representation Methods

Representing external parameters for an ANN model:

Factors to consider:

i. Is the data source continuous (radar signature) or discrete (Boolean function)?

ii. Range of parameters - e.g. wind speeds range from 0 to 250 mph. should we scale directly?

iii. Any interrelationship between parameters - e.g. precipitation type (rain, snow, sleet, etc.) and temperature.

iv. Any interrelationships between patterns - one may have duplicated data present.

Binary Patterns:

- Neural network PE's produce output signals that vary in magnitude, usually over (0, 1).

- Binary patterns are the easiest to represent.
  - 1 : presence of a feature 0 : its absence

Exercise: Consider a neural-network application to map a set of attributes to a room designation. For example, in a house, a living room normally contains a sofa, a coffee table, and a floor lamp, while a kitchen contains an oven, a refrigerator, and a table. Devise a data representation for a network that will accept an attribute pattern vector and produce a corresponding room designation for five different rooms. Generate a set of training patterns, consisting of a number of attribute-designation vector pairs, to show how your data representation solves the problem.

Attributes													
	Sofa	Coffee table	Floor lamp	Oven	Fridge	Table	Bed	Dresser	Shelves	Desk	Book Case	Work table	Tools
x1	1	1	1	0	0	0	0	0	1	0	0	0	0
x2	0	0	0	1	1	1	0	0	0	0	0	0	0
x3	0	0	1	0	0	0	1	1	1	1	0	0	0

And the rooms we want to classify					
	Living room	Kitchen	Bedroom	Den	Garage
y1 =	1	0	0	0	0

y2 =	0	1	0	0	0
y3 =	0	0	1	0	0

Ternary and n-ary patterns:

- Binary patterns - presence or absence of a feature sometimes there is a third possibility, a don't care state. E.g. representation for states of a game such as tic-tac-toe.
- A square may be occupied with an X, an 0 or it may be vacant.
- One may employ multiple binary inputs (or outputs). And consider each feature as an independent sub-pattern.

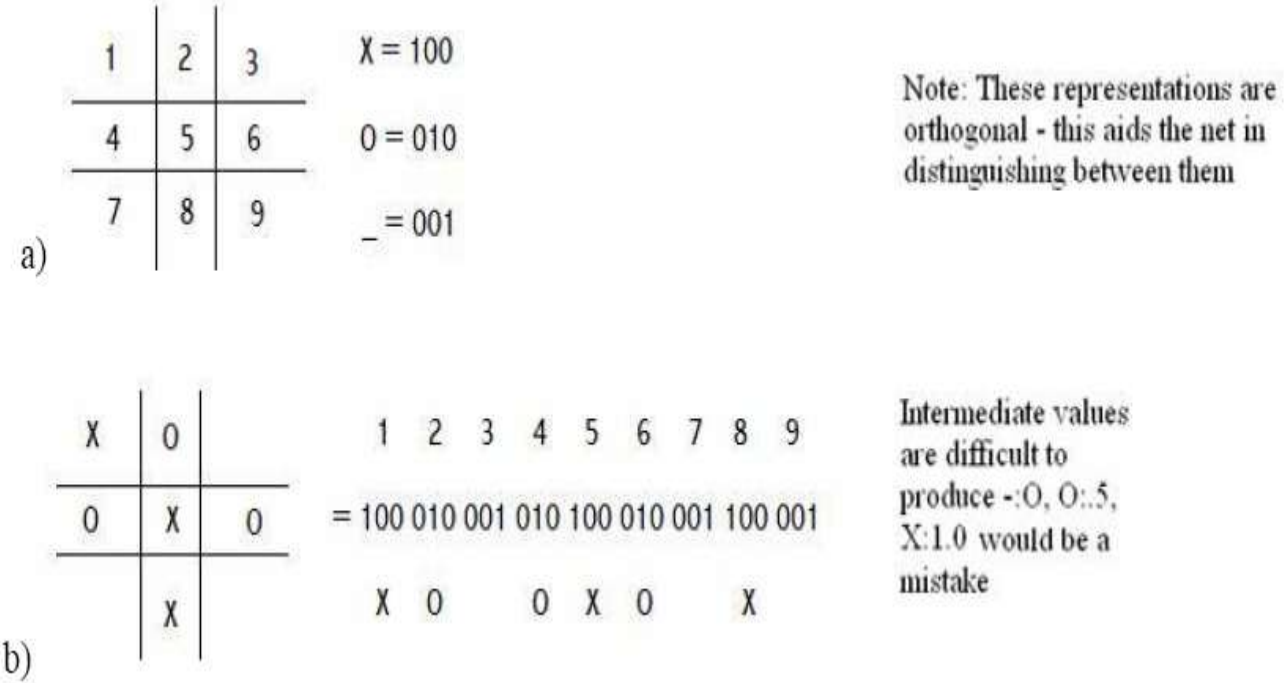


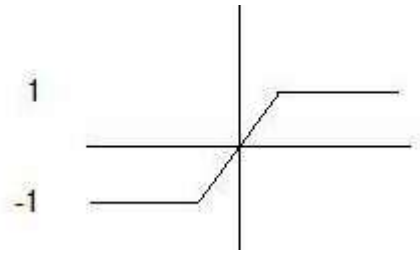
Figure 3.4: A scheme for representing the tertiary tic-tac-toe game situation is shown. In this example, each board position is allocated three pattern elements, one to indicate that the position is occupied by the "X" token, one to indicate that the position is occupied by the "O" token, and one to indicate the position is vacant. The entire data game situation then the concatenation of the nine sub-pattern representations, where the first three pattern elements

indicate the state of the first board position, the second three indicate the state of the second position, and so on.

- Sometimes scaling is required. E.g. A continuous variable ranges from 900 to 950. One should subtract 900 and divide by 50. Then,

0 = 900
.02 = 901
.04 = 902
.
.
.
1.0 = 950

One may not use sigmoid activation function here. It does not scale well!



A ramp function is preferable

Temporal Patterns

- How does one model parameters in the time domain for presentation to a spatial pattern network?

  Often one concatenates multiple discrete time patterns into a single pattern vector.

  An example: Recognizing the differences between two monochromatic photographs of the same scene taken a few seconds apart.

  1. Digitize the photographs to produce two matrices of pixels, each describing one photograph.

  2. Scale each pixel to a value between zero and one. Notice that because the pixels all come from the same image source, we can use the same scale factor for all pixels.

  3. Create an image vector for each photograph by concatenating the rows in each matrix.
  This produces two image vectors.

  4. Create a pattern vector for the network by concatenating the two image vectors.

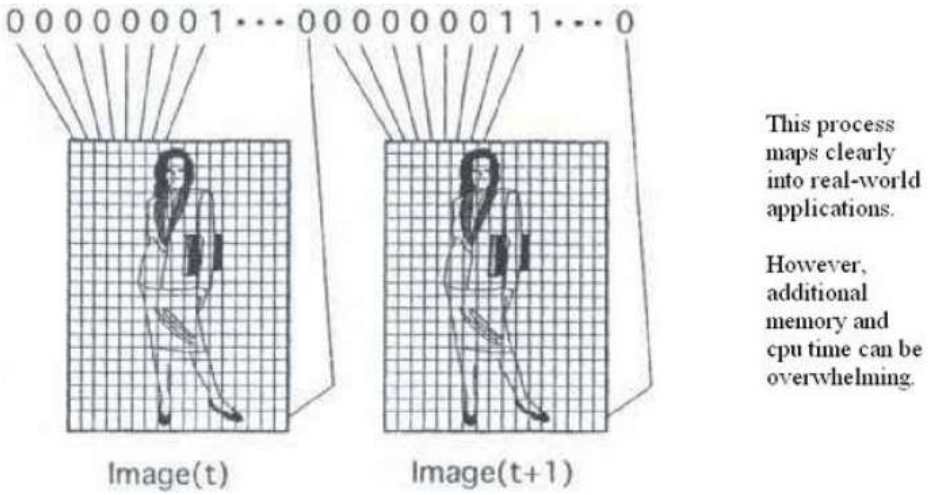


Figure 3.5: This diagram illustrates how temporal data can be encoded as a spatial pattern for examination by a neural network. Even though we show the pixel values as binary here, they could just as easily be analog values representing gray-scale intensity, or even color hue. The details of the pattern-generation process are described in the text.

Exemplar Analysis

- One has collected the application data that will be used to train the neural network and the data has been properly formatted. Now what?

- Analysis of the data is required.
  - The data collected provides an accurate representation of the application problem space.
  - There are no inconsistencies in the data that the network will not be able to resolve; and
  - Any problems uncovered as part of the analysis can be corrected without compromising the effectiveness of the network.

- If the training set is not complete, the network can produce the correct outputs but cannot identify the features in input space.

- We must provide our application with example patterns during training that not only represent the desired output but are also **contra-indicative** of the desired output.

- 1:1 rule: ~½ of the training patterns should represent the desired input-to-output pattern classification. The other half should represent **null patterns** (patterns that do not indicate the desired output).

Exemplar Analysis Contd.

- Exceptions to 1.1 rule: If one can guarantee that the network will never be used to classify an input after training that cannot be classified.

    - An example: This network will try to classify an input pattern into one of 26 classes, even when the input is *garbage*. We are struck! *There's a problem no matter what approach is taken*.

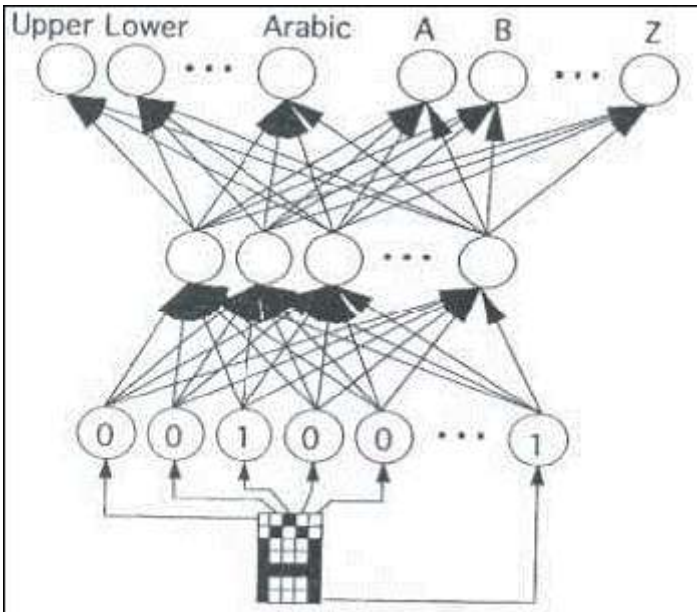


Figure 3.6: This diagram illustrates how a three-layer BPN could be constructed to classify a pixel image of a character into its corresponding type and identification. As shown on the output layer, we train the network to perform two simultaneous classifications, with a subset of the output units indicating the *type* of the character image, and the remainder identifying the character itself.

- If pre-filtering is employed (to remove bad patterns), then what is the purpose of the network - **we have already classified!** And our system does not get the benefit of noise.
- If no pre-filtering is used, then what do we consider a null pattern and how do we train for this?

Compromise - Data filtering - there are some non-blank pixels, properly framed and only one character present.

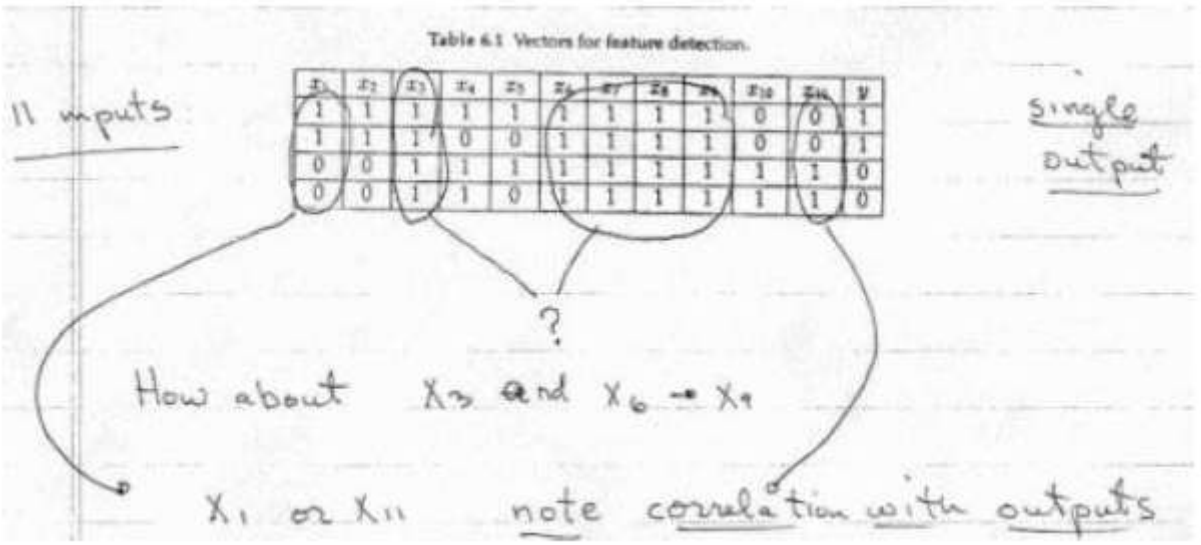
Artificial Neural Networks
Lecture Notes

Chapter 8

Contents

Hidden Nodes As Feature Extractors

- What are the features in the following training set?



- A feature: is a subset of the input space that helps us to discriminate the patterns by its examination without examining the entire pattern.
- Features contain the essential information content in the pattern set.
- If we trained a single semi-linear node using these vectors - we would expect:
  - A large positive weight to develop on input 1.
  - A negative weight on input 1 1.
- More formally - Use bipolar inputs and outputs

0 → -1
1 → 1

We have $x;^{-}_{i}, y;^{-}$

For each pattern and each component form $x;^{-p}_{i}\, y;^{-p}$ - measure of input/output correlation.

- Mean correlation C_i on the i_{th} component is

$$C_i = \sum_p \frac{1}{4} \bar{x}_i^p \bar{y}^p$$

- Plotting C_i vs. i

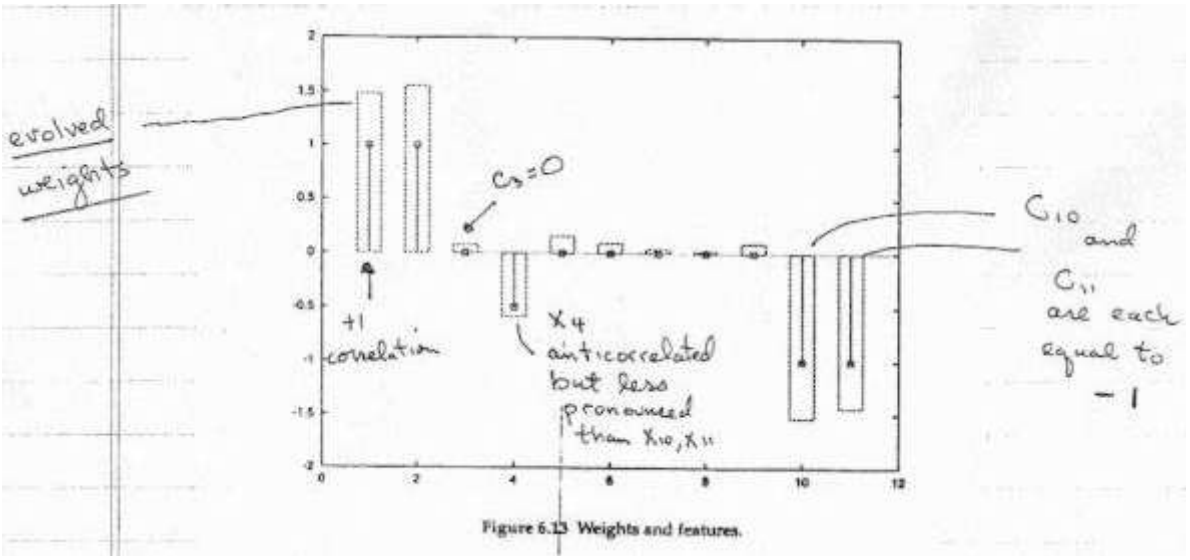


Figure 6.13 Weights and features.

Generalization and Overtraining

- In the left-hand graph (below), the training set has been "memorized" – poor generalization.

  The graph on the right has some errors but performs better on validation set.

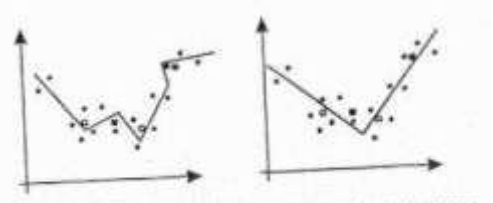


Figure 6.14 Generalization and overtraining in pattern space.

- Similar scenario:

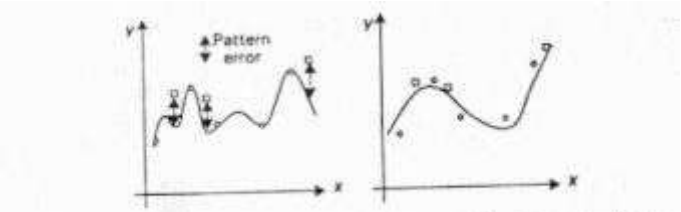


Figure 6.15 Generalization and overtraining in the context of function approximation.

- How can we avoid overtraining?

  Stop training when validation error is minimal.

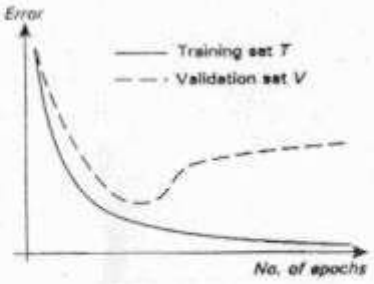


Figure 6.16 Cross-validation behaviour.

nonlinear regression. These similarities are explored further in the review article by Cheng & Titterington (1994).

On Data Representation

Internal Representation Issues
An example: *Credit Card Application*

Our database would consist of the these types of data:

- Numeric
  - income
  - expenses
  - number of current credit cards
- Category
  - Sex
  - Rent/Own
  - Checking Acct.
  - etc. ...
- Free-form
  - Text Occupation
  - Name
  - Address
  - etc. ...

- Name: Identifies the individual.
  Does not have a significant bearing on the credit-worthiness of the person.

- Address & Zipcode: Could be useful for integrating geographic block-code overlay data.
  N.B. It is illegal to use zipcode as an input by itself (*redlining*).

- Social Security Number: Could be useful as a key into other databases, but not directly useful here.

- Sex: Another field that *cannot* be used as network input!

- Marital Status: Can take on one of four values, typically encoded using a "one of N" coding - ie. one from among a set of N values.

  Example:
  M = Married; S = Single; W = Widowed; D = Divorced
  For a Married subject we may have:

M	S	W	D
1	0	0	0

- Number of Children: May range from 0 to 9 (or more) - rescale into the range 0 to 1.

- Number of Credit Cards: Similarly as above, rescale the number into the range 0 to 1.

- Occupation: Some problems here ...

More than 3,000 standard occupations and likely very few examples of each in our database.
Therefore, group them into some larger set of categories: e.g.,

Management

Professional

Skilled Labor

Unskilled Labor
etc. ...

- House Ownership: One of two categories - Rent or Own

Therefore, "one of N" encoding, as we did for the Marital status entry.
etc. ...

- Monthly Income & Expenses: - Any suggestions ???

- Checking & Savings Accounts: Flags for Y = Yes, N = No.

Thus,

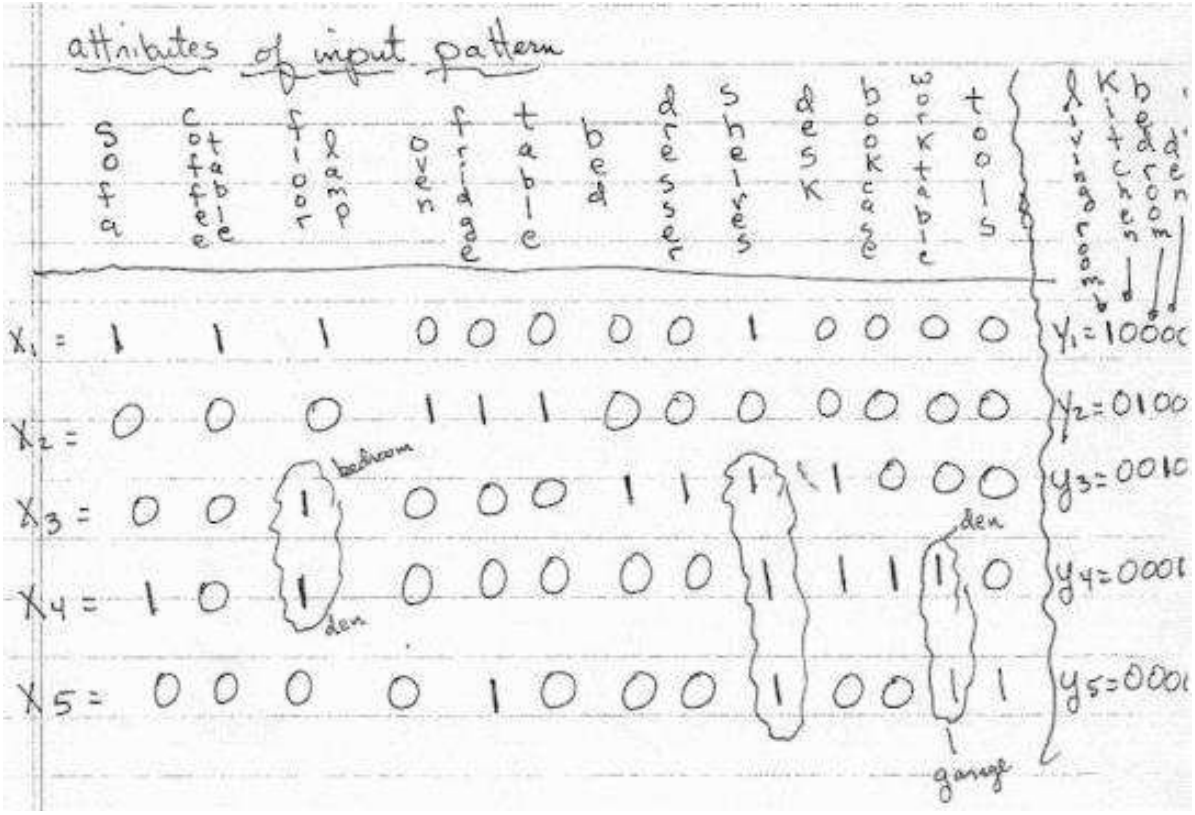
Y → 1

N → 0

or two inputs each

On Pattern Representation

- Neural network application to map a set of attributes to a room designation - room example revisited.
- Modify your previous data representation to include attributes that are common to more than one room.
- For example, a fireplace could be an attribute of a living room *and* a game room.
- Attributes of input pattern:



and what if the attribute is unknown for a certain room???

External Interpretation Issues

The processing elements within network have little or no understanding of the application.

Recall the L vs. $\mathcal{I}$ example.

Exemplar Analysis

- Accurate representation of a problem space.
- No inconsistencies.
- Problems can be corrected.

Ensuring Coverage

1 : 1 rule.
About half the inputs should be null patterns.

Exemplar Consistency Checking

Binary Search of exemplar set.

Resolving Inconsistencies

- Eliminating Patterns - if unlikely events.
- Combining Patterns - but be careful if outputs should be mutually exclusive.
- Altering the Representation Scheme - add a time or location stamp.

Training Guidelines

- Train BPN until global error falls below 0.2, note the epoch and save the state of the network for archival purposes.
- Continue training until error falls below 0.1
- If additional number of training epochs <= 30 % of original, then repeat process and try for 0.05.

After training - We test!

Partial-Set Training

Present the network with patterns not in the training set.

- Withold a number of patterns randomly

However ... network may be sensitive to the order in which training patterns are presented.

- Hold-One-Out Training: Network must be trained n times. costly!

