TABLE OF CONTENTS

Unit I

Chapter - 1 Introduction to ANN **(1 - 1) to (1 - 34)**

1.1 Introduction to ANN ..... 1 - 2

1.2 History of Neural Network ..... 1 - 3

1.3 Structure and Working of Biological Neural Network ..... 1 - 3

1.4 Introduction to Neural Network ..... 1 - 5

1.4.1 Advantages of Neural Network ..... 1 - 7

1.4.2 Application of Neural Network ..... 1 - 7

1.4.3 Difference between Digital Computer and Neural Networks ..... 1 - 8

1.5 Neural Net Architecture ..... 1 - 8

1.5.1 Single Layer Feed Forward Network ..... 1 - 8

1.5.2 Multi-Layer Feed Forward Network ..... 1 - 12

1.5.3 Recurrent Neural Network ..... 1 - 17

1.6 Activation Functions ..... 1 - 18

1.6.1 Identity or Linear Activation Function ..... 1 - 20

1.6.2 Sigmoid ..... 1 - 21

1.6.3 Tanh and ReLU ..... 1 - 22

1.7 Models of Neuron-McCulloch and Pitts Model ..... 1 - 23

1.8 Perceptron ..... 1 - 25

1.8.1 Single Layer Perceptron ..... 1 - 25

1.8.2 Multilayer Perceptron ..... 1 - 27

1.8.3 ADALINE Network ..... 1 - 29

1.9 Basic Learning Laws ..... 1 - 31

1.9.1 Weights and Bias ..... 1 - 31

Unit II

Chapter - 2 Learning Algorithms (2 - 1) to (2 - 26)

2.1 Learning and Memory 2-2
2.1.1 Auto-associative Memory 2-3
2.1.2 Hetero-associative Memory Network 2-4
2.1.3 The Hopfield Network 2-5
2.1.4 Bidirectional Associative Memory (BAM) 2-6
2.1.5 Difference between Auto-associative Memory and Hetero-Associative Memory 2-7
2.2 Hebbian Learning 2-7
2.3 Competitive Learning 2-10
2.4 Error Correction 2-11
2.5 Gradient Decent Rules 2-12
2.5.1 Finding the Optimal Hyper-Parameters through Grid Search 2-13
2.5.2 Vanishing Gradient Problem 2-15
2.6 Supervised Learning 2-16
2.6.1 Advantages and Disadvantages of Supervised Learning 2-19
2.6.2 Difference between Supervised and Unsupervised Learning 2-19
2.7 Backpropagation 2-20
2.7.1 Advantages and Disadvantages 2-23
2.8 Feed Forward Neural Networks 2-23
2.8.1 Feedback Neural Networks 2-24
2.8.3 Example and Applications of Feedforward NN 2-25
2.8.4 Difference between RNNs and feed-forward NN 2-25

Unit III

Chapter - 3 Associative Learning (3 - 1) to (3 - 58)

3.1 Associative Learning, Hopfield Networks, Simulated Annealing, Boltzman Machine 3-2
3.1.1 Associative Learning 3-2
3.1.2 Hopfield Network 3-4
3.1.2.1 Introduction 3-4

3.1.2.2 Characteristics of Hopfield Network - Qualities and Attributes ..... 3 - 5
3.1.2.3 Activation Function and Thresholding Function ..... 3 - 6
3.1.2.4 Evolution Towards a Memorized Pattern in Hopfield Network ..... 3 - 8
3.1.2.5 Types of Hopfield Networks (Discrete Hopfield Network and Continuous Hopfield Network) ..... 3 - 9
3.1.2.6 Energy Function ..... 3 - 11
3.1.2.7 Error Performance in Hopfield Networks ..... 3 - 11
3.2 Simulated Annealing ..... 3 - 12
3.2.1 Introduction ..... 3 - 12
3.2.2 Simulated Annealing terms ..... 3 - 14
3.2.3 Simulated Annealing Algorithm ..... 3 - 15
3.2.4 Advantages vs. Disadvantages of Simulated Annealing ..... 3 - 18
3.3 Boltzmann Machine and Boltzmann Learning ..... 3 - 18
3.3.1 Introduction ..... 3 - 18
3.3.2 Objective of Boltzmann Machine and Architecture ..... 3 - 19
3.3.3 Learning in Boltzmann Machine ..... 3 - 22
3.3.4 Uses of Boltzmann Machine ..... 3 - 23
3.3.5 Energy-Based Models ..... 3 - 23
3.3.6 Types of Boltzmann Machines (Restricted BMs, Deep Belief Networks, Deep BMs) ..... 3 - 24
3.3.7 Limitations of Boltzmann Machines ..... 3 - 31
3.4 Basic Functional Units of ANN for Pattern Recognition Tasks ..... 3 - 32
3.4.1 Introduction to Pattern Recognition ..... 3 - 32
3.4.2 Pattern Recognition Tasks ..... 3 - 32
3.4.3 Input Processing for Pattern Recognition ..... 3 - 34
3.4.4 Functional Units of Pattern Recognition and Working of ANN ..... 3 - 36
3.4.4.1 Pattern Recognition Tasks by Feedforward ANN ..... 3 - 37
3.4.4.2 Pattern Recognition Tasks by Feedback ANN ..... 3 - 40
3.4.4.3 Pattern Recognition Tasks by Feedforward and Feedback ANN ..... 3 - 54
Review Questions with Answers ..... 3 - 57