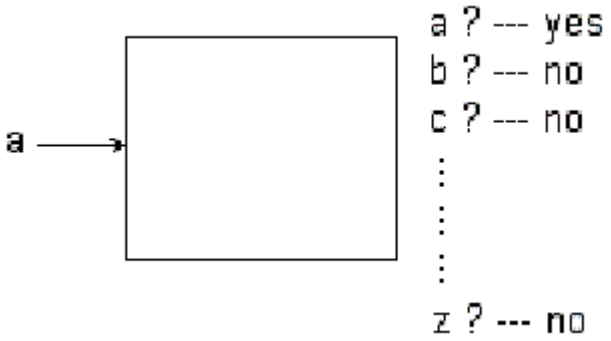
Pathology Analysis Hinton diagrams allow developer to view activity of each unit in the network.

Applications in Pattern Classification

Pattern Classification

The final state of the network is interpreted in a manner that allows us to classify the input pattern as belonging to one of several categories.

Example:
Handwritten letter analysis



Handwritten letter analysis

Applications

- NETtalk - A "talking typewriter."
- Radar Signature Classifier - Aircraft identification
- Prostate Cancer Detection - Analysis of sonargrams.

NETtalk
Developed by Terry Sejnowski, Charles Rosenberg.

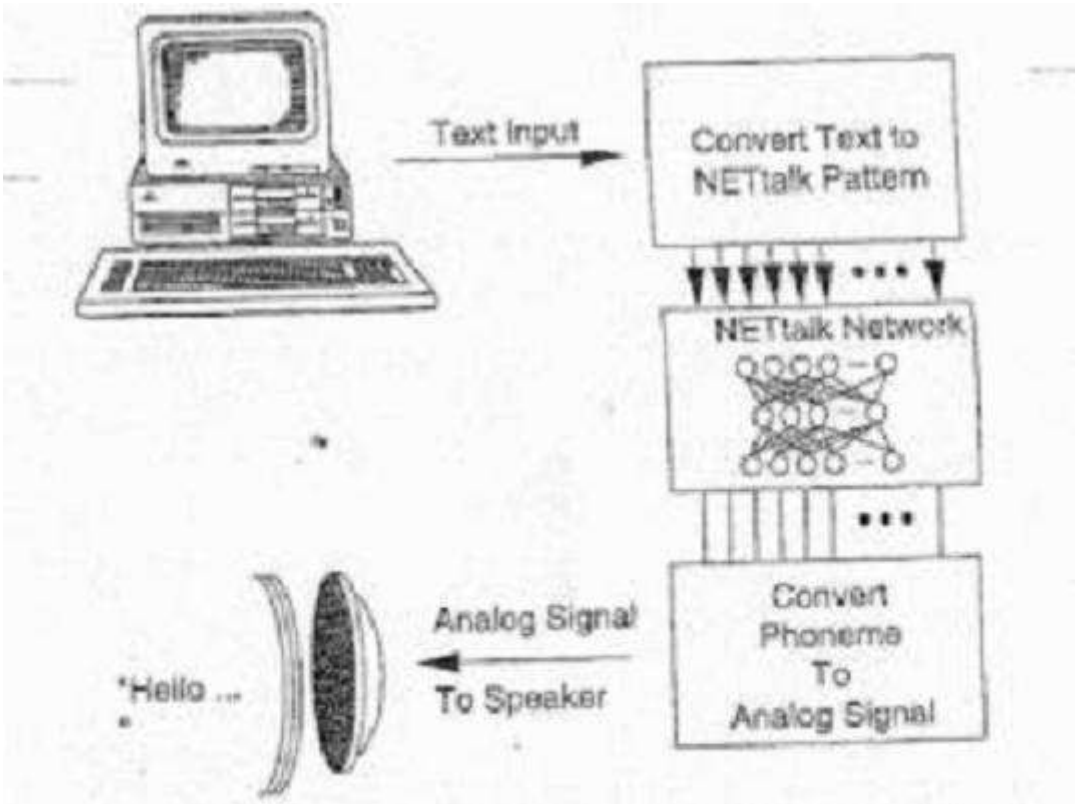


Figure 6.1 The system block diagram of the NETtalk application is shown. In this architecture, a front-end process converts the scanned text into the pattern representation for the BPN. The network, in turn, produces an output that is interpreted by a back-end process as the phoneme to be generated. The output of the system is therefore a synthetic pronunciation of written text.

A BPN is trained to classify a character sequence as one of 26 possible *phonemes* which are used to generate synthetic speech.

Pronunciation in English is problematic.
"Throw out the rules."

Consider

- *rough* vs. *through*
- *pizza* vs. *fizzy*
- *tortilla* vs. *villa*
- etc. ...

The pronounciation of the vowel(s) is dependent on a learned relationship between the vowel and its neighboring characters
(e.g., Note the *e* in *h<u>e</u>lp* and *h<u>e</u>ave*.)

NETtalk captures the relationship between text and sounds by using a BPN to learn these relationships through experience.

The textual representation of the word is converted into a pattern that the network can use.

Sejnowski and Rosenberg use a sliding-window for text.

The preceding three letters and the following three provide a context for a letter.

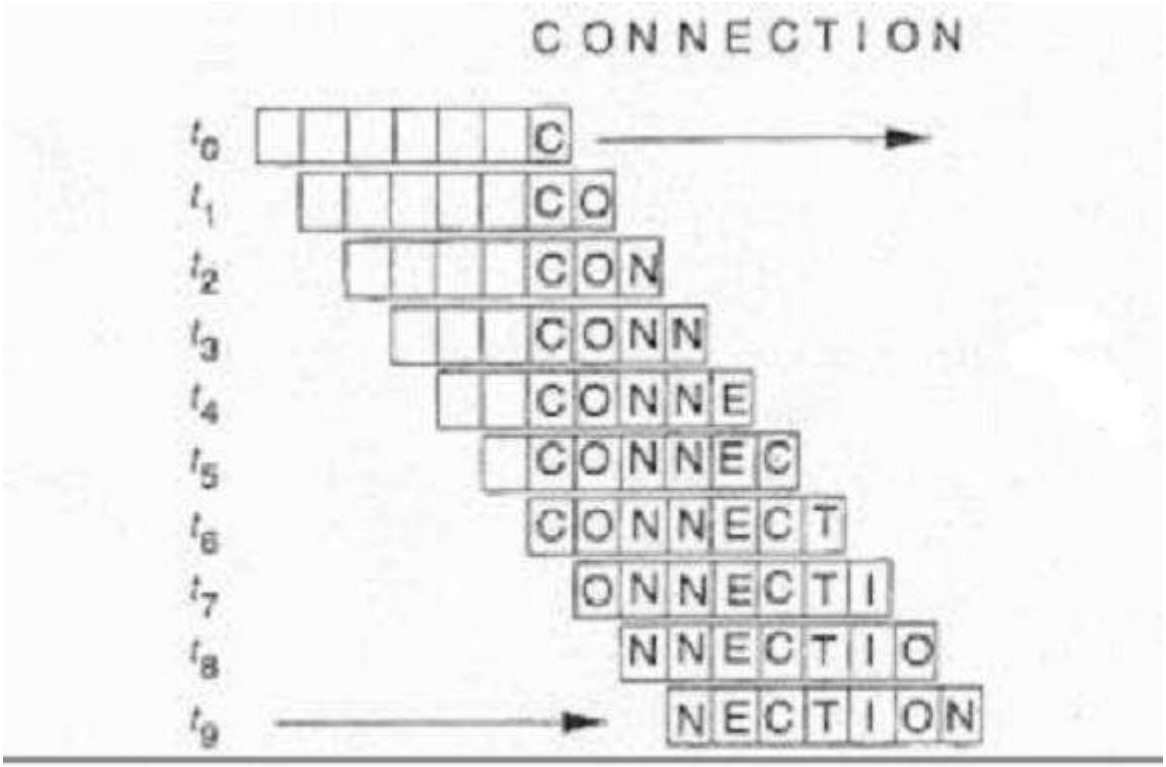


Figure 6.2 This diagram illustrates the sliding-window concept used to produce word patterns for the neural network. Initially, the window is padded with blanks to produce the silent phoneme. Characters from the word to be pronounced are then acquired by sliding the window over the word, one character at a time. This process continues until all of the characters in the word have passed by the focus position, with blank characters appended to the word to signal termination.

Letters are converted into pattern vectors consisting of 29 binary elements:

- 26 uppercase English alphabet characters.
- Three for punctuation characters that influence pronounciation
- "One of N" encoding (pattern characters are orthogonal.)

NETtalk Training

- Training Data:

5000 common English words together with the corresponding phonetic sequence for each word.

- Input Patterns:

203 binary inputs per pattern.
(a 7-character window) * (29 elements/char).

- Size of training set:

Average word length = 6 characters. Thus,
(5000 words) * (6 char/word) = 30,000 exemplars.

NETtalk Architecture

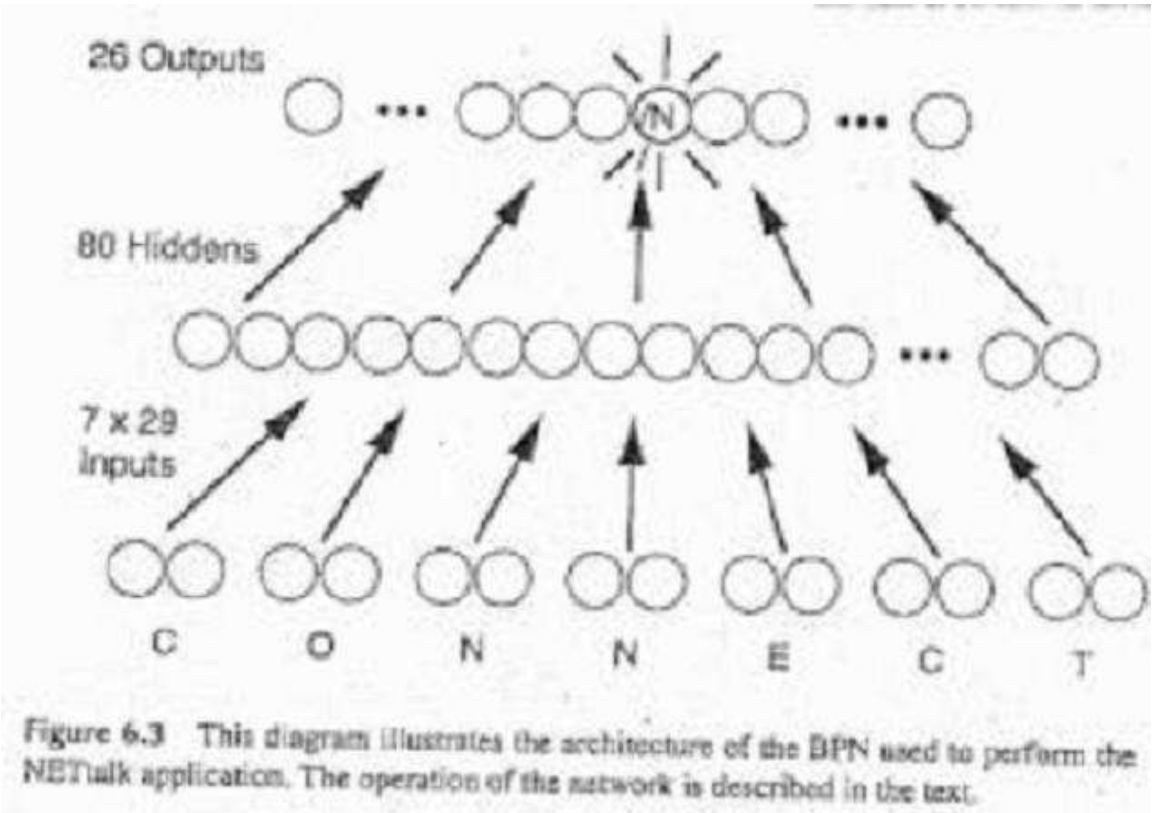


Figure 6.3 This diagram illustrates the architecture of the BPN used to perform the NETtalk application. The operation of the network is described in the text.

NETtalk Results

The classification produced by the network is converted into the proper phoneme, and used to drive a speech synthesizer.

- Before Training

The network produced random sounds, mixing consonant and vowel sounds.

- After 100 epochs

The network began to separate words, recognizing the role of blank characters.

- After 500 epochs

Clear distinction between vowel and consonant sounds

- After 1000 epochs

Words distinguishable but not phonetically correct.

- After 1500 epochs
  The network captured phonetic rules - correct pronunciation, but mechanical sound.

- Training stopped and NETtalk was given 200 words to pronounce (*obviously* NOT from the training set

- NETtalk can read English text with an accuracy of "about 95%".

Hence, the network did not merely memorize a set of words and pronunciations. It learned the *general interrelations* between English text and sounds.

- NETtalk learning vs. a child learning to read.

Radar Signature Classifier

Pulse Doppler Radar - Uses

- Detecting airborne targets.

- Traffic law enforcement.

Determining *range* and *velocity* of target relative to the radar system.

Physical Principles and Problem Formulation

- Electromagnetic radiation (*EMR*) travels at a constant speed.

- EMR waves reflected from a moving body are frequency shifted in the direction of travel (The Doppler Effect, or "Red-Shift").

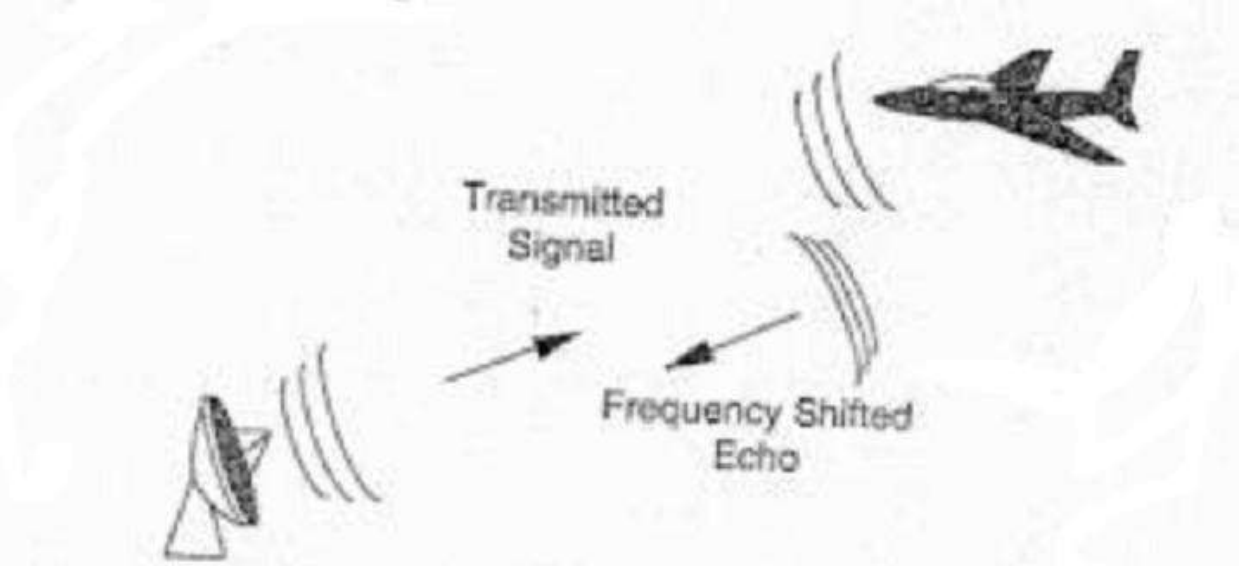


Figure 6.4 This diagram illustrates the basic principles that govern the operation of pulse Doppler radar. A radar pulse, or *chirp*, is transmitted from the radar station. The EMR burst travels outward from the transmitter at the speed of light. When an EMR-reflective target is hit by the outgoing signal, some of the energy is reflected back to the radar receiver, which then determines the range and velocity of the target based on concepts described in the text.

- Each radar target has its own unique way of reflecting radar emissions.

- Skilled radar operators are able to determine the type of target, even though the radar itself has no capability of making this determination.

The straightforward approach:
Use a fixed-frequency radar pulse.
Determine *range* as a function of the delay between transmission and the reception of an *echo*.

Determine *velocity* as a function of the *phase-shift* in the return frequency.

- Radio waves travel at a constant rates. The elapsed time between the transmitted and received signals provides the distance to the target.

Therefore,

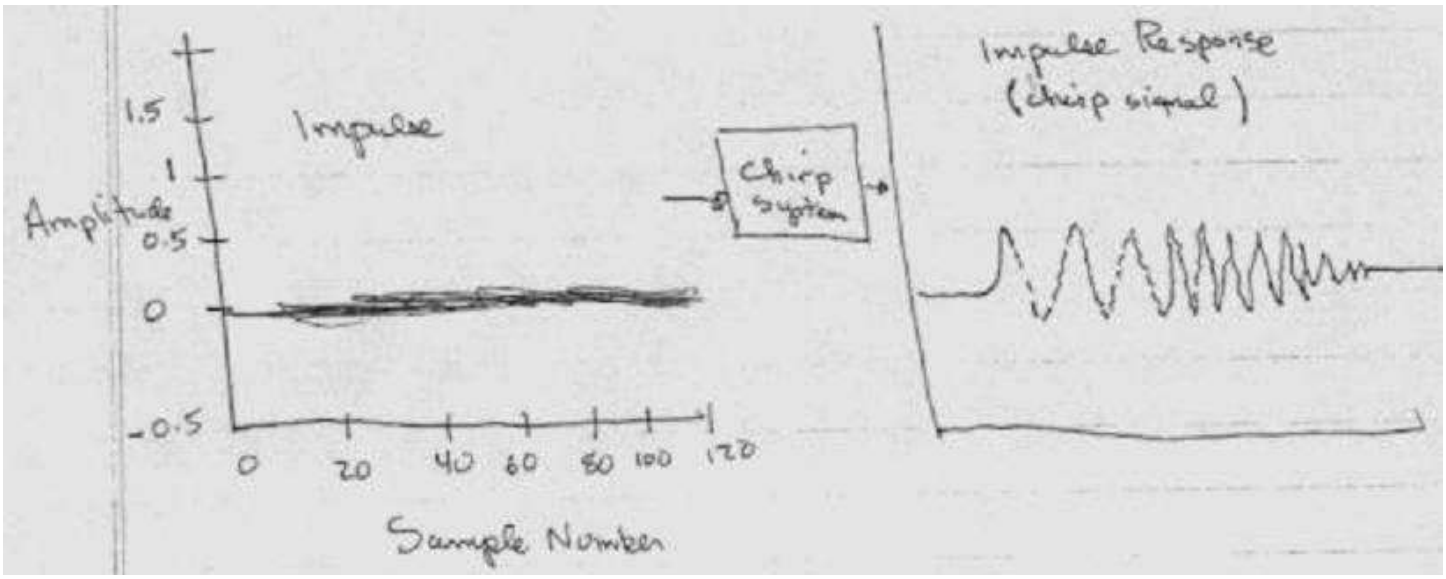
- *First Requirement for the pulse*:
  It needs to be as short as possible.

  Example:
  A 1 microsecond pulse provides a radio burst about 300 meters long.

- *Second Requirement for the pulse*:
  If we want to detect objects farther away, we need more energy in the pulse. Unfortunately, more energy and shorter pulse are conflicting requirements.

- ***Chirp signals***
  Chirp signals provide a way of breaking this limitation. Before the impulse reaches the final stage of the radio transmitter, it is passed through a *chirp system*.

- The reason it is called a chirp signal is because it sounds like the chirp of a bird, when played through a speaker.



- A *Fourier Transform* is used in processing.

- Inspection of the Doppler signature reveals information about the nature of the target.

  Example:If the airframe of the target has no moving blades (jet aircraft?) the Doppler signature is fairly compact.

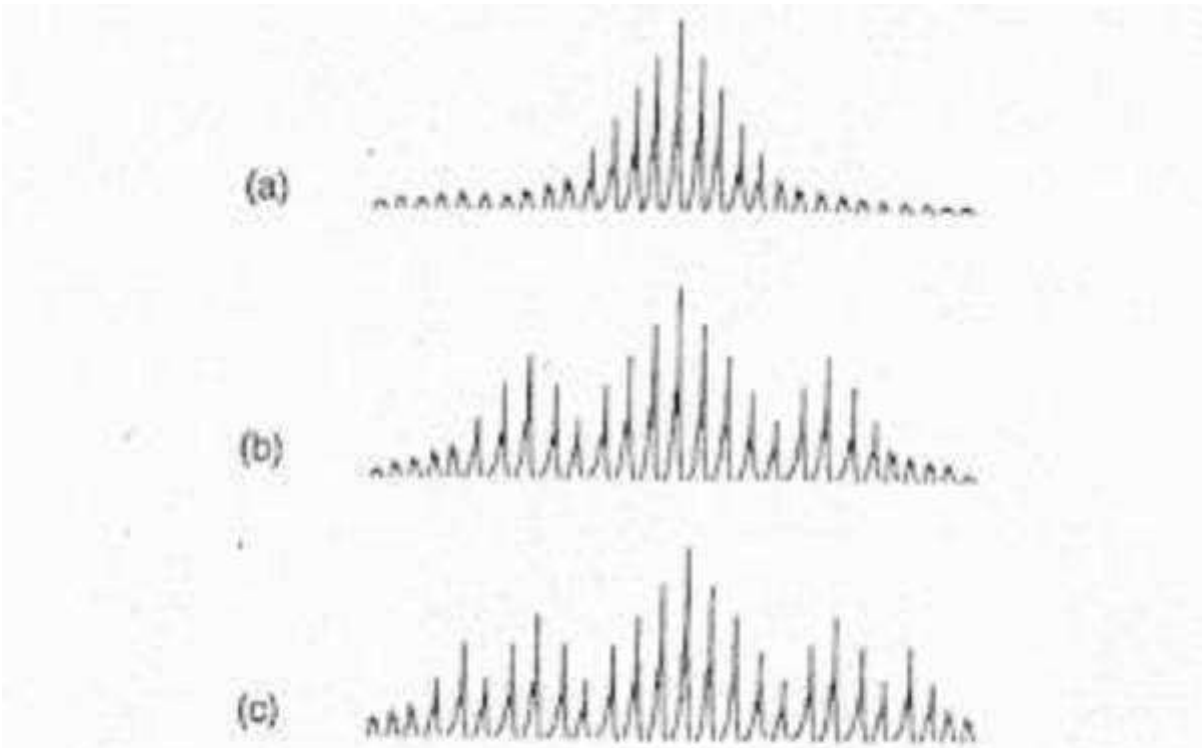


Figure 6.8 These diagrams illustrate Doppler signatures produced by different kinds of aircraft. (a) A jet (airframe only) produces a compact Doppler signature. (b) A two-bladed helicopter causes a spread in the signature, with side lobes forming on both sides of the peak, indicating rotation of the blades. (c) A four-bladed helicopter has a very wide signature, because blades rotate through the radar signal twice as often as with the two-bladed helicopter.

Learning

To Learn to classify these Doppler signatures, create a network that learns to produce the correct output for each signature.

Problem:

BPN is not a position-invariant pattern classifier.

Also, Doppler signatures shift as a function of target range:

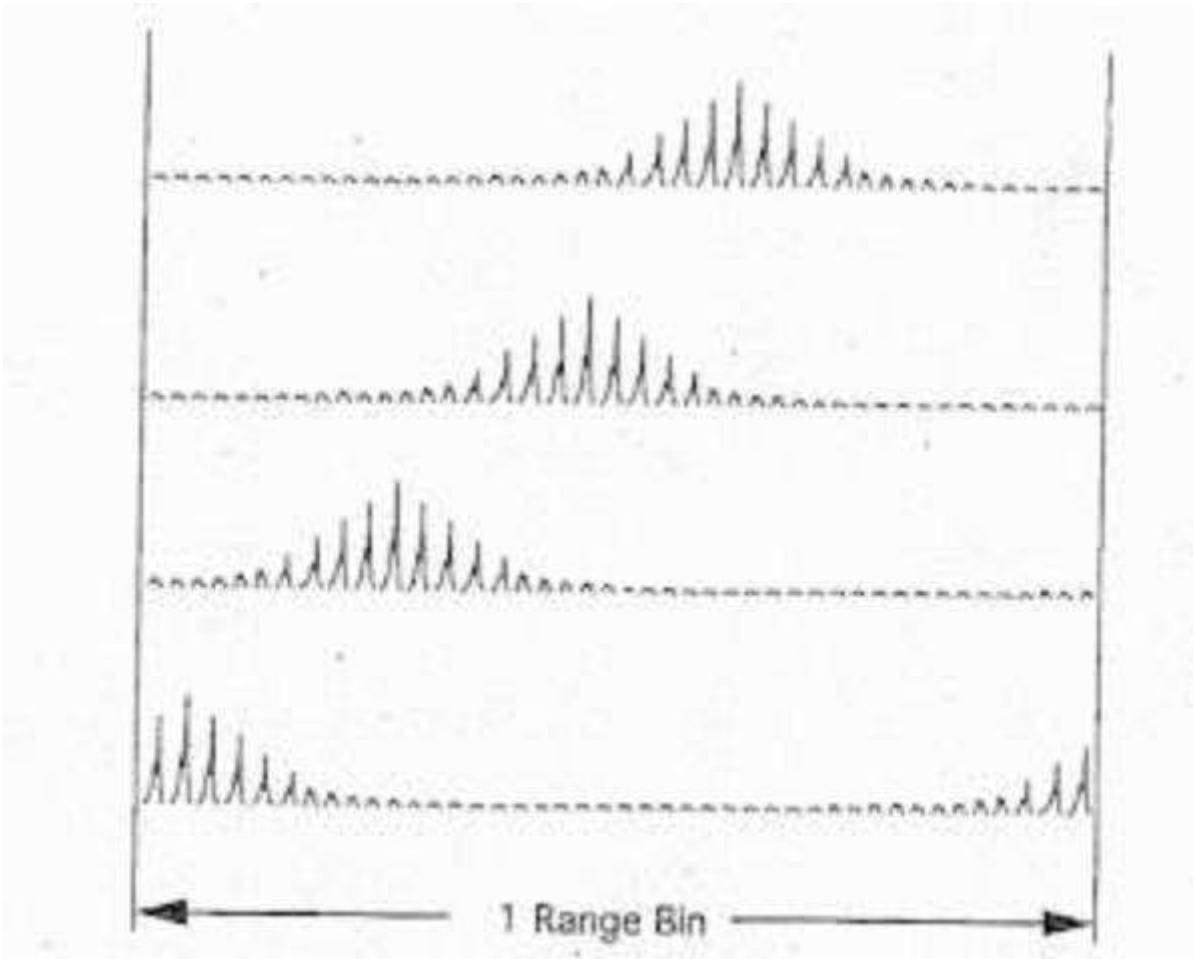


Figure 6.9 These graphs show how a Doppler signature shifts as a function of target range. Beginning at the top, the sequence shows a target approaching the radar station, with a corresponding left shift in the Doppler signature.

To compensate for the pattern shift, the network must be trained with exemplars that account for the pattern shift.

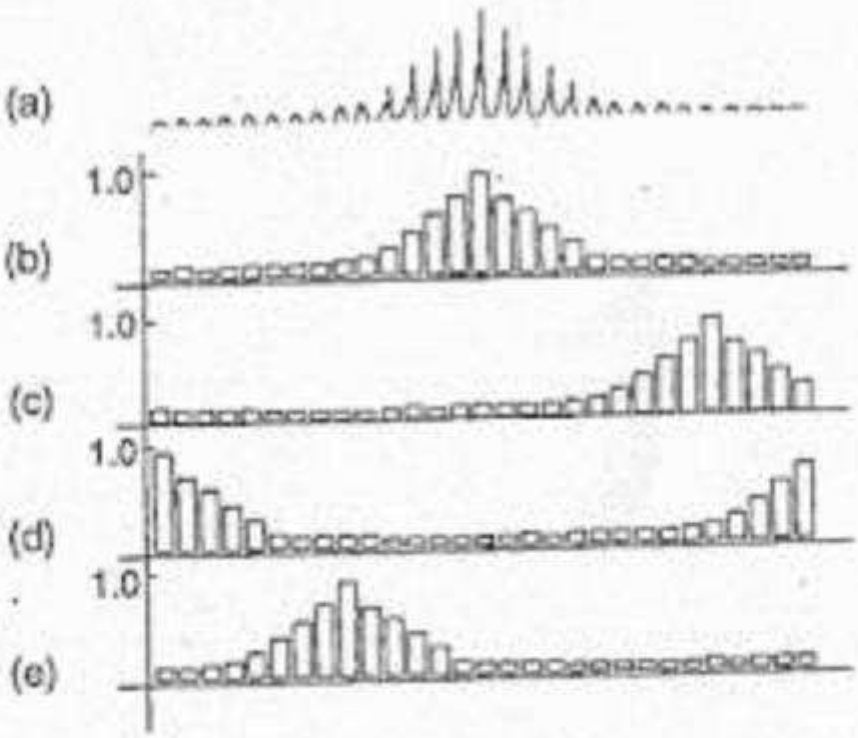


Figure 6.10 This diagram illustrates the exemplar encoding scheme used to capture the Doppler signature of various aircraft. (a) This diagram illustrates the general form of the radar return after extracting the signature. (b) The quantized form of the Doppler signature. (c) The quantized signature shifted right by ten range bins from the general form. (d) The quantized signature shifted by fifteen range bins. (e) The quantized signature shifted right by 26 range bins (or left by 6 range bins).

Architecture of the Network

- Three-layer BPN.
- 32 input units.
- 8 units on the hidden layer.
- 3 output units.

Training

- 96 Doppler signature aircraft-classification pairs.
- Learning rate n = 0.5
- Momentum α = 0.6

- 1700 epochs resulted in error of 0.01.

  $\Delta w(n) = \alpha \delta^{j}(n) x(n) + \lambda w(n-1)$

Artificial Neural Networks
Lecture Notes

Chapter 9

Contents

Fuzzy Logic

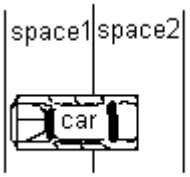
- Neural networks can learn. However, they are rather opaque.
- "The network acts like a black box by computing a statistically sound approximation to a function known only from a training set.", Rojas.

- Fuzzy logic has an explanatory capability. However, it is unable to learn

Fuzzy Sets Vs. Crisp Sets

- "Raise your hand if you're male" ... hands down. Now,
- "Raise your hand if you're female" ... hands down.
- These are *crisp sets*.
- "Raise your hand if you're satisfied with your jobs" ... hands down. Now,
- Some hands probably went up both times, and
- Hands may have been only somewhat raised in each case.
- We would say that job satisfaction is a *fuzzy concept*, in that most people are not entirely satisfied or dissatisfied with their jobs.

- Parking spaces in a lot would be another example:
Most of the time one has a situation like the one shown on the right.



- *Fuzzy logic* was developed by Lotfi Zadeh during the mid 1960's.
- Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set

- The subset A of X consisting of x_1 alone, can be described by the n-dimensional membership vector

  Z(A) = (1, 0, 0, ... 0).

- The subset B of X with elements x_1 and x_n is described by the vector

Z(A) = (1, 0, 0, ... , 1).

- Any other crisp subset of X can be represented in the same way by an n-dimensional binary vector.

- Consider the fuzzy set C

  Z(C) = (0.5, 0, 0, ... , 0).

In classical set theory, this would be impossible. For classically, either $x_1 \in C$ or it doesn't. The world is black and white in classical set theory.

- With *Fuzzy* set theory, this type of description is permitted.
- The element x_1 belongs to the set C only to some extent.
- The degree of membership is expressed by a real number in the interval [0,1].
- An example: "x_1 is a tall person"

  Is a 6-foot individual tall? ... yes.
  but a 7-foot person is taller?

- Other diffuse statements:
  x is old
  y is young
  z is mature

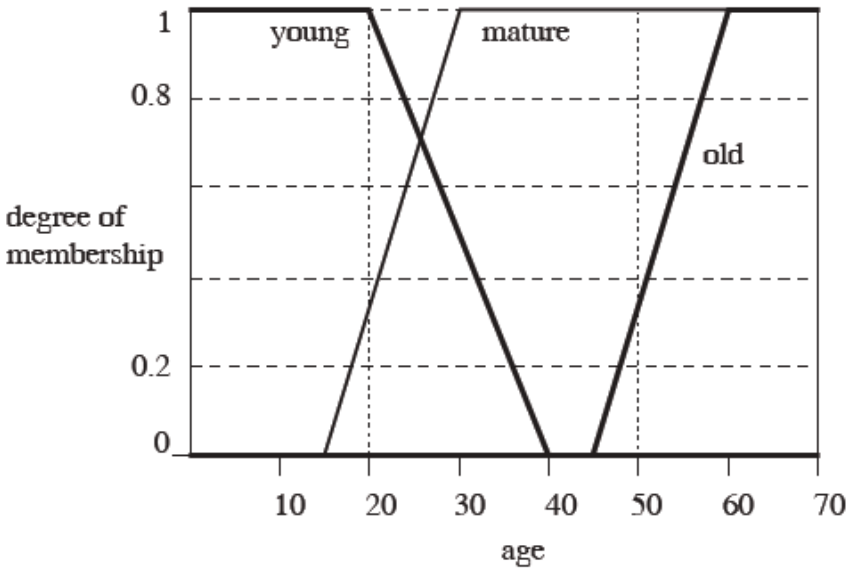


Fig. 11.1. Membership functions for the concepts young, mature and old

- The three functions define the degree of membership of any given age in the sets of young, mature and old ages.
- A 20-year old - degree of membership in the set
  of young people is 1.0
  of mature adults is 0.35 , and
  of old persons is 0.0.
- If someone is 50 years old, the degrees of membership are 0.0, 1.0, 0.3 in the respective sets.
- Definition 11.1:

Let X be a classical universal set. A real function $\mu_A : X \rightarrow [0,1]$ is called the **membership function** of A, and defines the fuzzy set A of X.
This is the set of all pairs $(x, \mu_A(x))$ with $x \in X$.

- A fuzzy set is completely determined by its membership function.
- The ***set of support*** of a fuzzy set A, is the set of all elements x of X for which $(x,\mu_A(x)) \in A$, and $\mu_A(x) > 0$ holds.
- A fuzzy set A with finite set of support $\{a_1, a_2, \ldots , a_m\}$ can be described as:

$A = \mu_1 / a_1 + \mu_2 / a_2 + \ldots + \mu_m / a_m$
where $\mu_i = \mu_A(a_i)$ for $i = 1, \ldots , m$.

- Let $X = \{x_1, x_2, x_3\}$.

The classical (crisp) subsets $A = \{x_1, x_2\}$ and $B = \{x_2, x_3\}$ can be represented as

$$A = 1/x_1 + 1/x_2 + 0/x_3$$

$$B = 0/x_1 + 1/x_2 + 1/x_3$$

- The union of A and B is computed by taking for each element x_i the maximum of its membership in both sets. I.e.,

$$A \cup B = 1/x_1 + 1/x_2 + 1/x_3 .$$

- The fuzzy union of two fuzzy sets can be computed in the same way.

Let

$$C = 0.5/x_1 + 0.6/x_2 + 0.3/x_3$$

$$D = 0.7/x_1 + 0.2/x_2 + 0.8/x_3$$

Then

$$C \cup D = 0.7/x_1 + 0.6/x_2 + 0.8/x_3 .$$

- The fuzzy intersection of two sets can be defined in a similar manner (min instead of max).

Geometric Interpretation of Fuzzy Sets

- Let $X = \{x_1, x_2\}$ be a universal set.
- Each point in the interior represents a subset of X
- The crisp subsets of X are located at the vertices of the unit square.

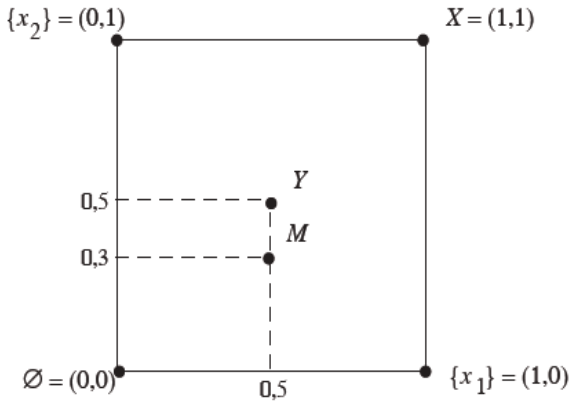


Fig. 11.2. Geometric visualization of fuzzy sets

- The point (1,0) represents the set x_1.

The point (0,1) represents the set x_2

etc. ...

- The crisp subsets of X are located at the vertices of the unit square.
- The inner region of a unit hypercube in an n -dimensional space - the fuzzy region.
- The point M (in the figure above) corresponds to the fuzzy set

$M = 0.5/x_1 + 0.3/x_2$.

- Center of square - most diffuse of all fuzzy sets.
  $Y = 0.5/x1 + 0.5/x2$

- The **degree of fuzziness** of a fuzzy set can be measured by its entropy (here different from the entropy used in information theory or physics.)

- The **entropy** (index of fuzziness, crispness, certitude, ambiguity) corresponds inversely to the distance between the representation of the set and the center of the unit square.

- The set Y in the previous figure has the maximum entropy.

- The vertices represent the crisp sets and have the lowest entropy, i.e. 0

Entropy of a Fuzzy Set M

- Quotient of the distance of the nearest corner from M to the distance d_2 from the corner which is farthest away.
  $E(M) = d_1/d_2$.
  $0 \leq$ entropy of a set ≤ 1.

- Maximum entropy - at the center of square.

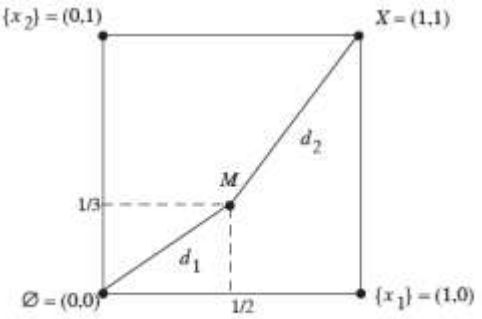


Fig. 11.3. Distance of the set M to the universal and to the void set

Visualization for Union and Intersection of Fuzzy Sets

- The membership function for the *union* of two sets A and B is

$$\mu_{AUB}(x) = \max(\mu_A(x) , \mu_B(x)) , \forall x \in X.$$

- This corresponds to the maximum of the corresponding coordinates in the geometric visualization.

- The membership function for the *intersection* of two sets A and B is given by

$$\mu_{A \cap B}(x) = \min(\mu_A(x) , \mu_B(x)) , \forall x \in X.$$

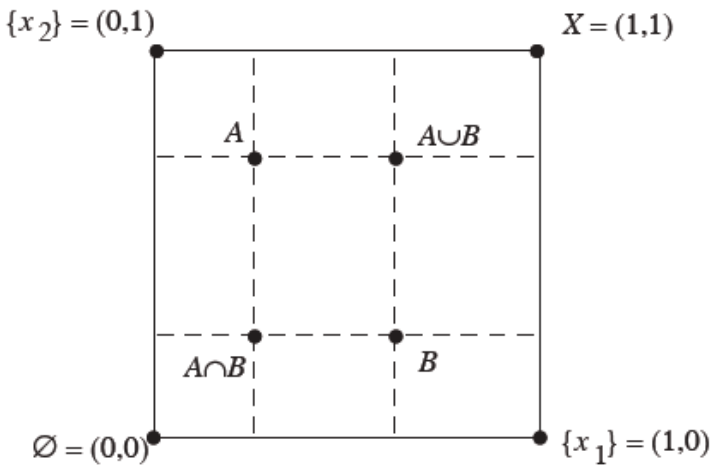


Fig. 11.4. Intersection and union of two fuzzy sets

- The union or intersection of two fuzzy sets is in general a fuzzy, not a crisp set.
- The *complement* A^C of a fuzzy set is given by

$$\mu_{A^c}(x) = 1 - \mu_A(x) \ , \ \forall \ x \in X.$$

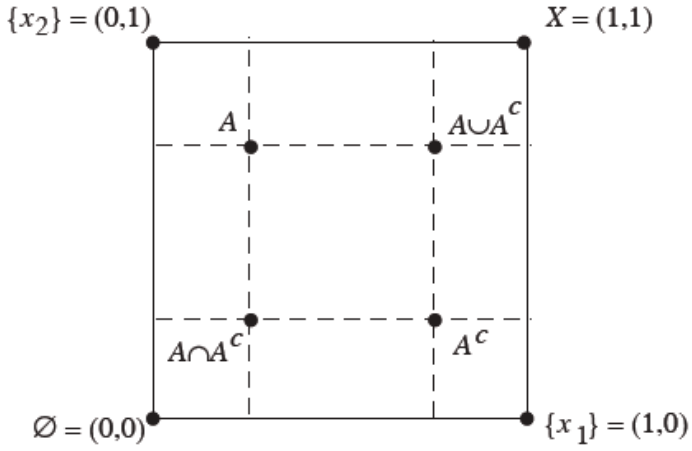


Fig. 11.5. Complement A^c of a fuzzy set

The line joining the representation of A and A^C goes through the center of the square.

- In general, for fuzzy sets we have

$A \cup A^C \neq X$, and

$A \cap A^C \neq 0$, and

Fuzzy Sets and Logic:

On the Relationship between Set Theory and the Propositional Logic

- Let A, B be two crisp sets. We have,
  $\mu_A , \mu_B : X \rightarrow \{0,1\}$.

- The membership function μ_{AUB} for $A \cup B$ is

  $\mu_{AUB}(x) = \mu_A(x) \vee \mu_B(x) , \forall x \in X$

where 0 False , 1 True.

- Similarly, for the membership function for $A \cap B$, we have
  $\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x) , \forall x \in X$

- For the complement A^C,

$\mu_{Ac}(x) = \neg\mu_A(x)$.

- ***De Morgan's Laws***

$(A \cup B)^C \equiv A^C \cap B^C$ corresponds to $\neg(A \vee B) \equiv \neg A \wedge \neg B$

$(A \cap B)^C \equiv A^C \cup B^C$ corresponds to $\neg(A \wedge B) \equiv \neg A \vee \neg B$

- *Fuzzy Counterparts to Logic Operators*

We may identify:

the OR operation ($\vee; \bar{}$) with *maximum*,

the AND ($\wedge; \bar{}$) with *minimum*, and

Complementation ($\neg; \bar{}$) with $x \rightarrow 1 - x$.

- Then we may write:

  $\mu_{AUB}(x) = \mu_A(x) \vee; \bar{} \mu_B(x) , \forall x \in X$

  $\mu_{A \cap B}(x) = \mu_A(x) \wedge; \bar{} \mu_B(x) , \forall x \in X$

  $\mu_{Ac}(x) = \neg; \bar{} \mu_A(x), \forall x \in X$

- The properties of the isomorphism present in the classical theories are preserved.

- Many classical logic rules are valid for fuzzy operators.

E.g., *min* and *max* are commutative and associative.

- But note:

  the principle of no contradiction does *not* hold.

  E.g., proposition A with truth value 0.4:

  $A \wedge\!;\bar{}\ \neg\!;\bar{}A = \min(0.4, (1 - 0.4)) \neq 0.$

- Similarly, the ***Law of Excluded Middle*** is not valid either.

  $A \wedge\!;\bar{}\ \neg\!;\bar{}A = \max(0.4, (1 - 0.4)) \neq 1.$

Axiomatic Definitions of Fuzzy Operators

Fuzzy OR Operator

- Axiom μ_1: *Boundary Conditions*

  $0 \vee\!;\bar{}\ 0 = 0$

  $1 \vee\!;\bar{}\ 0 = 1$

  $0 \vee\!;\bar{}\ 1 = 1$

  $1 \vee\!;\bar{}\ 1 = 1.$

- Axiom μ_2: *Commutativity*

  $a \vee\!;\bar{}\ b = b \vee\!;\bar{}\ a.$

- Axiom μ_3: *Monotonicity*

  If $a \leq a'$ and $b \leq b'$,

  then $a \vee\!;\bar{}\ b \leq a' \vee\!;\bar{}\ b'$.

- Axiom μ_4: *Associativity*

  $a \vee\!;\bar{}\ (b \vee\!;\bar{}\ c) = (a \vee\!;\bar{}\ b) \vee\!;\bar{}\ c.$

- Axiom μ_5: *Idempotence*

  $a \vee\!;\bar{}\ a = a.$

- The *maximum* function satisfies $\mu_1 \rightarrow \mu_4$

Fuzzy AND Operator

- The fuzzy AND is monotonic, commutative and associative.

Boundary Conditions

$0 \wedge; \bar{} \; 0 = 0$

$1 \wedge; \bar{} \; 0 = 0$

$0 \wedge; \bar{} \; 1 = 0$

$1 \wedge; \bar{} \; 1 = 1.$

Fuzzy Negation

- Axiom N_1: *Boundary Conditions*

  $\neg; \bar{} \; 0 = 1$

  $\neg; \bar{} \; 1 = 0$

- Axiom N_2: *Monotonicity*

  If $a \leq b$, then $\neg; \bar{} \; b \leq \neg; \bar{} \; a$.

- Axiom N_3: *Involution*

  $a = \neg; \bar{} \; \neg; \bar{} \; a$.

- Graphs of the functions max and min (i.e., a possible fuzzy AND and fuzzy OR combination,)

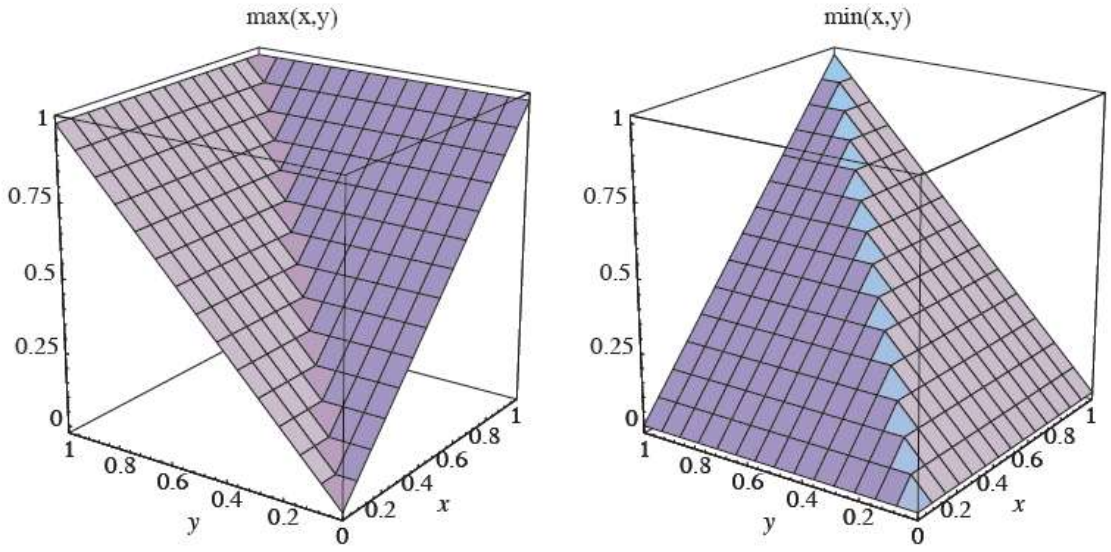


Fig. 11.7. The max and min functions

- They could be used as activation functions in neural networks.
- Using output functions derived from fuzzy logic can have the added benefit of

providing a logical interpretation of the neural output.

Fuzzy Inferences

- Fuzzy logic operators can be used as the basis for inference systems.
- Fuzzy inference rules have the same structure as classical ones.

  e.g., rule R1: if (A $\wedge;^{-}$ B) Then C
  R2: if (A $\vee;^{-}$ B) Then D.

  Note $\wedge;^{-}$ ~ min and $\vee;^{-}$ ~ max, though other choices are possible.

- Let the truth values of A and B be 0.4 and 0.7. Then

  A $\wedge;^{-}$ B = min(0.4, 0.7) = 0.4

  A $\vee;^{-}$ B = max(0.4, 0.7) = 0.7

- Fuzzy inference mechanism - Rules R1 and R2 can only be partially applied (i.e., rule R1 is applied 40% and rule R2 70%.)

- The result of the inference is a combination of the propositions C and D.

Second Example:

A temperature controller regulates an electrical heater.
Three rules for this system:

- R1 : If (temperature = cold) then heat.
- R2 : If (temperature = normal) then maintain.
- R3 : If (temperature = warm) then reduce power.

Assume for a temperature = 12° C

Membership degree of 0.5 to set of cold temperature
Membership degree of 0.3 to set of normal temperature
12° C is first converted into a fuzzy category

- T = cold/0.5 + normal/0.3 + warm/0.0 [note different notation]

Next T is used to evaluate R1, R2 and R3 in parallel.

A fuzzy inference is obtained:

- action = heat/0.5 + maintain/0.3 + reduce/0.0 with
  Expert Systems or Fuzzy Controllers - one must obtain a crisp value.

Transforming a measurement into fuzzy categories

Triangular or trapezium-shaped membership functions are often used.

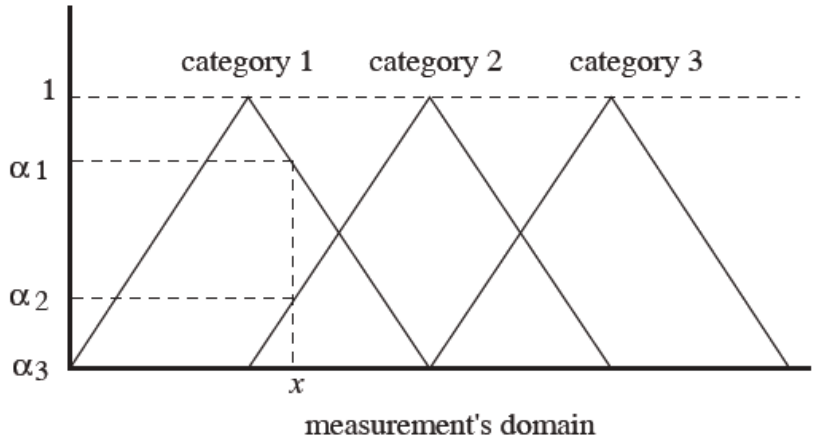


Fig. 11.10. Categories with triangular membership functions

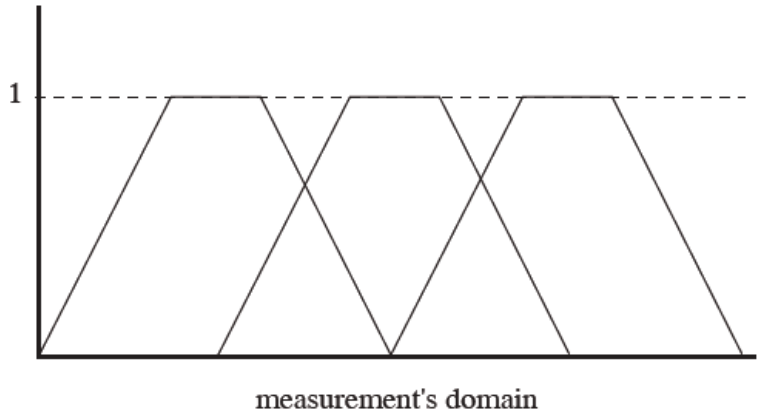


Fig. 11.11. Categories with trapezium-shaped membership functions

The transformation of a measurement x into a fuzzy category is given by the membership values $\alpha 1$,
$\alpha 2$ and $\alpha 3$ derived from the membership functions.

How does one transform the membership values $\alpha 1$, $\alpha 2$ and $\alpha 3$ back into the measurement x? i.e. How to implement the inverse operation?

Centroid Method

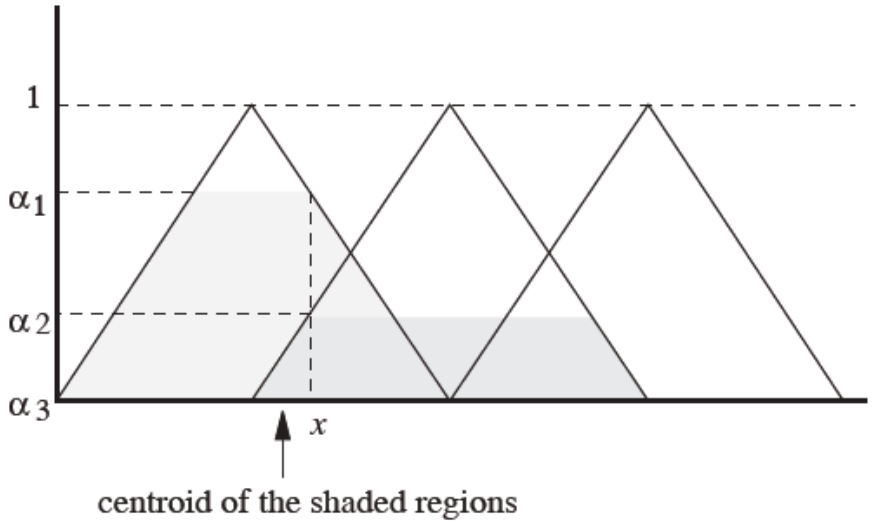


Fig. 11.12. The centroid method

- From $\alpha 1$, $\alpha 2$ and $\alpha 3$ we can reconstruct the original x.
- The surfaces of the triangular regions limited by the heights $\alpha 1$, $\alpha 2$ and $\alpha 3$ are computed.
    - The horizontal component of the centroid of the total surface is the approximation to x we are looking for
- For all x values where at least two of the three numbers $\alpha 1$, $\alpha 2$ and $\alpha 3$ are $\neq 0$, we obtain a good approximation.

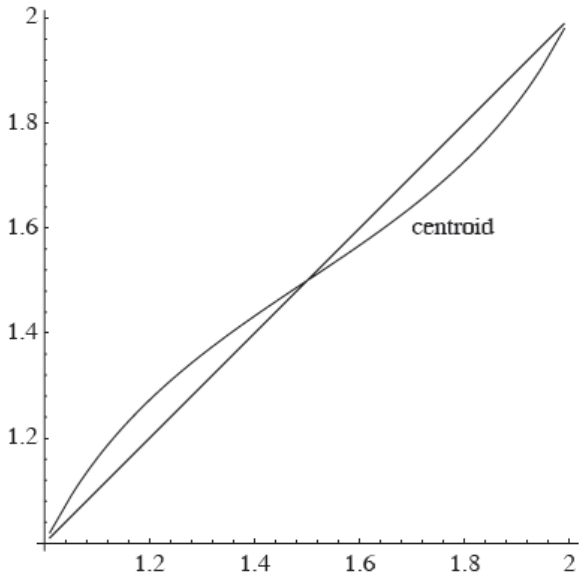


Fig. 11.13. Reconstruction error of the centroid method

- x vs approximation to x
  - Basis of Δ's of length 2 height 1
  - x chosen in [1, 2]
  - Relative difference ≤10%

Fuzzy Controllers

- A fuzzy controller is a regulating system whose mode of operation is specified with fuzzy rules.
  - Electrical heater example - revisited
  - Assume room temperature between 0° and 40° C.

    The controller can vary electrical power consumed between 0 and 100 (in some Units)
    where 50 is the normal standby value.

  - Membership functions for the temperature categories are "cold", "normal" and "warm" and for the control categories are "reduce", maintain" and "heat".

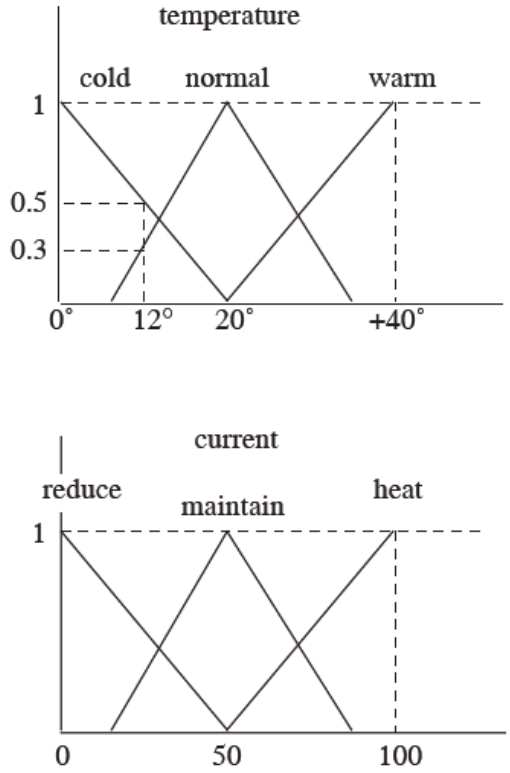


Fig. 11.14. Membership functions for temperature and electric current categories

12° C corresponds to the fuzzy numbers.

- T = cold/0.5 + normal/0.3 + warm/0.0

These values lead to inference action:

- heat/0.5 + maintain/0.3 + reduce/0.0

○ The controller must transform this into a definite value.3

- The surfaces of the membership triangles below the inferred degree of membership are calculated.

○ Light shaded surface ≡ “heat” valid to 50%

○ Darker region ≡ “maintain” valid to 30%
The centroid of the two shaded regions lies at about 70.

- This value for the power consumption is adjusted by the controller to heat the room.

Fuzzy Networks

Fuzzy systems can be represented as networks. The computing units (neurons) must implement fuzzy operations.

Network with four hidden units:

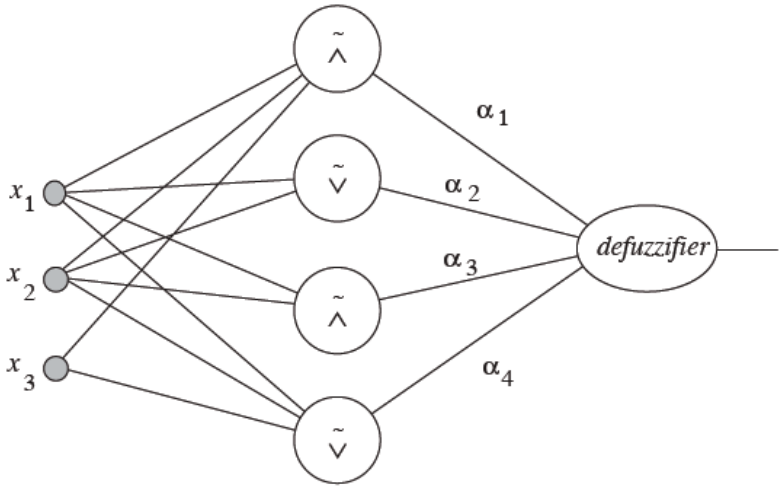


Fig. 11.16. Example of a fuzzy network

- Inputs x1, x2 and x3 correspond to the fuzzy categorization of a specific number.
- The fuzzy operators are evaluated in parallel in the hidden layer of the network, which corresponds to the set of inference rules.
- The last unit is the defuzzifier - transforms the fuzzy inferences into a specific control variable.

  - Fuzzy systems do not usually lead to deep networks.

  - The importance of each fuzzy inference rule is weighted by the numbers 1, 2 and 3 as in a weighted centroid computation.

  - Fuzzy operators cannot be computed exactly by sigmoid units, good approximation if bounded sum or difference used.

The Eye as a Fuzzy System - Color Vision

- Photons of different wavelengths, corresponding to different colors in the visible spectrum impinge on the retina.

- The eye contains receptors for only three types of colors (contrast with cochlea and hearing)

- Receptors, which are maximally excited with light from the spectral regions corresponding to the colors: blue, green, red.

    - **Visible spectrum**: 400 - 650 nanometers wavelength
      400 - 6500 angstrom units

- Monochromatic color will excite all three receptor types. The output of each receptor class depends on wavelength.

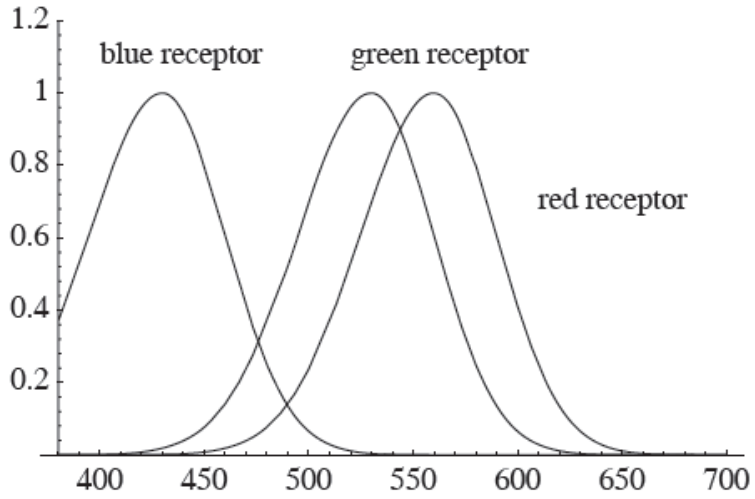


Fig. 11.18. Response function of the three receptors in the retina

- Maximum excitation for each receptor:
  Blue ~430 nm
  Green ~530 nm
  Red ~560 nm

- Monochromatic light is thereby transformed into three different excitation levels, i.e. into a relative excitation of the three receptor types.

- The wavelength is transformed into a fuzzy ~category, just as with fuzzy controllers.

- The three excitation levels measure the degree of membership in each of the three color categories: **blue**, **green** and **red**.

- Coding of the wavelength using three excitation values reduces the number of rules needed in subsequent processing.

- Sparseness of rules in fuzzy controllers counterpart in the sparseness of the biological components.

Artificial Neural Networks
Lecture Notes

Chapter 10

Contents

Financial Modeling

- Predicting success for one's investments.
- In any model, the underlying component is time.

Converting Temporal Sequences into Spatial Patterns

- We have to convert *temporal sequences* (eg, is market up since yesterday?) into *spatial patterns*.

- In order to do this, *Discrete Time Sampling* is employed:

A continuously variable signal is quantized at regular time intervals. A sequence of n samples is concatenated to form a single pattern that encapsulates a quantized signature of the signal.

The Dow Jones Industrial Average

- The Dow Jones Industrial Average is a single value that provides an instantaneous indication of the state of the market.

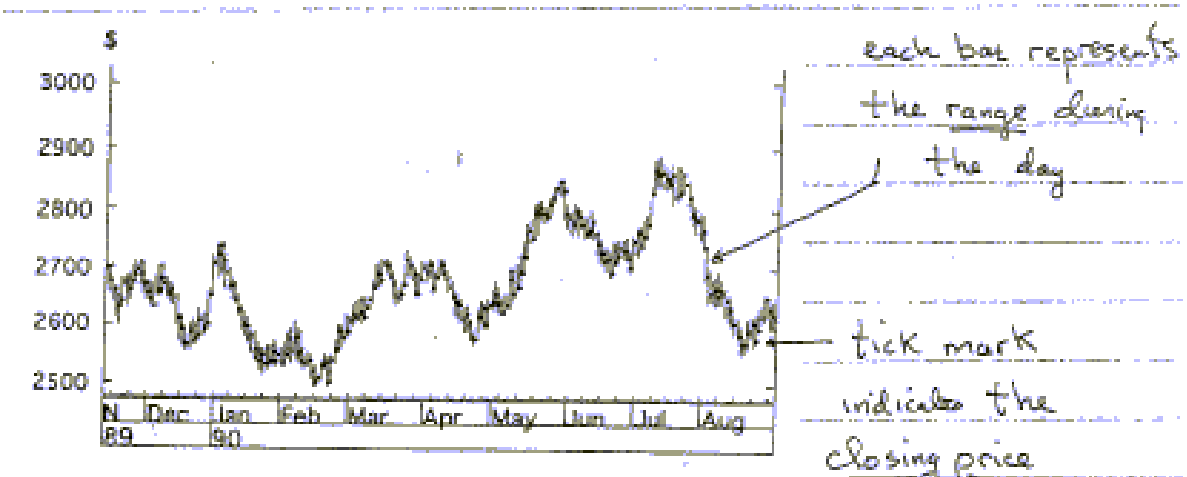


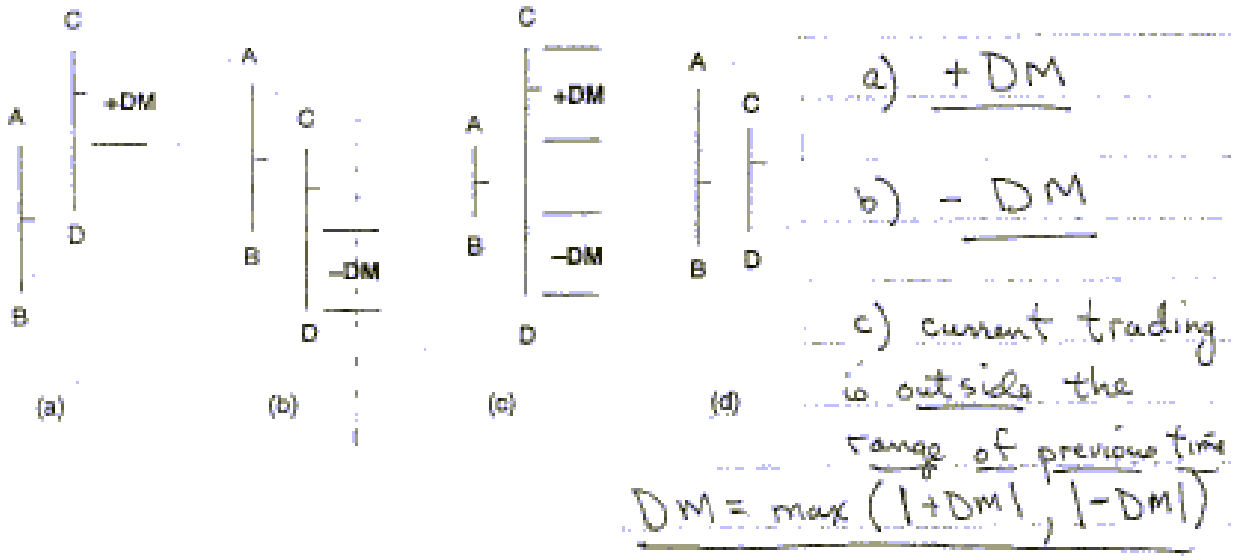
Figure 5.1 This graph shows the daily high and low of the Dow Jones Industrial Average from November 1989 through August 1990. In this chart, the vertical axis represents the numerical value of the average, while the horizontal axis depicts the passage of time. Each bar represents the range of the average during the day, while the tick mark on the range indicates the closing average. Thus, we have quantized the dynamic behavior of the market by taking a daily sample of the high, low, and closing points of the average. When looking at this chart, do you see a series of discrete points, or do you see a line moving from left to right, indicating the long-term behavior of the market?

However ...

- The Dow Jones Average, by itself, offers practically no insight into what the market might do tomorrow.
- Gaining insight into the direction the market (or a particular stock) may be going, a financial analyst will consider certain mathematical indicators:
  - *ADX* - market-intensity indicator - is the market trending?
  - *MACD*- moving-average convergence/divergence - optimal buy & sell signals in a trending market.
  - *Slow Stochastic Analysis* - complements MACD.

Average Directional Movement

- ADX: The high and low values of the market at the current time are compared with the high and low values at the previous time.
- The difference values obtained indicate the *plus directional movement* (+DM) and the *minus directional movement* (-DM)



- In the above figure, the vertical lines represent the high and low points during a given day, and the tick on a vertical line represents the closing value for that day. We have,

(a) +DM
(b) -DM
(c)Current trading is *outside* the range of previous time

DM = MAX(|+DM| , |-DM|).

(d) The trading range at the current time is *inside* the range of trading at previous time.

DM=0.

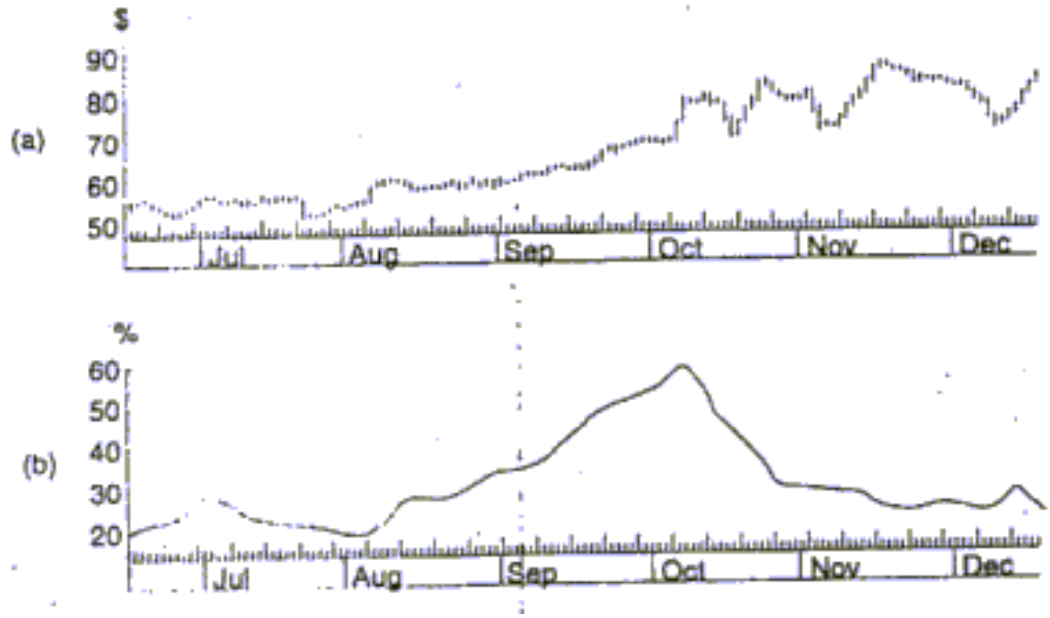
- DI: The *Directional Indicator* (wilder) is the percentage of the price range that is directional for the given time period.

DI = DM / TR

where TR is the *True Range*.

- TR is the largest of:
  - The difference between the current high and low value
  - The difference between the current high and the previous closing value
  - The difference beetween the current low and the previous closing value.
- Note: DI may be positive or negative.
- Positive Notation - wilder, defines two separate indicators
  - +DI indicates a time period with positive DI.
  - -DI absolute value of DI where DI is negative.

- ADX - *Average Directional Movement* is a *smoothed* moving average of the DI values across an interval of n time periods.
- It is useful to convert DI (Directional Indicator) to a *Directional Movement Index* (DMI) which indicates the magnitude of the trend on a scale from 0 to 100.
- The ADX is computed as an n-period moving average of the DMI.



(a) Discrete time behavior of a stock on NYSE over six months.
You can see *daily fluctuations* yet *long-term increase*. (b) ADX for the same stock.

Notice how ADX peaks concomitant with point in time when stock growth trend diminishes.

Stochastic

- *Stochastic Oscillator* is a signal designed to anticipate sudden reversals in market value (George C. Lane).
- A well-known sock market phenomenon:

A market top, or high point for a particular stock, is usually indicated by daily closing prices that tend to cluster around the high value for the stock.

A market bottom is indicated when daily closing prices cluster around the low value of the stock.

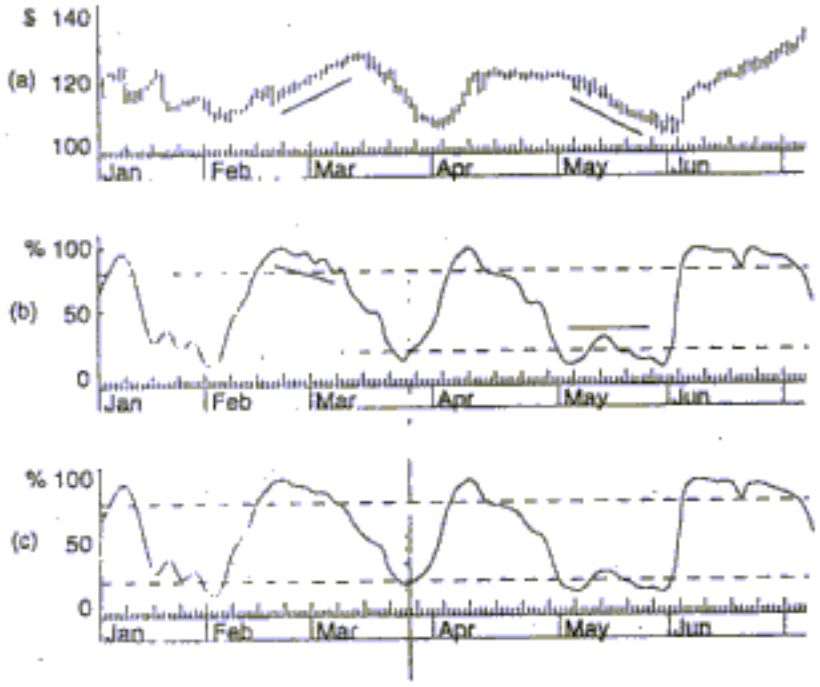
- *Stock prices tend to reverse their trends during a top (or bottom) period*.
- We can anticipate reversals by detecting when a stock is at (or near) its limit.

- We can develop such an indicator by comparing the current closing price of a stock, with its highest high and lowest low values over a period of time.

- *Lane's indicators* are a mathematical comparison over some fixed period of time (5 - 14 days) of the closing value...(illegible -cut off in photocopy)

%K - Comparison for 14 days
%D - Three-Day smoothed version of %K indicator.

- Daily closing prices of a stock:

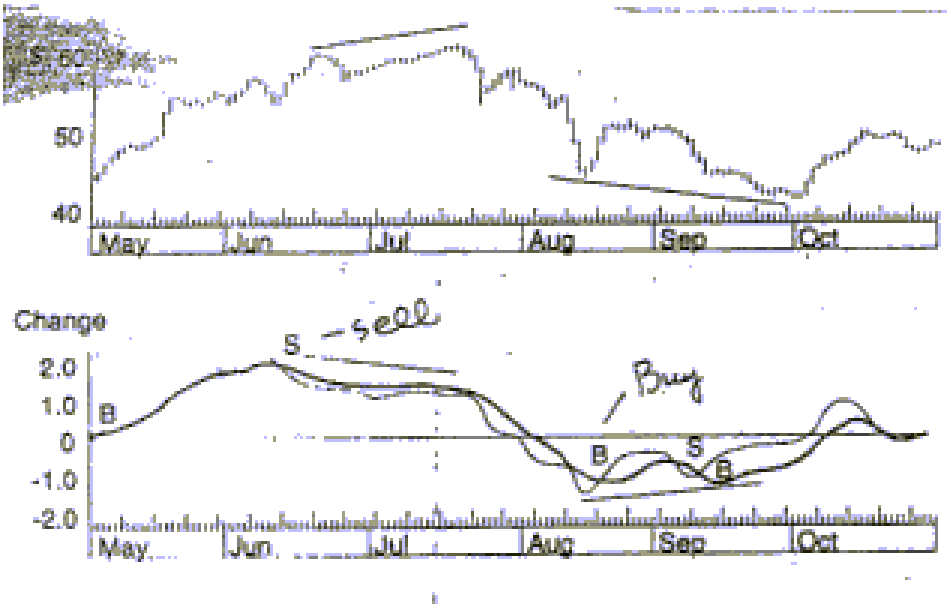


(a) Sell periods indicated by stochastic.
(b) Buy period indicated at right. Indicators have gone below 20%.
(c) A slow stochastic indicator tends to smooth data (uses three-day moving average of the %D indicator.)

- A stock is considered overbought when the stochastic indicator goes above 80%.
- A stock is considered oversold when the indicator goes below 20%.

Moving Average Convergence/Divergence

- *Moving Average Convergence/Divergence* or MACD measures the trend of a stock over a period of time.



Top graph: Closing price of a stock

Bottom graph: MACD computed.

- Note in the graph that:
  - Buy signals tend to proceed periods of increasing stock value.
  - Sell signals tend to proceed periods of decreasing stock value.
- To be successful in the capital management business, one must beat the market, i.e., to forecast what the market will do in the near future.
- Is the market *chaotic*?... and hence unpredictable?
- Yet models employing mathematical indicators have met with success.
- Barr, Loick of LBS Capital Management, and Fishman of Eckard College showed success at predicting the Standard & Poors (S&P) 500 Index five days into the future.

Network Architecture

- Backpropagation
- n inputs - where n = number of market indicators used.
- Normalized indicator values prior to training (BPN requires input values $\in[0,1]$.)
- Number of layers - 3 ? Indicate success with 4.
- Output - a single linear unit - a scaled prediction of the amount of change in the S&P 500 Average, five days into the future.
- **Exercise 5.1**: In their articles, Barr and Loick do not provide any specific guidance with regard to selecting indicators (beyond the four described here) that might improve the performance of the network, although they do indicate that their best network contained a total of 26 indicators. From your understanding of the

BPN, describe the selection criteria you would apply to determine if a financial indicator could improve the performance of the network.

The Network

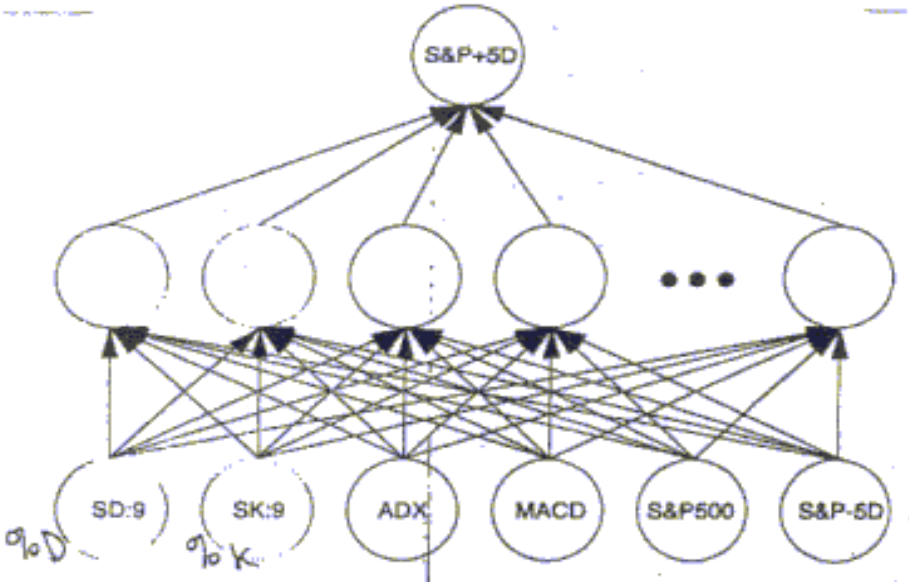


Figure 5.6 The three-layer BPN architecture used at LBS Capital Management to predict the S&P 500 average five days in the future is depicted. In this example network, the indicators used are the slow %D and slow %K stochastic indicators, using $n = 9$, an 18-day ADX, and histogram (difference) MACD values. Also used as input to the network are the current value of the S&P 500, as well as the net change in the S&P 500 value from five days prior. The output of the network is a continuously variable signal that is interpreted as a scaled estimate of the change expected in the market five days hence. *Source: Adapted from* Using neural networks in market analysis [4]. *Used with permission. Copyright ©1991, Technical Analysis, Inc.*

three layer implementation

net change in S&P from 5 days in past

- Primary source of data - recent history of S&P 500.
- Training examples developed from historical market data.
- Performance of the trained network - can be evaluated by comparing actual market performance with the projections made by the network using current market data.
- Initial training data to 20 exemplars (was the LBS network overwhelmed by too much data or was training on conflicting exemplars occuring?)
- Network results:

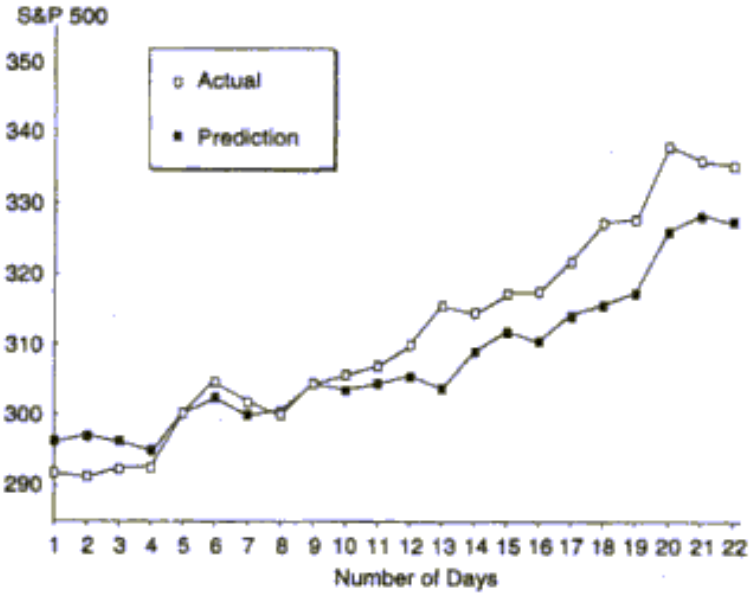


Figure 5.7 The response of the market-prediction BPN after training is shown. As this graph illustrates, the network has its lowest error in the earlier estimates, and tends to diverge from actual market values as time goes on. *Source: Adapted from* Using neural networks in market analysis [4]. *Used with permission. Copyright ©1991, Technical Analysis, Inc.*

Best estimates occur in the near future.

- Exercise 5.2: The graph in Figure 5.7 [above] shows that the neural network tends to become less accurate as the prediction date becomes more distant from the time when the training data were collected. Suggest a strategy that could be employed to reduce this error in the network's response. Explain the advantages and disadvantages of your approach.

Genetic Algorithms

- Genetic Algorithms (GA) are a parallel search method.
- A population of points encloses a local maximum of the target function after some iteration:

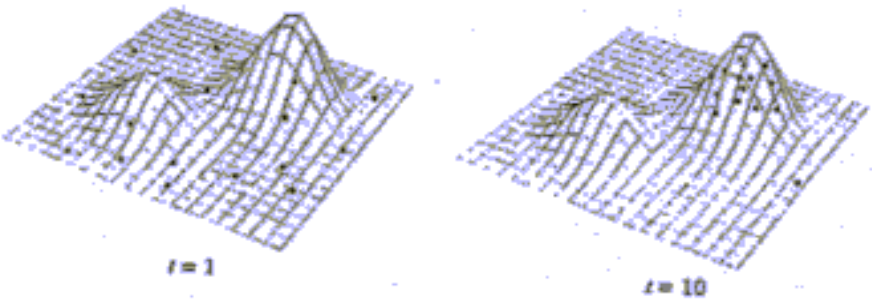


Fig. 17.1. A population of points encircles the global maximum after some generations.

- GA is a blind search - i.e., no information need be known about the search space.

- The parallel and iterative nature of the search comes at a price: Increased computation time.

- Unlike purely local search methodologies (e.g., gradient descent,) GA will not become stuck in local maxima or minima.

- Through evolution, the networking pattern of biological neural networks has been created and improved.

GA's Vs. Other Stochastic Methods

- Random Search

Starting point $\bar{x} = (x_1, x_2, \ldots, x_n)$ vs. randomly generated and $f(x_1, x_2, \ldots, x_n)$ is computed.

- A vector $\bar{\delta} = (\delta_1, \ldots, \delta_n)$ is randomly generated and f is computed at $(x_1+\delta_1, \ldots, x_n+\delta_n)$.

IF the value of f is thereby decreased, THEN $(x_1+\delta_1, \ldots, x_n+\delta_n)$ is taken as the *new search point* and the algorithm is started again.

ELSE a new direction is generated.

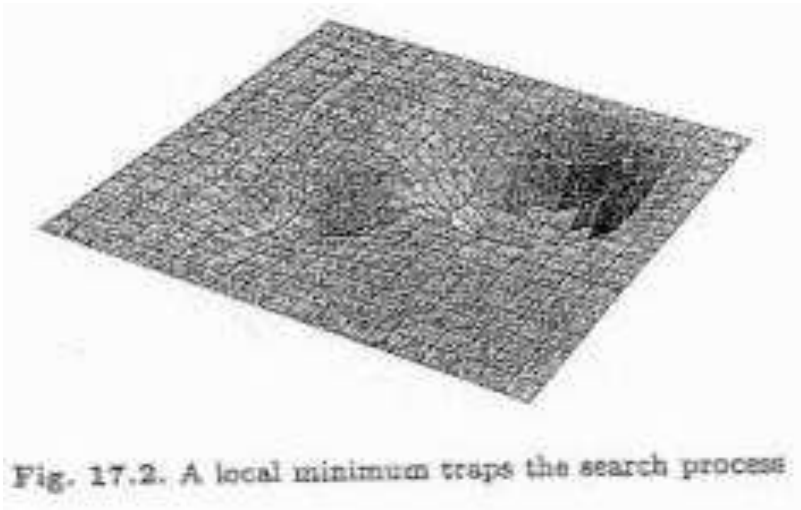


Fig. 17.2. A local minimum traps the search process

- The disadvantage of simple stochastic search is that a *local minimum* can steer the search in the wrong direction.

- One remedy is to carry out several independent searches.

The Metropolis Algorithm

- Annealing:

A metal is heated until it liquefies. It is then slowly cooled until it once again changes phase, i.e., solidifies.

- Simulated Annealing:

Variation on stochastic search.

If a new search direction (δ_1, ... , δ_n) decreases the function value, it is used to update the search position.

However, if the function value increases, this new direction is still used with probability p with,

$$p = \frac{1}{1 + \exp\left[\frac{1}{\alpha}\left(f(x_1 + \delta_1, \ldots, x_n + \delta_n) - f(x_1, \ldots, x_n)\right)\right]}$$

where the constant α (the "temperature") approaches zero gradually.

- At the beginning of the search process, counter-productive jumps are taken with a relatively high probability.
- In the final iterations, only productive jumps are taken.
- This methodology prevents getting trapped in a local minimum (or maximum.)

Bit-Based Descent Methods

- The problem is encoded (re-coded) so that the function f is calculated with the help of a binary string.
- Example:

  $x \rightarrow (1 - x^2)$.
- Let $x = b_7\, b_6\, b_5\, b_4\, b_3 b_2\, b_1\, b_0$

Where $b_7\, b_6\, b_5$ is the Whole part, and $b_4\, b_3 b_2\, b_1\, b_0$ is the Fractional part. Numbers between 0 and 8 can be represented this way. A sign bit may also be added.

- A randomly chosen initial string $b_7\, b_6\, b_5\, b_4\, b_3 b_2\, b_1\, b_0$ is generated. The function f is then computed for this point.
- A bit of the string is selected at random and flipped. For example:

  $x' = b'_7\, b_6\, b_5\, b_4\, b_3 b_2\, b_1\, b_0$

  Here the b_7 was flipped to b'_7.
- The function f is computed at x'

If f(x') < f(x) then x' is accepted and another iteration is performed.

- The algorithm runs until no bit flip improves the value of f.
- [Illegible ... cut off in photocopy...]

Genetic Algorithms

- Genetic algorithms are stochastic search methods managing a population of simultaneous search positions.
- GA's work with a coding of the optimization problem.

Example

- Consider the following puzzle. A robot starts at square S, and the goal square is G:

F		Goal G
E	D	C
Start S	A	B

The robot can move one square at a time in any of the following directions:

North
South
East
West

As long as a move is not prevented by constraints of the board (e.g., cannot move West from D or East from F.

- We wish to reach the Goal in four moves.
- We may encode our moves as:

Try: 00101000

Where

North = 00
South = 01
East = 10
West = 11
And hence, a series of four moves.

- GA's begin with a population of strings (search points) where population size is fixed at the beginning.

GA Operators

- ***Selection***

A string is chosen to help generate the next population based upon its fitness measure.

For example, the string 10101111 is not likely to participate in reproduction (why not?)

- ***Mutation Operator***

A bit may be complemented with some small probability, say 0.001 during reproduction.

- ***Information Exchange Operators***

eg. crossover.

Example: Consider two strings chosen for reproduction:
String 1: 10100001 , (lands robot in square B)
String 2: 00010000 , (lands robot in square F)

A crossover point is randomly chosen, say at k = 6 (i.e., after six bits from the left)
101000|01
000100|00

Then the Descendent point equals

10100000 , sends robot to Goal.

In the descendant string above, the first six bits are from string 1, and the last 2 are from string 2.

- ***Schema***

00 representation for all strings of length 6 with two zeros in the center positions. (e.g., 11<u>00</u>01, 00<u>00</u>10, ...)

- During the course of a GA, the best bit patterns are selected - "Schema Theorem".

Deceptive Functions

- However, some problems are difficult for GA's. These are termed *Deceptive Functions*.

This occurs when there is a correlation between optimal bits.

- e.g.,

$(x_1, x_2, \ldots, x_n) \rightarrow ((x_1^2, \ldots, x_n^2)/(x_1^2 + \varepsilon) + \ldots + (x_1^2, \ldots, x_n^2)/(x_n^2 + \varepsilon))$.

where ε is a small positive constant.

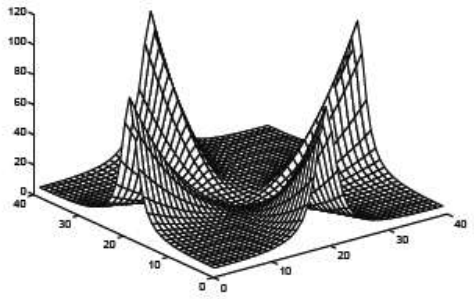


Fig. 17.4. A deceptive function

Need to approach from just the right direction.

Relatively easy problem for Gradient Descent.

- Other problems are easy. Like the so-called "*royal road*" functions.

- **Metagenetic Algorithms**

Encode the mutation rate ("tendency to explore" regions of the search space) or the length of stochastic changes to the points in the population in the individual bit strings themselves.

- *Optimal mutation rate* sought simultaneously with the optimum of the fitness function.

Neural Nets and Genetic Algorithms

- Is it possible to use GA's to find the weights in a network?
- Is it possible to let networks evolve so that they find an optimal topology?
- There is a problem with finding the weights

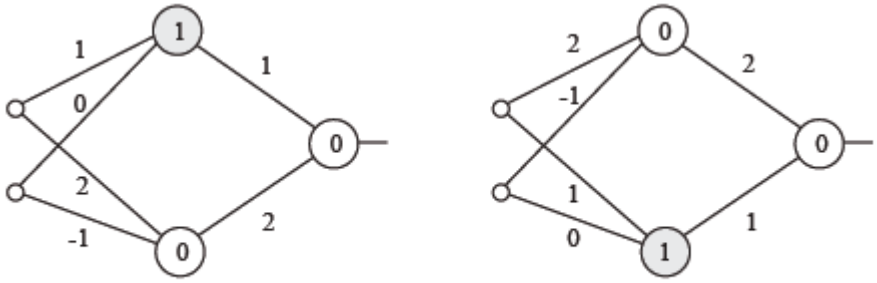


Fig. 17.6. Equivalent networks (weight permutation)

High number of symmetries for the error function...

permuting the weights...
coding for each would look very different.

Encoding/Decoding Problem with Eight Inputs

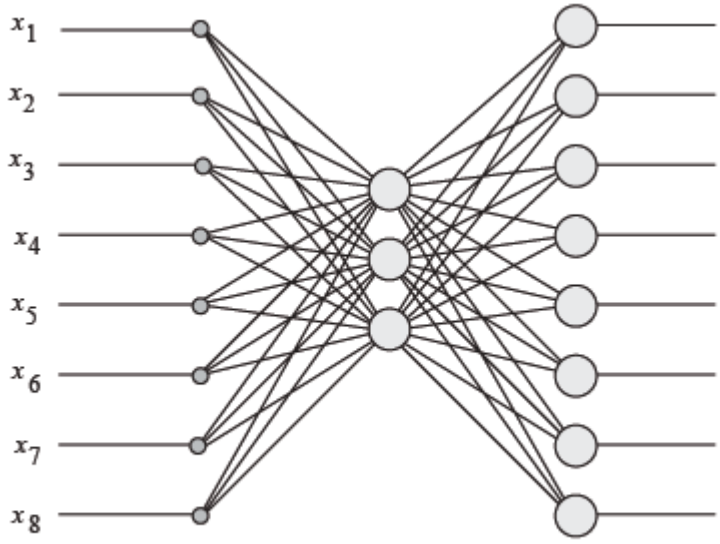


Fig. 17.8. An 8-bit encoder-decoder

- Only 1 input "on" - it must be replicated at output.
- We have,
 48 weights
+ 8 input bits
+ 3 bits to encode input at hidden units
 59 floating point numbers.
- Crossover through the middle of a parameter was avoided.
- Evolution of the Error curve:

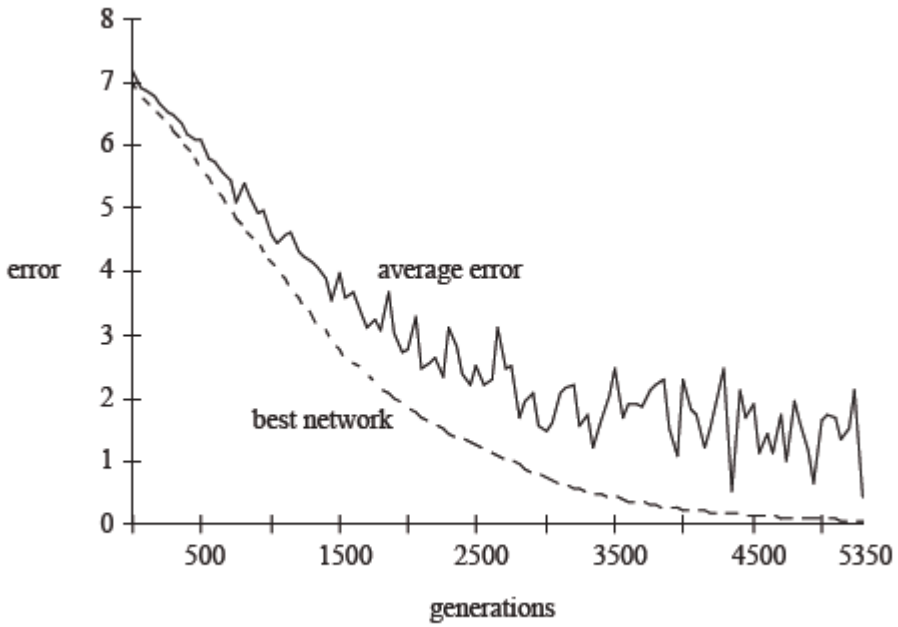


Fig. 17.9. Error function at each generation: population average and best network

After 5350 generations, error < 0.05.

- For larger networks, this is an interesting research problem.

Prisoner's Dilemma

- In this game between two players, each one decides independently whether they wish to cooperate with the other player (partner in crime) or betray him (squeal to the police.)

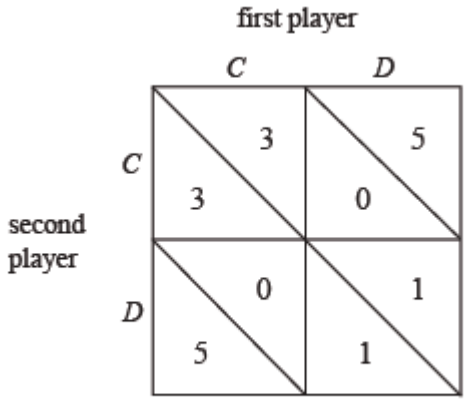


Fig. 17.10. Pay-off matrix for the prisoner's dilemma

C = Cooperate
B = Betray
Pay-off = 5, number of years in prison.

- If the first player betrays his partner, then the first is set free, the second has 5 years in jail.
- Incentive to commit treachery: If just played once, naturally one should betray!
- More complicated if game is repeated: Iterated Prisoner's Dilemma.
- If player only stores last three games; tit-for-tat (TFT) is a successful strategy (i.e., repeat last move of your opponent.)
- Axelrod and Hamilton, 1981.

Artificial Neural Networks
Lecture Notes

Chapter 11

Contents

Associative Memory Networks

- Remembering something: Associating an idea or thought with a sensory cue.

- Human memory connects items (ideas, sensations, &c.) that are similar, that are contrary, that occur in close proximity, or that occur in close succession
  - *Aristotle*

- An input stimulus which is similar to the stimulus for the association will invoke the associated response pattern.
  - A woman's perfume on an elevator...
  - A song on the radio...
  - An old photograph...

- An *Associative Memory Net* may serve as a highly simplified model of human memory.

- These associative memory units should not be confused with *Content Addressable Memory Units*.

A Taxonomy of Associative Memories

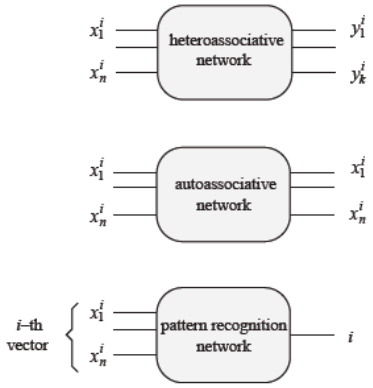


Fig. 12.1. Types of associative networks

The superscripts of x and y are all i

- *Heteroassociative network*

Maps n input vectors x^{-1}, x^{-2}, ..., x^{-n}, in n-dimensional space to m output vectors y^{-1}, y^{-2}, ..., y^{-m}, in m-dimensional space,

$x^{-i} \rightarrow y^{-i}$

if $||\tilde{x} - x^{-i}||^2 < \varepsilon$ then $\tilde{x} \rightarrow y^{-i}$

- *Autoassociative Network*

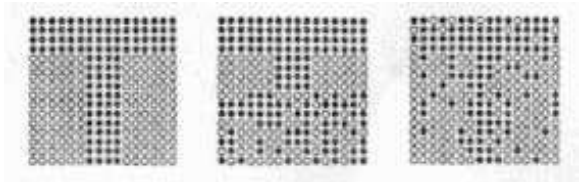
A type of heteroassociative network.
Each vector is associated with itself; i.e.,

$x^{-i} = y^{-i}$, i = 1, ..., n.

Features *correction of noisy input vectors*.

- *Pattern Recognition Network*

A type of heteroassociative network.
Each vector x^{-i} is associated with the scalar i.
[*illegible - remainder cut-off in photocopy*]



An Example of Associative Recall

To the left is a binarized version of the letter "T".

The middle picture is the same "T" but with the bottom half replaced by noise.

Pixels have been assigned a value 1 with probability 0.5
Upper half: The cue Bottom half: has to be recalled from memory.

The pattern on the right is obtained from the original "T" by adding 20% noise. Each pixel is inverted with probability 0.2.
The whole memory is available, but in an imperfectly recalled form ("hazy" or inaccurate memory of some scene.)

(Compare/contrast the following with database searches)
In each case, when part of the pattern of data is presented in the form of a sensory cue, the rest of the pattern (memory) is associated with it.

Alternatively, we may be offered an *imperfect* version of the...
[*illegible - remainder cut-off in photocopy*]

Hebbian Learning
Donald Hebb - psychologist, 1949.

Two neurons which are simultaneously active should develop a degree of interaction higher than those neurons whose activities are uncorrelated.

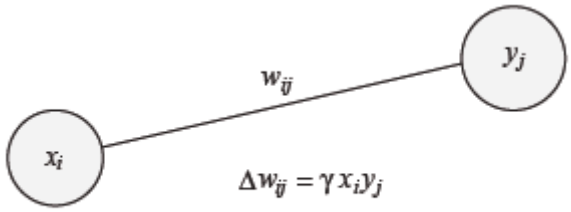


Fig. 12.4. The Hebb rule

Input x_i
Output y_j
Weight update $\Delta w_{ij} = \gamma\, x_i\, y_j$

Hebb Rule for Pattern Association

It can be used with patterns that are represented as either binary or bipolar vectors.

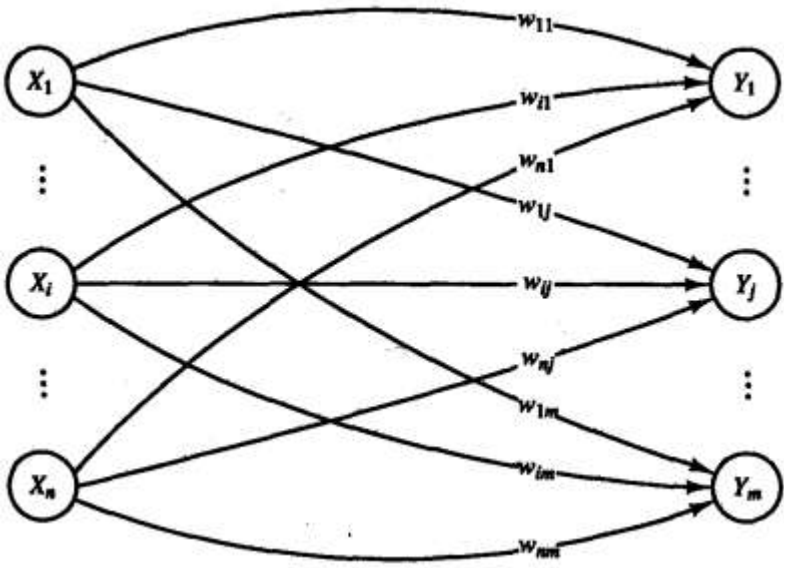
- Training Vector Pairs $s;^{\_} : t;^{\_}$
- Testing Input Vector $x;^{\_}$ (which may or may not be the same as one of the training input vectors.)

Algorithm

Step 0. Initialize all weights (i = 1, . . . , *n; j* = **1, . . .** , m):
w_{ij} =***0***

Step 1. For each input training-target output vector pair **s:** t, do Steps 2-4.
Step 2. Set activations for input units to current training input
(i = 1,. . . ,*n):*
$x_i = s_i$
***Step* 3.** Set activations for output units to current target output
(j = 1. . . , m):
$y_j = t_j$
Step 4. Adjust the weights (i = 1. . . , *n; j* = 1, . . . , *m**):***
w_{ij}***(new)*** = w_{ij}***(old)*** + $x_i y_j$

In this simple form of Hebbian Learning, one generally employs *outer product* calculations instead.



Architecture of a Heteroassociative Neural Net

Procedure

Step ***0.*** Initialize weights using either the Hebb rule (Section 3.1.1) or the delta rule (Section 3.1.2).
Step 1. For each input vector, do Steps 2-4.
Step 2. Set activations for input layer units equal to the current input vector
x_i
Step 3. Compute net input to the output units:

$y\_in_j = \sum_i x_i w_{ij}$

Step 4. Determine the activation of the output units:

$$y_j = \begin{cases} 1 & \text{if } y\_in_j > 0 \\ 0 & \text{if } y\_in_j = 0 \\ -1 & \text{if } y\_in_j < 0 \end{cases},$$

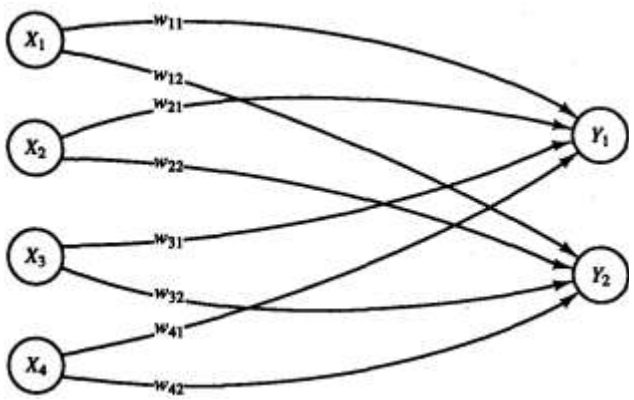
(for bipolar targets).

The output vector y gives the pattern associated with the input vector **x.** This heteroassociative memory is not iterative.

Other activation functions can also be used. If the target responses of the net are binary, a suitable activation function is given by

$$f(x) = \begin{cases} 1 & \text{if } x > 0; \\ 0 & \text{if } x \leq 0. \end{cases}$$

A simple example (from Fausett's text)



Heteroassociative network.

Input vectors - 4 components

Output vectors - 2 components

Example :A Heteroassociative net trained using the Hebb rule: algorithm
Suppose a net is to be trained to store the following mapping from input row vectors **S =*(s1,s2,s3,s4)*** , to output row vectors t = ***(t1***, ***t2):***

		s1	s2	s3	s4			t1	t2
1st	s	(1,	0,	0,	0)	1st	t	(1,	0)
2nd	s	(1,	1,	0,	0)	2nd	t	(1,	0)
3rd	s	(0,	0,	0,	1)	3rd	t	(0,	1)
4th	s	(0,	0,	1,	1)	4th	t	(0,	1)

The input vectors are not mutually orthogonal (i.e., the dot product ≠ 0,) - in which case the response will include a portion of each of their target values - *cross-talk*.

Note: target values are chosen to be related to the input vectors in a simple manner.
The *cross-talk* between the first and second input vectors does not pose any difficulties (since these target values are the same.
[Illegible - cut off in photocopy]

The training is accomplished by the Hebb rule:

$w_{ij}(new) = w_{ij}(old) + s_i t_j$
ie, $\Delta w_{ij} = s_i t_j$, $\alpha = 1$.

Step 0. Initialize all weights to 0.
Step 1. For the first **s:t** pair (1, 0, 0, 0):(1, 0):
Step 2. x1 = 1; x2 = x3 = x4 = 0.
Step 3. y1 = 1; y2 = 0.
Step 4. $w_{11}(new) = w_{11}(old) + x_1 y_1 = 0 + 1 = 1.$
(All other weights remain ***0.)***
Step 1. For the second s: **t** pair (1, 1, 0, 0):(1, 0):
Step 2. x1 = 1; x2 = 1; x3 = x4 = 0.
Step 3. y1 = 1; y2 = 0.
Step 4. $w_{11}(new) = w_{11}(old) + x_1 y_1 = 1 + 1 = 2;$
$w_{21}(new) = w_{21}(old) + x_2 y_1 = 0 + 1 = 1.$
(All other weights remain **0.)**
Step 1. For the third **s:t** pair (0, 0, 0, 1):(0, 1):
Step 2. x1 = xz = x3 = 0; x4 = 1.
Step 3. y1=O; y2=1.
Step 4. $w_{42}(new) = w_{42}(old) + x_4 y_2 = 0 + 1 = 1.$
(All other weights remain unchanged.)
Step 1. For the fourth **s:t** pair (0, 0, 1, 1):(0, 1):
Step 2. x1=x2=O; x3=1; x4=1.
step 3. y1=O: y2=1.
Step 4. $w_{32}(new) = w_{32}(old) + x_3 y_2 = 0 + 1 = 1;$
$w_{42}(new) = w_{42}(old) + x_4 y_2 = 1 + 1 = 2.$
(All other weights remain unchanged.)

The weight matrix is

$$\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}.$$

Example 3.3 Testing a heteroassociative net using the training input
We now test the ability of the net to produce the correct output for each of the training inputs. The steps are as given in the application procedure at the beginning of this section, using the activation function

$$f(x) = \begin{cases} 1 & \text{if } x > 0; \\ 0 & \text{if } x \leq 0. \end{cases}$$

The weights are as found in Examples 3.1 and 3.2.

step 0. $\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}$.

Step 1. For the first input pattern, do Steps 2–4.

Step 2. $\mathbf{x} = (1, 0, 0, 0)$.

Step 3. $y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$

$= 1(2) + 0(1) + 0(0) + 0(0)$

$= 2;$

$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$

$= 1(0) + 0(0) + 0(1) + 0(2)$

$= 0.$

Step 4. $y_1 = f(y\_in_1) = f(2) = 1;$

$y_2 = f(y\_in_2) = f(0) = 0.$

(This is the correct response for the first training pattern.)

Step 1. For the second input pattern, do Steps 2–4.

Step 2. $\mathbf{x} = (1, 1, 0, 0)$.

Step 3. $y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$

$= 1(2) + 1(1) + 0(0) + 0(0)$

$= 3;$

$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$

$= 1(0) + 1(0) + 0(1) + 0(2)$

$= 0.$

Step 4. $y_1 = f(y\_in_1) = f(3) = 1;$

$y_2 = f(y\_in_2) = f(0) = 0.$

(This is the correct response for the second training pattern.)

testing - cont'd:

Step 1. For the third input pattern, do Steps 2–4.

Step 2. x = (0, 0, 0, 1).

Step 3. $y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$

$= 0(2) + 0(1) + 0(0) + 1(0)$

$= 0;$

$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$

$= 0(0) + 0(0) + 0(1) + 1(2)$

$= 2.$

Step 4. $y_1 = f(y\_in_1) = f(0) = 0;$

$y_2 = f(y\_in_2) = f(2) = 1.$

(This is the correct response for the third training pattern.)

Step 1. For the fourth input pattern, do Steps 2–4.

Step 2. x = (0, 0, 1, 1).

Step 3. $y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41}$

$= 0(2) + 0(1) + 1(0) + 1(0)$

$= 0;$

$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42}$

$= 0(0) + 0(0) + 1(1) + 1(2)$

$= 3.$

Step 4. $y_1 = f(y\_in_1) = f(0) = 0;$

$y_2 = f(y\_in_2) = f(2) = 1.$

(This is the correct response for the fourth training pattern.)

We can employ vector-matrix rotation to illustrate the testing process.

We repeat the steps of the application procedure for the input vector **x,** which is the first of the training input vectors **s**

Step 0. $\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}$.

Step 1. For the input vector:

Step 2. $\mathbf{x} = (1, 0, 0, 0)$.

Step 3. $\mathbf{x}\,\mathbf{W} = (y\_in_1, y\_in_2)$

$$(1, 0, 0, 0) \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix} = (2, 0).$$

Step 4. $\mathbf{f}(2) = 1$; $\mathbf{f}(0) = 0$;

$\mathbf{y} = (1, 0)$.

The entire process (Steps 2–4) can be represented by

$$\mathbf{x}\,\mathbf{W} = (y\_in_1, y\_in_2) \rightarrow \mathrm{y}$$

$$(1, 0, 0, 0) \begin{bmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix} = (2, 0) \qquad \rightarrow (1, 0),$$

or, in slightly more compact notation,

$$(1, 0, 0, 0)\cdot\mathbf{W} = (2, 0) \rightarrow (1, 0).$$

Note that the output activation vector is the same as the training output vector that was stored in the weight matrix for this input vector.

Similarly, applying the same algorithm, with x equal to each of the other three training input vectors, yields

$$(1, 1, 0, 0)\cdot\mathbf{W} = (3, 0) \rightarrow (1, 0),$$

$$(0, 0, 0, 1)\cdot\mathbf{W} = (0, 2) \rightarrow (0, 1),$$

$$(0, 0, 1, 1)\cdot\mathbf{W} = (0, 3) \rightarrow (0, 1).$$

Note that the net has responded correctly to (has produced the desired vector of output activations for) each of the training patterns.

Example 3.4 Testing a heteroassociative net with input similar to the training input

The test vector $\mathbf{x} = (0, 1, 0, 0)$ differs from the training vector $\mathbf{s} = (1, 1, 0, 0)$ only in the first component. We have

$$(0, 1, 0, 0)\cdot\mathbf{W} = (1, 0) \rightarrow (1, 0).$$

Thus, the net also associates a known output pattern with this input.

Example 3.5 Testing a heteroassociative net with input that is not similar to the training input

The test pattern (0 1, 1, 0) differs from each of the training input patterns in at least two components. We have

$$(0\ 1,\ 1,\ 0)\cdot\mathbf{W} = (1,\ 1) \rightarrow (1,\ 1).$$

The output is not one of the outputs with which the net was trained; in other words, the net does not recognize the pattern. In this case, we can view $\mathbf{x} = (0, 1, 1, 0)$ as differing from the training vector $s = (1, 1, 0, 0)$ in the first and third components, so that the two "mistakes" in the input pattern make it impossible for the net to recognize it. This is not surprising, since the vector could equally well be viewed as formed from $\mathbf{s} = (0, 0, 1, 1)$, with "mistakes" in the second and fourth components.

- A bipolar representation would be preferable. More robust in the presence of noise.
- The weight matrix obtained from the previous examples would be:

$$w = \begin{bmatrix} 4 & -4 \\ 2 & -2 \\ -2 & 2 \\ -4 & 4 \end{bmatrix}$$

with two "mistakes".
Trouble remains:

ie, (-1, 1, 1, -1) . w = (0,0) → (0,0).

- However, the net can respond correctly when given an input vector with two components missing.

e.g., X = (0, 1, 0, -1) formed from S = (1, 1,-1, -1) with the first and third components missing rather than wrong.
(0, 1, 0, -1). w = (6,-6) = (1,1) which is the correct response for the stored vector s = (1, 1, - 1, - 1). These "missing" components are really just a particular form of noise that produces an input vector which is not as dissimilar to a training vector as is the input vector produced with the more extreme "noise" denoted by the term "mistake."

Character Recognition Example

(Example 3.9) A heteroassociative net for associating letters from different fonts

A heteroassociative neural net was trained using the Hebb rule (outer products) to associate three vector pairs. The x vectors have 63 components, the y vectors 15.

The vectors represent patterns.

The pattern is converted to a vector representation that is suitable for processing as follows: The #'s are replaced by 1's and the dots by -1's, reading across each row (starting with the top row). The pattern shown becomes the vector (-1,1,-1 1,-1,1 1,1,1 1,-1,1 1,-1,1).

The extra spaces between the vector components, which separate the different rows of the original pattern for ease of reading, are not necessary for the network.

The figure below shows the vector pairs in their original two-dimensional form.

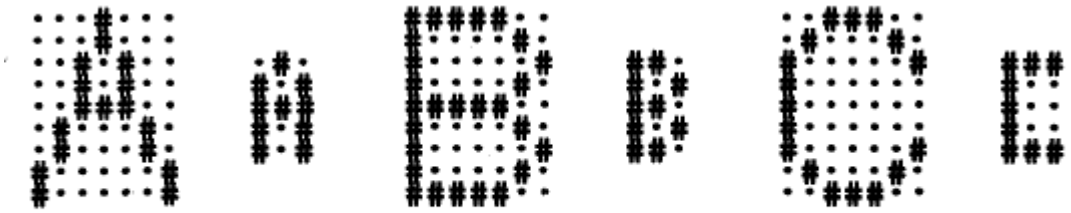


Figure 3.3 Training patterns for character recognition using heteroassociative net.

After training, the net was used with input patterns that were noisy versions of the training input patterns. The results are shown in figures 3.4 and 3.5 (below). The noise took the form of turning pixels "on" that should have been "off" and vice versa.

These are denoted as follows:
@ Pixel is now "on", but this is a mistake (noise).
O Pixel is now "off", but this is a mistake (noise).

Figure 3.5 (below) shows that the neural net can recognize the small letters that are stored in it, even when given input patterns representing the large training patterns with 30% noise.

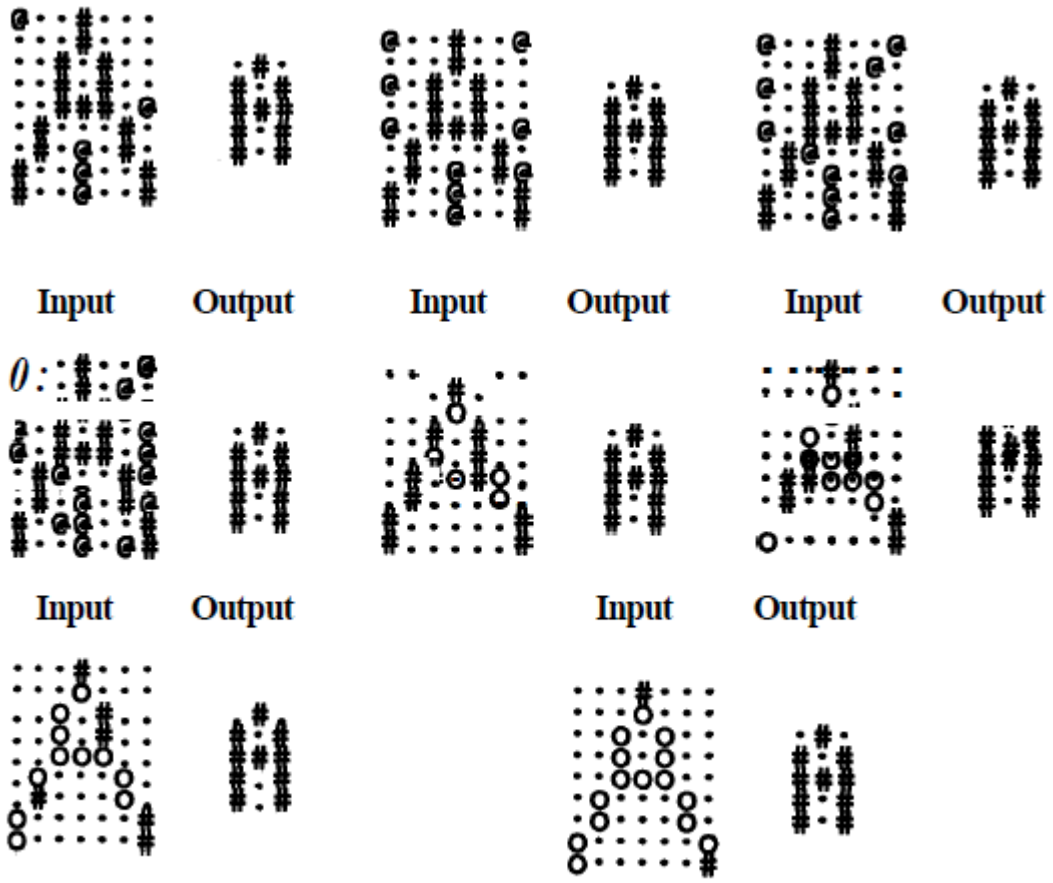


Figure 3.4 Response of heteroassociative net to several noisy versions of pattern A.

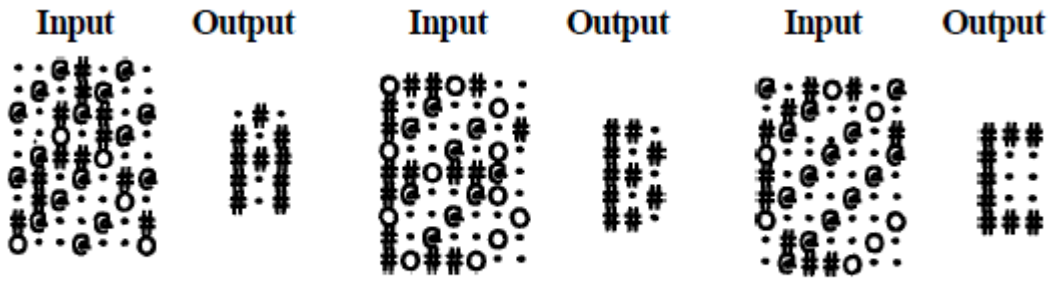
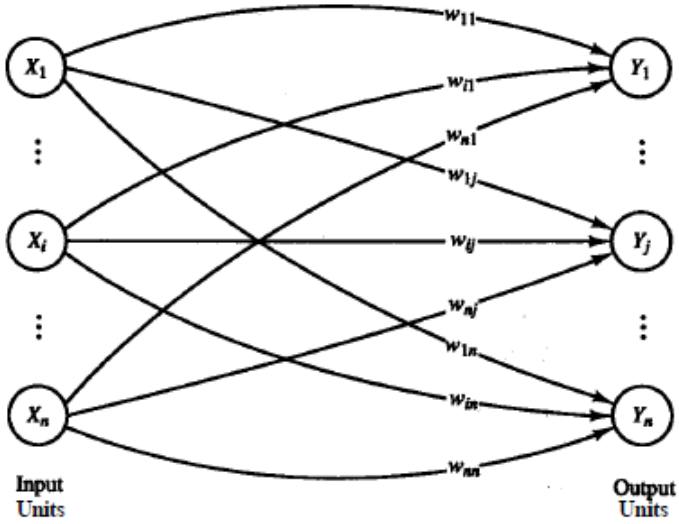


Figure 3.5 Response of heteroassociative net to patterns A, B, and C with mistakes in 1/3 of the components.

Autoassociative Nets

- For an autoassociative net, the training input and target output vectors are identical.

- The process of training is often called *storing* the vectors, which may be binary or bipolar.
- A *stored vector* can be *retrieved* from distorted or partial (noisy) input if the input is sufficiently close to it.
- The performance of the net is judged by its ability to reproduce a stored pattern from noisy input; performance is generally better for bipolar vectors than for binary vectors.



Architecture of an Autoassociative neural net

It is common for weights on the diagonal (those which connect an input pattern component to the corresponding component in the output pattern) to be set to zero.

3.3.2 Algorithm

For mutually orthogonal vectors, the Hebb rule can be used for setting the weights in an autoassociative net because the input and output vectors are perfectly correlated, component by component (i.e., they are the same). The algorithm is as given in Section 3.1.1; note that there are the same number of output units as input units.

Step 0. Initialize all weights, $i = 1, \ldots, n$; $j = 1, \ldots, n$:

$$w_{ij} = 0;$$

Step 1. For each vector to be stored, do Steps **2–4**:

Step 2. Set activation for each input unit, $i = 1, \ldots, n$:

$$x_i = s_i.$$

Step 3. Set activation for each output unit, $j = 1, \ldots, n$:

$$yj = sj;$$

Step 4. Adjust the weights, $i = 1, \ldots, n$; $j = 1, \ldots, n$:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j.$$

As discussed earlier, in practice the weights are usually set from the formula

$$\mathbf{W} = \sum_{p=1}^{P} \mathbf{s}^{\mathrm{T}}(p)\mathbf{s}(p),$$

rather than from the algorithmic form of Hebb learning.

Application and examples of Autoassociative Nets

3.3.3 Application

An autoassociative neural net can be used to determine whether an input vector is "known" (i.e., stored in the net) or "unknown." The net recognizes a "known" vector by producing a pattern of activation on the output units of the net that is the same as one of the vectors stored in it. The application procedure (with bipolar inputs and activations) is as follows:

Step 0. Set the weights (using Hebb rule or outer product).

Step 1. For each testing input vector, do Steps **2–4.**

Step 2. Set activations of the input units equal to the input vector.

Step 3. Compute net input to each output unit, $j = 1, \ldots, n$:

$$y\_in_j = \sum_i x_i w_{ij}.$$

Step 4. Apply activation function ($j = 1, \ldots, n$):

$$y_j = f(y\_in_j) = \begin{cases} 1 & \text{if } y\_in_j > 0; \\ -1 & \text{if } y\_in_j \leq 0. \end{cases}$$

Simple examples

Example 3.10 An autoassociative net to store one vector: recognizing the stored vector

We illustrate the process of storing one pattern in an autoassociative net and then recalling, or recognizing, that stored pattern.

Step 0. The vector $s = (1, 1, 1, -1)$ is stored with the weight matrix:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}.$$

Step 1. For the testing input vector:

Step 2. $x = (1, 1, 1, -1)$.

Step 3. $y\_in = (4, 4, 4, -4)$.

Step 4. $y = f(4, 4, 4, -4) = (1, 1, 1, -1)$.

Since the response vector y is the same as the stored vector, we can say the input vector is recognized as a "known" vector.

The preceding process of using the net can be written more succinctly as

$$(1, 1, 1, -1)\cdot\mathbf{W} = (4, 4, 4, -4) \rightarrow (1, 1, 1, -1).$$

Now, if recognizing the vector that was stored were all that this weight matrix enabled the net to do, it would be no better than using the identity matrix for the weights. However, an autoassociative neural net can recognize as "known" vectors that are similar to the stored vector, but that differ slightly from it. As before, the differences take one of two forms: "mistakes" in the data or "missing" data. The only "mistakes" we consider are changes from $+1$ to -1 or vice versa. We use the term "missing" data to refer to a component that has the value 0, rather than either $+1$ or -1.

Example 3.11 Testing an autoassociative net: one mistake in the input vector

Using the succinct notation just introduced, consider the performance of the net for each of the input vectors x that follow. Each vector x is formed from the original stored vector s with a mistake in one component.

$$(-1, 1, 1, -1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, -1, 1, -1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 1, -1, -1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 1, 1, 1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1).$$

Note that in each case the input vector is recognized as "known" after a single update of the activation vector in Step 4 of the algorithm. The reader can verify that the net also recognizes the vectors formed when one component is "missing." Those vectors are (0, 1, 1, -1), (1, 0, 1, −1), (1, 1, 0, −1), and (1, 1, 1, 0).

In general, a net is more tolerant of "missing" data than it is of "mistakes" in the data, as the examples that follow demonstrate. This is not surprising, since the vectors with "missing" data are closer (both intuitively and in a mathematical sense) to the training patterns than are the vectors with "mistakes."

Example 3.12 Testing an autoassociative net: two "missing" entries in the input vector

The vectors formed from (1, 1, 1, −1) with two "missing" data are (0, 0, 1, −1), (0, 1, 0, -1), (0, 1, 1, 0), (1, 0, 0, -1), (1, 0, 1, 0), and (1, 1, 0, 0). As before, consider the performance of the net for each of these input vectors:

$$(0, 0, 1, -1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(0, 1, 0, -1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(0, 1, 1, 0)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 0, 0, -1)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 0, 1, 0)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 1, 0, 0)\cdot\mathbf{W} = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1).$$

The response of the net indicates that it recognizes each of these input vectors as the training vector (1, 1, 1, − 1), which is what one would expect, or at least hope for.

Example 3.13 Testing an autoassociative net: two mistakes in the input vector

The vector (− 1, − 1, 1, − 1) can be viewed as being formed from the stored vector (1, 1, 1, − 1) with two mistakes (in the first and second components). We have:

$$(-1, -1, 1, -1).\mathrm{W} = (0,0,0,0).$$

The net does not recognize this input vector.

Example 3.14 An autoassociative net with no self-connections: zeroing-out the diagonal

It is fairly common for an autoassociative network to have its diagonal terms set to zero, e.g.,

$$\mathbf{W}_0 = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Consider again the input vector (− 1, − 1, 1, − 1) formed from the stored vector (1, 1, 1, − 1) with two mistakes (in the first and second components). We have:

$$(-1, -1, 1, -1)\cdot\mathbf{W}_0 = (-1, 1, -1, 1).$$

The net still does not recognize this input vector.

It is interesting to note that if the weight matrix $\mathbf{W}_0$ (with 0's on the diagonal) is used in the case of "missing" components in the input data (see Example 3.12), the output unit or units with the net input of largest magnitude coincide with the input unit or units whose input component or components were zero. We have:

$$(0, 0, 1, -1)\cdot\mathbf{W}_0 = (2, 2, 1, -1) \rightarrow (1, 1, 1, -1)$$

$$(0, 1, 0, -1)\cdot\mathbf{W}_0 = (2, 1, 2, -1) \rightarrow (1, 1, 1, -1)$$

$$(0, 1, 1, 0)\cdot\mathbf{W}_0 = (2, 1, 1, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 0, 0, -1)\cdot\mathbf{W}_0 = (1, 2, 2, -1) \rightarrow (1, 1, 1, -1)$$

$$(1, 0, 1, 0)\cdot\mathbf{W}_0 = (1, 2, 1, -2) \rightarrow (1, 1, 1, -1)$$

$$(1, 1, 0, 0)\cdot\mathbf{W}_0 = (1, 1, 2, -2) \rightarrow (1, 1, 1, -1).$$

The net recognizes each of these input vectors.

Storage Capacity

An important consideration for associative memory neural networks is the number of patterns or pattern pairs that can be stored before the net begins to forget. In this section we consider some simple examples and theorems for noniterative autoassociative nets.

Examples

Example 3.15 Storing two vectors in an autoassociative net

More than one vector can be stored in an autoassociative net by adding the weight matrices for each vector together. For example, if **W1** is the weight matrix used to store the vector (1, 1, - 1, - 1) and **W2** is the weight matrix used to store the vector (- 1, 1, 1, - 1), then the weight matrix used to store both (1, 1, - 1, - 1) and (- 1, 1, 1, - 1) is the sum of **W1** and **W2** . Because it is desired that the net respond with one of the stored vectors when it is presented with an input vector that is similar (but not identical) to a stored vector, it is customary to set the diagonal terms in the weight matrices to zero. If this is not done, the diagonal terms (which would each be equal to the number of vectors stored in the net) would dominate, and the net would tend to reproduce the input vector rather than a stored vector. The addition of **W1** and **W1**proceeds as follows:

$$\underset{\mathbf{W}_1}{\begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}} + \underset{\mathbf{W}_2}{\begin{bmatrix} 0 & -1 & -1 & 1 \\ -1 & 0 & 1 & -1 \\ -1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 0 \end{bmatrix}} = \underset{\mathbf{W}_1 + \mathbf{W}_2}{\begin{bmatrix} 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -2 \\ -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \end{bmatrix}}.$$

The reader should verify that the net with weight matrix **W1**+ **W2** can recognize both of the vectors (1, 1, - 1, - 1) and (-1, 1, 1, - 1). The number of vectors that can be stored in a net is called the ***capacity*** of the net.

Example 3.16 Attempting to store two nonorthogonal vectors in an autoassociative net

Not every pair of bipolar vectors can be stored in an autoassociative net with four nodes; attempting to store the vectors **(1, - 1, - 1, 1)** and **(1, 1, - 1, 1)** by adding their weight matrices gives a net that cannot distinguish between the two vectors it was trained to recognize:

$$\begin{bmatrix} 0 & -1 & -1 & 1 \\ -1 & 0 & 1 & -1 \\ -1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & -1 & 1 \\ 1 & 0 & -1 & 1 \\ -1 & -1 & 0 & -1 \\ 1 & 1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & -2 \\ 2 & 0 & -2 & 0 \end{bmatrix}.$$

The difference between Example **3.15** and this example is that there the vectors are orthogonal, while here they are not. Recall that two vectors **x** and **y** are orthogonal **if**

$$\mathbf{x}\,\mathbf{y}^{\mathrm{T}} = \sum_i x_i y_i = 0.$$

Informally, this example illustrates the difficulty that results from trying to store vectors that are too similar.

An autoassociative net with four nodes can store three orthogonal vectors (i.e., each vector is orthogonal to each of the other two vectors). However, the weight matrix for four mutually orthogonal vectors is always singular (so four vectors cannot be stored in an autoassociative net with four nodes, even if the vectors are orthogonal). These properties are illustrated in Examples 3.17 and 3.18.

Example 3.17 Storing three mutually orthogonal vectors in an autoassociative net

Let **W1** + **W2** be the weight matrix to store the orthogonal vectors (1, 1, - **1, - 1)** and (- **1, 1, 1, - 1)** and **W3** be the weight matrix that stores (- **1, 1, - 1, 1).** Then the weight matrix to store all three orthogonal vectors is **W1** + **W2** + **W3.** We have

$$\underset{\mathbf{W}_1 + \mathbf{W}_2}{\begin{bmatrix} 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -2 \\ -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \end{bmatrix}} + \underset{\mathbf{W}_3}{\begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}} = \underset{\mathbf{W}_1 + \mathbf{W}_2 + \mathbf{W}_3}{\begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}},$$

which correctly classifies each of the three vectors on which it was trained.

Example 3.18 Attempting to store four vectors in an autoassociative net

Attempting to store a fourth vector, **(1, 1, 1, 1),** with weight matrix **W4,** orthogonal to each of the foregoing three, demonstrates the difficulties encountered in over training a net, namely, previous learning is erased. Adding the weight matrix for the new vector to the matrix for the first three vectors gives

$$\underset{\mathbf{W_1}+\mathbf{W_2}+\mathbf{W_3}}{\begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}} + \underset{\mathbf{W_4}}{\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}} = \underset{\mathbf{W^*}}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}},$$

which cannot recognize any vector.

3.4 ITERATIVE AUTOASSOCIATNE NET

We see from the next example that in some cases the net does not respond immediately to an input signal with a stored target pattern, but the response may be enough like a stored pattern (at least in the sense of having more nodes committed to values of +**1** or - **1** and fewer nodes with the "unsure" response of 0) to suggest using this first response as input to the net again.

Example 3.19 Testing a recurrent autoassociative net: stored vector with second, third and fourth components set to zero

The weight matrix to store the vector (1, 1, 1, - 1) is

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

The vector (1,0,0,0) is an example of a vector formed from the stored vector with three "missing" components (three zero entries). The performance of the net for this vector is given next.

Input vector (1, 0, 0, 0):

(1, *0, 0,* 0).W = (0, 1, 1, - 1) → iterate

(0*,* 1, 1, -1).W= (3,2,2, 1, 1, - 1) → (1, 1, 1, - 1).

Thus, for the input vector (1, 0, 0, 0), the net produces the "known" vector (1, 1, 1, - 1) as its response in two iterations.

We can also take this iterative feedback scheme a step further and simply let the input and output units be the same, to obtain a recurrent autoassociative neural net. In Sections **3.4.1-3.4.3,** we consider three that differ primarily in their activation function. Then, in Section **3.4.4,** we examine a net developed by Nobel prize-winning physicist John Hopfield **(1982, 1988).** Hopfield's work (and his prestige) enhanced greatly the respectability of neural nets as a field of study in the 1980s. The differences between his net and the others in this section, although fairly slight, have a significant impact on the performance of the net. For iterative nets, one key question is whether the activations will converge. The weights are fixed (by the Hebb rule for example), but the activations of the units change.

Artificial Neural Networks
Lecture Notes

Chapter 12

Hopfield Networks

Hopfield Network (Discrete) – A recurrent autoassociative network.

Recurrent autoassociative network:

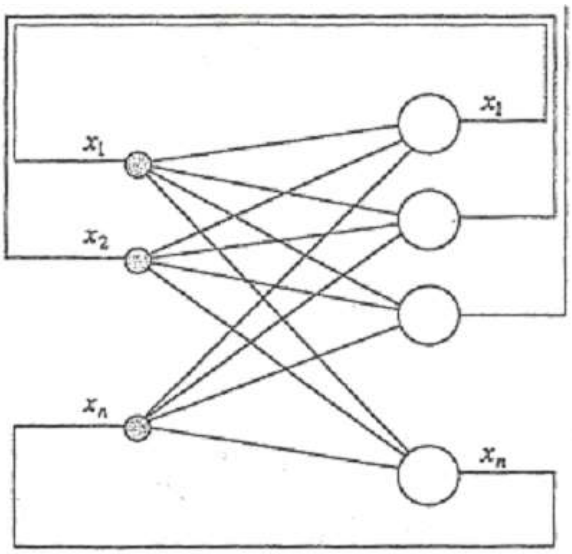


Fig. 12.3. Autoassociative network with feedback

Contrast with recurrent autoassociative network shown above

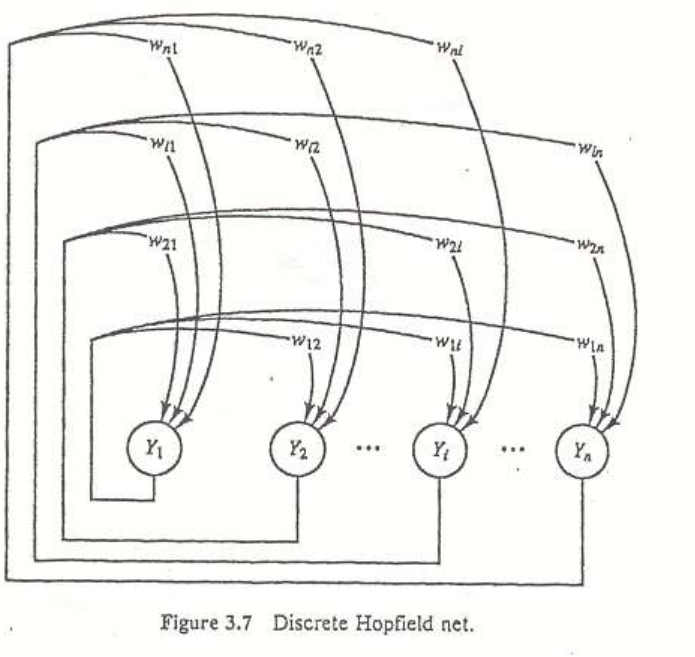


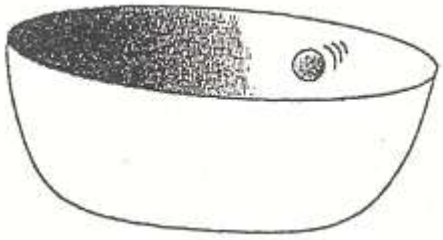
Figure 3.7 Discrete Hopfield net.

Note: There are no self-loops in a Hopfield net.

Hopfield Nets – Example of a dynamical physical system that may be thought of as
instantiating “memories” as stable states associated with minima of a suitably defined
energy.

A Physical Analogy with Memory

- Consider a bowl in which a ball bearing is allowed to roll freely…



- Suppose we let the ball go from a point somewhere up the side of the bowl.

- The ball will roll back and forth and around the bowl until it comes to rest at the bottom.

- The physical description of what has happened may be couched in terms of the energy of the system.

- The energy of the system is just the potential energy of the ball and is directly related to the height of the ball above the bowl's center; the higher the ball the greater its energy.

- Eventually the ball comes to rest at the bottom of [cut off in the note]

- The boll-bowl system settles in an energy minimum at equilibrium when it is allowed to operate under its own dynamics.

- The resting state is said to be stable because the system remains there after it has been reached.

An Alternate Paradigm

- We suppose that the ball comes to rest in the same place each time because it "remembers" where the bottom of the bowl is.

- Position vector

$$\bar{x}(t) = (x(t), y(t), z(t))$$

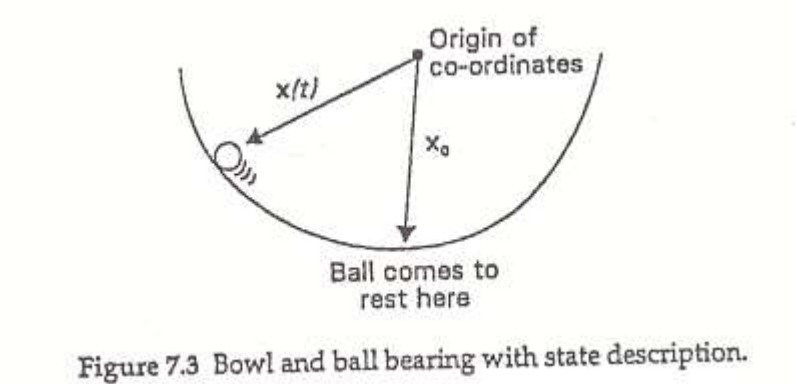
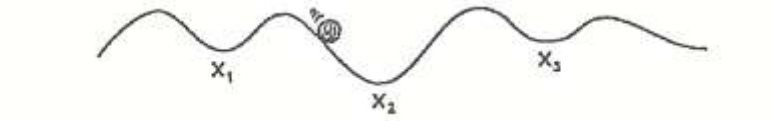


Figure 7.3 Bowl and ball bearing with state description.

- The location of the bottom of the bowl, x_p, represents the pattern that is stored. Where, ball's initial position.

- We may think of the ball's initial position as representing the partial knowledge or cue for recall [cut off from notes]

- If a corrugated surface is employed instead of a single depression (as in the bowl), we may store many "memories".



- If the ball is started somewhere on this surface, it will eventually come to rest at the local depression that is closest to its initial starting point. i.e. It evokes the stored pattern which is closest to its initial partial pattern or cue. This corresponds to an energy minimum of the system.

- The memories shown correspond to states x_1, x_2, x_3.

- Notice that the energy of each of the equilibria Ei may differ, but each one is the lowest available locally within its basin of attraction.

John Hopfield (1982) – American physicist proposed an asynchronous neural network model.

Hopfield Model

- Individual units preserve their own states until they are selected for an update.

- The selection (for updating) is made randomly.

- n totally coupled units – each unit is connected to all other units except itself.

- The network is symmetric $w_{ij} = w_{ji}$ for all i,j

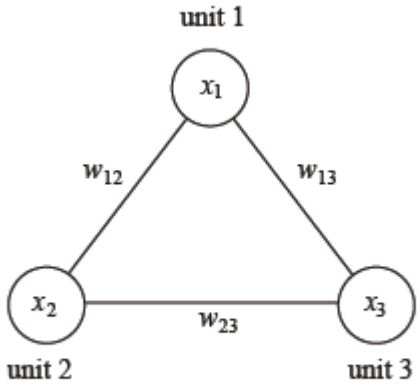


Fig. 13.2. A Hopfield network of three units

- Each unit can assume the state 1 or -1.

- If the weight matrix does not contain a zero diagonal, the network dynamics do not necessarily lead to stable states.

- For Example: (-1 0 0) transforms the state vector (1, 1, 1) [cut off from notes]

- A connection matrix with a zero diagonal can also lead to oscillations in the case where the weight matrix is not symmetric.

- The weight matrix

$$W = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

Describes this network:

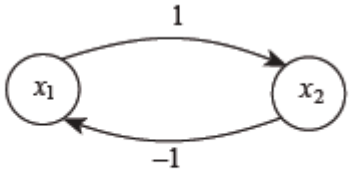


Fig. 13.3. Network with asymmetric connections

- The state vector (1, -1) is transformed into (1, 1) (-1, 1) → (-1, -1) → (1, -1)
- The symmetry of the weight matrix and a zero diagonal are thus necessary conditions for the convergence of an asynchronous totally connected network to a stable states. (…sufficient as well).

The Energy Function

Definition: Let W denote the weight matrix of a Hopfield network of n units and let be the n-dimensional row vector of units“ thresholds. The energy E(x) of a state x of the network is given by

$$E(x) = -\tfrac{1}{2}\, xWx^T + \theta x^T$$

The energy function can also be written in the form

$$E(x) = -\tfrac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} w_{ij} x_i x_i + \sum_{i=1}^{n} \theta_i x_i$$

The factor ½ is used because the identical terms $w_{ij}x_ix_j$ and $w_{ji}x_jx_i$ are present in the double sum.

The energy function of a Hopfield network is a quadratic form. A Hopfield network always finds a local minimum of the energy function.

An Example

Two units with threshold 0

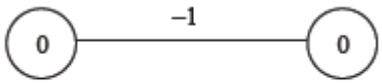


Fig. 13.4. A flip-flop

The only stable states are (1, -1) and (-1, 1)

$w_{12} = w_{21} = -1$

$E(x_1, x_2) = x_1 x_2$

Energy Function for Continuous Hopfield Model

- Units states can assume all real values between 0 and 1.
- The energy function has local minima at (1, -1) and (-1, 1)

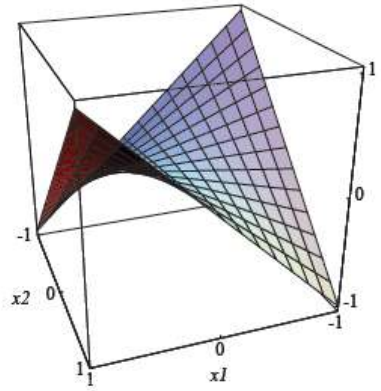


Fig. 13.5. Energy function of a flip-flop

Hopfield Network for Computing Logic Functions

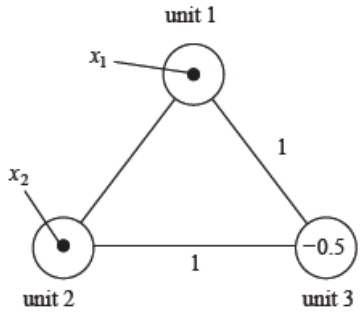


Fig. 13.6. Network for the computation of the OR function

- Note: The individual units are perceptorns…hence we suspect XOR cannot be implemented with 3 units.
- Suppose a Hopfield network with three units should store stable states given by:

unit	1	2	3
state 1	−1	−1	−1
state 2	1	−1	1
state 3	−1	1	1
state 4	1	1	−1

- From point of view of third unit, this is XOR function.
- The third unit should be capable of linearly separating the vectors (-1, -1) and (1,1) from (-1, 1) and (1, -1) but this is impossible.
- XOR problem can be solved if four units employed. The unknown weights can be found using a learning algorithm (Hebbian learning or a variation on perceptron learning).

unit	1	2	3	4
state 1	−1	−1	−1	1
state 2	1	−1	1	1
state 3	−1	1	1	1
state 4	1	1	−1	−1

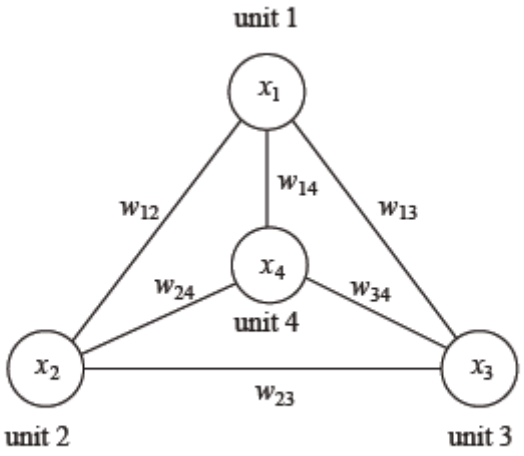


Fig. 13.7. Network for the computation of XOR

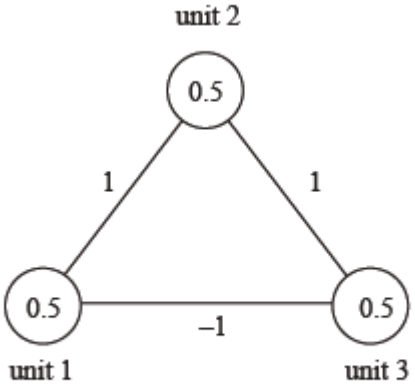
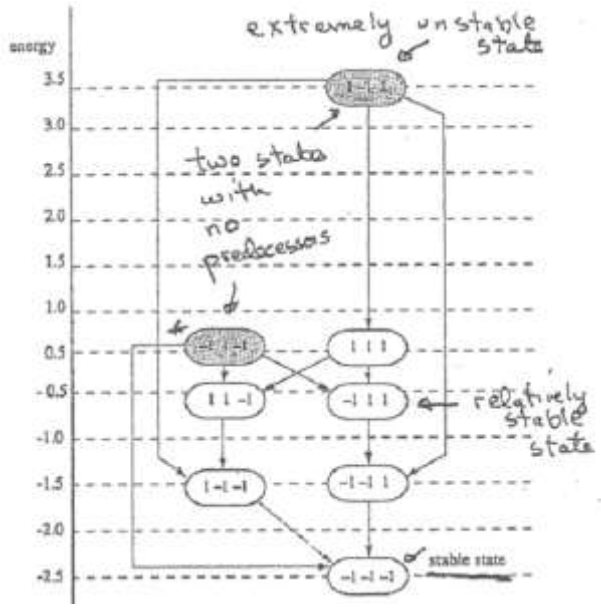


Fig. 13.9. Example of a Hopfield network

- Arbitrarily chosen weights and thresholds.
- This network can adopt any of eight possible states.



- Each transition has the same probability – the probability of selecting one of the three units for a stated transition is uniform and equal to energy calculation

$$E(x) = -1/2 \sum_{j=1}^{3} \sum_{i=1}^{3} w_{ij} x_i x_j + \theta_i x_i + \sum_{i=1}^{3} \theta_i x_i$$

- For state (111)

-1/2 (w12x1x2 + w13x1x3 + w21x2x1 + w23x2x3 + w32x3x2 + w31x3x1) + θ1x1 + θ2x2 + θ3x3
= -1/2 (1 + (-1) + 1 + 1 + 1 + (-1)) + 3/2

- Hopfield networks can be used to find approximate solution for difficult problems.
- Hopfield networks do not require synchronization; they guarantee that a local minimum of the energy function will be reached.
- If an optimization function can be written in an analytical form isomorphic to the Hopfield energy function – it can be solved by a Hopfield network.
- Every unit in the network is simulated by a small processor.
- The states of the units can be computed asynchronously by transmitting the current unit states from processor to processor.

The Multiflop Problem

- Binary vector of dimension n whose components are all zero except for a single 1.

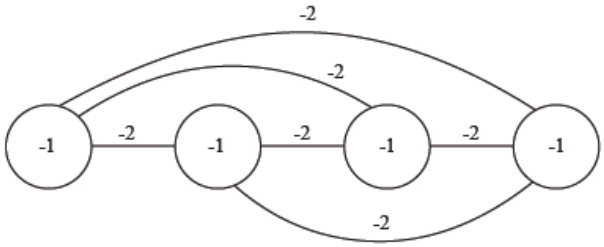


Fig. 13.15. A multiflop network

Hopfield Network to Solve This Problem

- Whenever a unit is set to 1, it inhibits the other units through the edges with weight-2.
- If network is started with all units set to zero then the excitation of every unit is 0, which is greater than the threshold and therefore the first unit to be asynchronously selected will flip its state to 1. No other unit can change its state after this first unit has been set 1.

The Eight Rooks Problem

- n rooks must be positioned on an n * n chessboard so no one figure can take another. Each rook must be positioned on a different row and column to others.

A solution to the 4-rooks problem

			1
	1		
1			
		1	

Hopfield Network

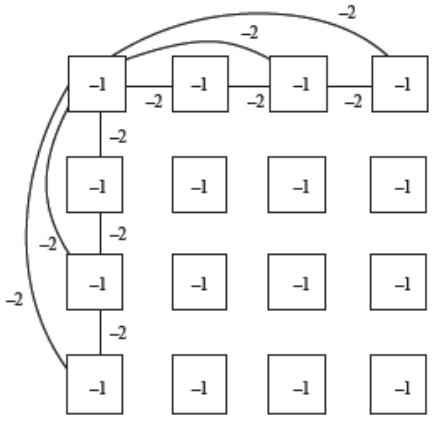


Fig. 13.16. Network for the solution of a four rooks problem

- Each field is represented by a unit.
- The connections of each unit to all elements in the same row or column have the weight – 2, all others have a weight 0.
- All units have a threshold of -1.
- Any unit set to 1 inhibits any other units in the same row or column. [cut off from notes]

Eight Queens Problem

- 8 Queens on an 8 * 8 chessboard so that no two Queen are attacking.
- Problem is solved by overlapping multiflop problems at each square.

Solution to 4-queens problem

			Q
Q			
			Q
	Q		

Hopfield network…must inhibit rows, columns and diagonals….

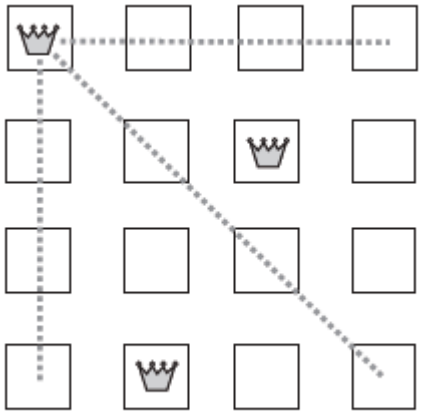


Fig. 13.17. The eight queens problem

- Three multiflop chains for each field.
- Wij = -2 when unit i is different from unit j and belongs to the same row, column or diagonal as unit j
- Otherwise, Wij ← 0. Threshold of all units ← -1.
- We do not always get a correct solution. Energy function [cut off]

The Traveling Salesperson Problem (TSP)

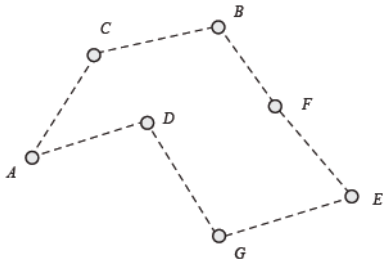


Fig. 13.18. A TSP and its solution

- A round trip can be represented using a matrix.
- Each of the n rows of the matrix is associated with a city.
- The columns are labeled 1 through n and they correspond to the n necessary visits.

	1	2	3	4
S_1	1	0	0	0
S_2	0	1	0	0
S_3	0	0	1	0
S_4	0	0	0	1

- This matrix shows a path going through the cities S1, S2, S3 and S4 in that order.

- A single 1 is allowed in each row and each column (same conditions as rooks problem).

- We need to minimize

$$L = 1/2 \sum_{i,j,k}^{n} dij \, Xik \, Xj, k+1$$

Hopfield and Tank – were the first to use a Hopfield network to solve a difficult problem.

- Could NP – Complete problems be solved in polynomial time? (or at least approximate solutions obtained)

- However, to obtain an optimal solution to an NP-Complete problem, the size of the network explodes exponentially.

- Large Hopfield networks may however be implemented if electrical circuits or optical circuits are employed.

Artificial Neural Networks
Lecture Notes

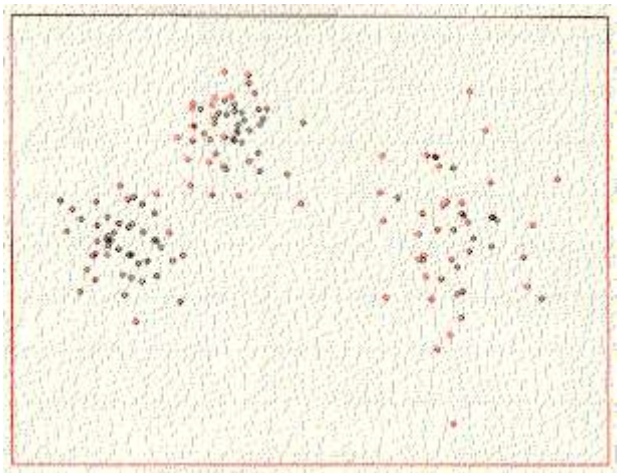
Chapter 13

Contents

Self-Organization

Introduction

- A neural network discovers clusters of similar patterns in data without supervision (i.e., no target information is provided.)



A Typical Dataset

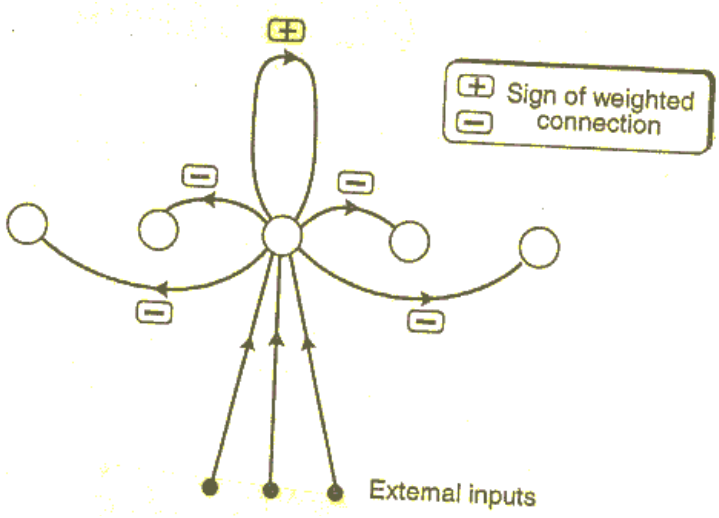
- Self-organization and unsupervised learning -

The network encodes such data by assigning nodes to clusters in some way.

- Network is supplied with extra resources to search for the location of largest activity.

Competitive Networks

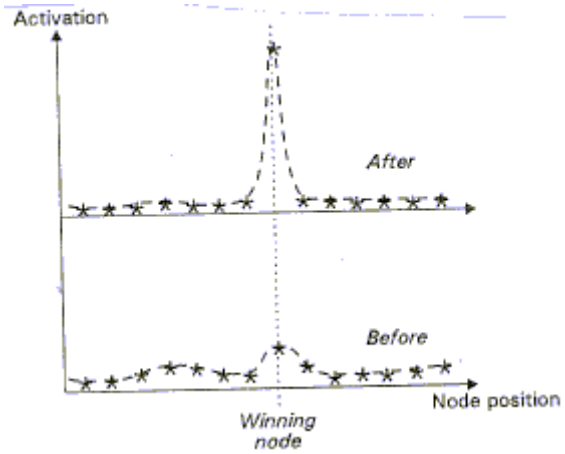
- A network of units as shown below



- Each unit receives the same set of inputs from an external input layer.
- There are intra-layer or lateral connections such that each node j is connected to itself via an excitatory (positive) weight $v_j^{(+)}$ and inhibits all other nodes in the layer with negative weights $v_{ij}^{(-)}$ - these are symmetric, i.e., $v_{ij}^{(-)} = v_{ji}^{(-)}$.
- This is said to be an *on-center, off-surround connection scheme*.
- Upon input $x;^{-}$, each unit computes its external input $s = x;^{-}.w;^{-}$
- One node, say K, will have maximal s.
  Activation a_k will increase, while other activations decrease.
- The total input input to each node j consists of:
  - The external input s_j, and
  - a contribution ℓ_j from other nodes in the layer.
- Node's output $y_j = \sigma(a_j)$ then l_j may be written thus

$$\ell_j = v_j^{(+)} y_j + \sum_{i \neq j} v_{ij}^{(-)} y_i$$

- Node K has its activation stimulated directly from the external input more strongly than any other node.
- This is reinforced via the self-excitatory connection.
- As output y_k grows, node k starts to inhibit the other nodes more than they can inhibit node k.
- The layer gradually reaches a point of equilibrium:



Winner Takes All

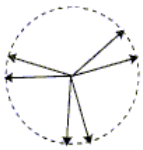
Requirements For There To Be A Unique Winning Node

- Only one excitatory weight emanating from any node that is connected to itself (vs. a small excitatory neighborhood.)
- Lateral connections sufficiently strong to ensure a well-defined equilibrium.
- The inhibitory connections from each node must extend to all other nodes.

Competitive Learning

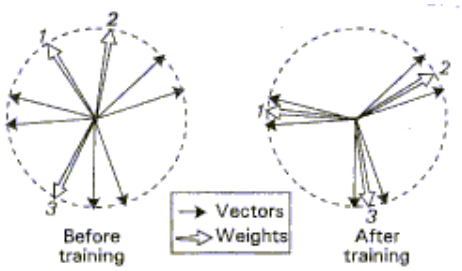
- Consider a training set whose vectors all have the same unit length, i.e., $||\bar{x}|| = 1$ for all $\bar{x}$.
- Normalization may be required:

  $\bar{x}' = (1 / || \bar{x} ||)\bar{x}$



In 2-D the vectors all fall on the unit circle.

- **Example**:
  - A competitive neural layer with a set of normalized weight vectors for connection to external input.



Normalized weights and vectors.

  - There are three nodes and three clusters - we would expect to be able to encode each cluster with a single node. Then, when a vector is presented to the net, there will be a single node that responds maximally to the input.

  - Note, s will be large and positive if the weight and input vectors are well aligned.
Recall,

$$s = \| \bar{w} \| \ \| \bar{x} \| \cos\theta$$

  - The ideal network will have its three weight vectors aligned with the three pattern clusters.

  - This may be achieved by "rotating" each weight vector so that it becomes aligned with the cluster it is nearest to.

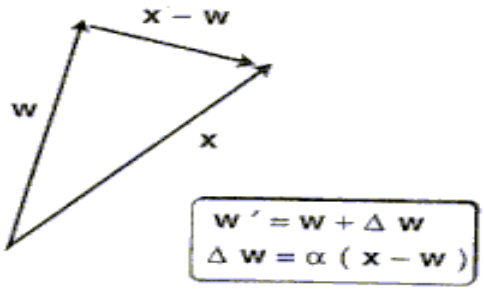
Training Methodology

- Iteratively apply vectors and adjust the weights of the node whose external input
is largest.

- Let the "winning node" have index k and weight vector $\bar{w}_k$ (note $\bar{w}_k$ is the k^{th} weight vector, not the k^{th} component of $\bar{w}_k$ which is denoted w_k.)

- $\bar{w}_k$ should be rotated toward $\bar{x}$.

Thus, we merely add a fraction of the difference vector $\bar{x} - \bar{w}_k$



- **The Learning Rule**:

$$\Delta \overline{w}_j = \begin{cases} \alpha(\overline{x} - \overline{w}_j) , j = k \\ 0 , j \neq k. \end{cases}$$

- If the network is "winner-takes-all", then node k will have its output close to one while others close to zero.

- After letting network reach equilibrium:
$\Delta \bar{w}_j = \alpha(\bar{x} - \bar{w}_j)\, y$.

Training Algorithm

1. Apply a vector at the input to the network and evaluate s for each node.
2. Update the net until it reaches equilibrium where

$$da_j / dt = c_1 (s_j + \ell_j - c_2 a_j.$$

3. Train all nodes using:

$$\Delta \bar{w}_j = \alpha(\bar{x} - \bar{w}_j)\, y .$$

- Note: It is possible that if the patterns are not so well clustered and if there are many more patterns than nodes, then the weight coding may be unstable.

Problem with Normalization

- Weights are initially of unit length - as they adapt, their length will change. It would be impractical to renormalize after each training step.

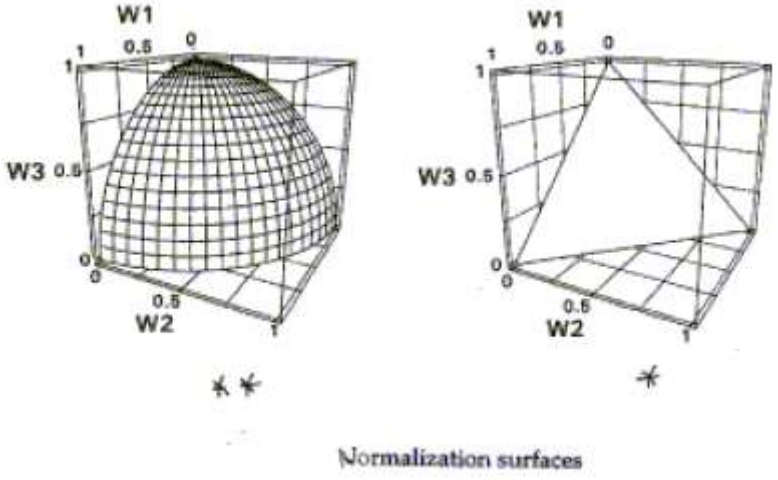
- If all inputs are positive

$$\sum_i w_i = 1 \quad *$$

is approximately equal to Euclidean length.
Since the Euclidean length is one, the squared length is also one

$$\sum_i w_i^2 = 1 \quad **$$

****** defines point on a unit sphere.
***** defines point on a plane.



Normalization surfaces

- According to how closely the distance of the plane from the origin approximates to one, the two schemes may be said to be equivalent.
- Our learning rule may be rewritten as
  $\Delta = \alpha x; \bar{y} - \alpha w; \bar{y}$.

The first term looks like *Hebbian Learning*.

The second is the *weight decay*.

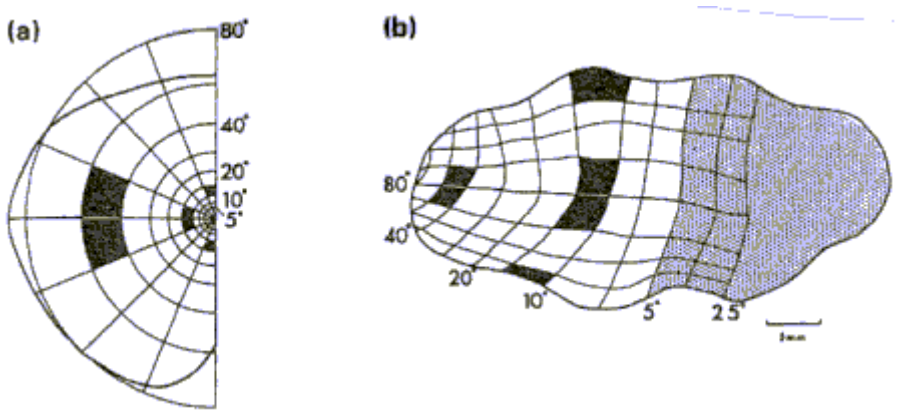
This latter property has a biological interpretation in terms of preservation of metabolic resources - the sum of synaptic strengths may not exceed a certain value.

Kohonen's Self-Organizing Feature Maps

- Competitive Networks:
  - Identify clusters in training data.
  - Also, nodes that are physically adjacent in the network encode patterns that are "adjacent" in some sense, in the pattern space of the input.

- The concept of proximity leads to the idea of a topography or map defined over a neural layer in which these maps represent some feature of the input space.

- The mammalian cortex is the externally visible sheet of neural tissue that is folded and wrapped to enclose more central areas of the brain. The cortex is responsible for processing sensory information, e.g., sound and vision.



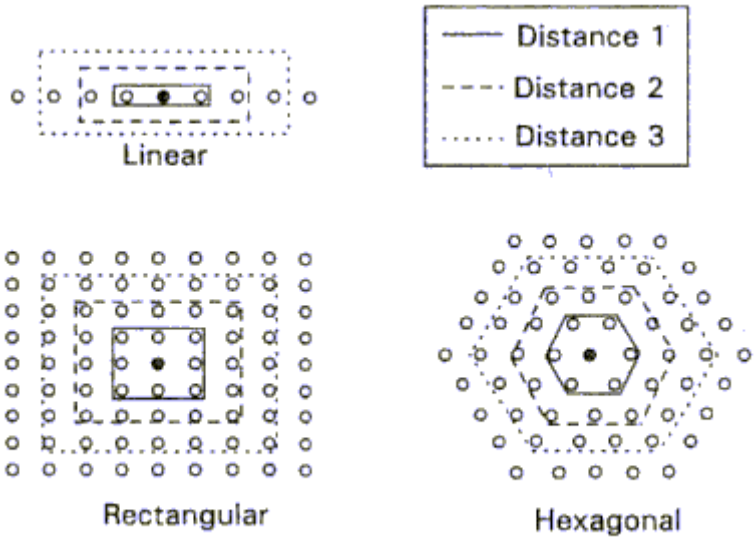
SOM Algorithm

- The network architecture consists of a set of inputs that are fully connected to the self-organizing layer, but now there are no lateral connections.

- The key principle for map formation is that training should take place over an extended region of the network centered on the maximally active node.

- Neighborhood Schemes:
  - Linear array of nodes
  - rectangular grid
  - hexagonal grid

- Three neighborhoods are shown delimited with respect to a shaded unit at distances of 1, 2 and 3 away from this node.



Neighbourhood schemes for SOMs.

- The linear, rectangular and hexagonal arrays have 5, 25 and 19 nodes respectively in their distance-2 neighborhoods (including the center node.)
- Weights are initialized to small random values.
- The neighborhood distance d_N is set to cover over half the network.
- Vectors are drawn randomly from the training set.
- The following operations performed at each selection:

  1. Find the best matching or "winning" node k whose weight vector $\bar{w}_k$ is closest to the current input vector using the vector difference as criterion:

$$\| \bar{w}_k - \bar{x} \| = \min\{ \| \bar{w}_i - \bar{x} \| \}$$

  2. Train node k and all nodes in the neighborhood N_k of k

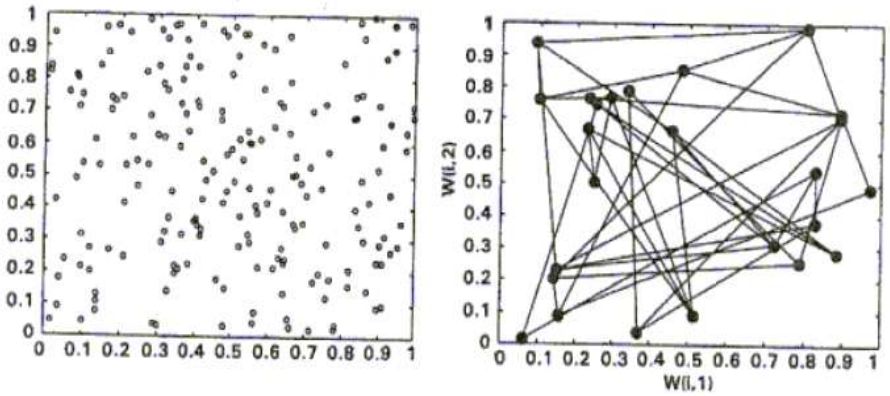
$$\Delta \bar{w}_j = \begin{cases} \alpha(\bar{x} - \bar{w}_j) \text{ , if j is in } N_k \\ 0 \text{ , if j is not in } N_k \end{cases}$$

  3. Decrease the learning rate slightly.

  4. After a certain number M of cycles, decrease the size of the neighborhood d_N.

- **An Example:**

200 vectors chosen at random from the unit square in pattern space and used to train a net of 25 nodes on a 5 * 5 rectangular grid -



A 25-node net: training set and initial weight space.