Unit IV

Chapter - 4 Competitive Learning Neural Network (4 - 1) to (4 - 24)

4.1 Competitive Learning and Adaptive Resonance Theory (ART) ..... 4 - 2
4.1.1 Competitive Learning ..... 4 - 2
4.1.1.1 Introduction to Competitive Learning ..... 4 - 2

4.1.1.2 Components of CL network ..... 4 - 2
4.1.2 ART Networks - Working, Pattern Clustering, Feature Mapping, Advantages, Applications ..... 4 - 8
4.1.2.1 Introduction to ART Networks ..... 4 - 8
4.1.2.2 Operational Principal and Features of ART Models, ART 1 ..... 4 - 9
4.1.2.3 ART Applications Character Recognition using ART Network ..... 4 - 14
4.2 Self-Organization Maps (SOM) - Two Basic Feature Mapping Models, Self-Organization Map, SOM Algorithm, Properties of Feature Map, Computer Simulations, Learning Vector Quantization, Adaptive Pattern Classification ..... 4 - 16
4.2.1 Basics of SOM ..... 4 - 16
4.2.2 Working of SOM ..... 4 - 17
4.2.3 Feature Map ..... 4 - 19
4.2.4 Advantages and Disadvantages of Self - organizing Maps ..... 4 - 22
4.2.5 Learning Vector Quantization ..... 4 - 22
Review Questions with Answers ..... 4 - 24

Unit V

Chapter - 5 Convolution Neural Network (5 - 1) to (5 - 36)
5.1 Introduction to Convolution Neural Network ..... 5 - 2
5.1.1 Advantages and Disadvantages of CNN ..... 5 - 3
5.1.2 Application of CNN ..... 5 - 3
5.2 The Basic Structure of CNN ..... 5 - 4
5.3 Convolution Operation ..... 5 - 4
5.3.1 Parameter Sharing ..... 5 - 7
5.3.2 Equivariant Representation ..... 5 - 8
5.3.3 Padding ..... 5 - 9
5.3.4 Stride ..... 5 - 10
5.3.5 Typical Setting ..... 5 - 11
5.3.6 ReLU Layer ..... 5 - 11
5.4 Pooling ..... 5 - 11
5.5 Fully Connected Layers ..... 5 - 13
5.6 Convolutions Over Volume ..... 5 - 13
5.7 SoftMax Regression ..... 5 - 14
5.8 Deep Learning Frameworks ..... 5 - 15

5.8.1 TensorFlow ..... 5 - 17
5.8.2 Keras ..... 5 - 19
5.8.2.1 Difference Between Keras and Tensorflow ..... 5 - 20
5.8.3 PyTorch ..... 5 - 20
5.8.4 Caffe ..... 5 - 21
5.8.5 Shogun ..... 5 - 21
5.9 Bias and Variance with Mismatched Data Distributions ..... 5 - 22
5.10 Learning Representations from Data ..... 5 - 24
5.10.1 Greedy Layer - Wise Unsupervised Pretraining ..... 5 - 25
5.10.2 Transfer Learning and Domain Adaptation ..... 5 - 27
5.11 Multi-Task Learning ..... 5 - 27
5.11.1 Types of Multi-Task Learning ..... 5 - 28
5.11.2 Advantages of Multi-task Learning ..... 5 - 29
5.12 End-to-End Deep Learning ..... 5 - 29
5.13 Introduction to CNN Models ..... 5 - 30
5.13.1 LeNet - 5 ..... 5 - 30
5.13.2 AlexNet ..... 5 - 31
5.13.3 VGG - 16 ..... 5 - 33
5.13.4 Residual Networks ..... 5 - 34

Unit VI

Chapter - 6 Applications of ANN **(6 - 1) to (6 - 24)**

6.1 Applications Artificial Neural Network I - Pattern Classification ..... 6 - 2
6.2 Applications Artificial Neural Network II - Recognition of Printed Characters ..... 6 - 7
6.3 Applications Artificial Neural Network III - Neocognitron - Recognition of Handwritten Characters ..... 6 - 8
6.4 Applications Artificial Neural Network IV - NETtalk - Convert English Text to Speech Application ..... 6 - 18
6.5 Applications Artificial Neural Network V - Recognition of Consonant Vowel (CV) Segments, Texture Classification and Segmentation ..... 6 - 20
Review Questions with Answers ..... 6 - 23

Solved Model Question Papers **(M - 1) to (M - 4)**

Unit I

1 Introduction to ANN

Syllabus

Introduction to ANN, History of Neural Network, Structure and working of Biological Neural Network, Neural net architecture, Topology of neural network architecture, Features, Characteristics, Types, Activation functions, Models of neuron-Mc Culloch & Pitts model, Perceptron, Adaline model, Basic learning laws, Applications of neural networks, Comparison of BNN and ANN.

Contents

1.1 Introduction to ANN

1.2 History of Neural Network

1.3 Structure and Working of Biological Neural Network

1.4 Introduction to Neural Network

1.5 Neural Net Architecture

1.6 Activation Functions

1.7 Models of Neuron-McCulloch and Pitts Model

1.8 Perceptron

1.9 Basic Learning Laws

1.1 Introduction to ANN

- Artificial Neural Network (ANN) is a computational system inspired by the structure, processing method, learning ability of a biological brain. An artificial neural network is composed of many artificial neurons that are linked together according to specific network architecture. The objective of the neural network is to transform the inputs into meaningful outputs.
- ANNs do not execute programmed instructions; they respond in parallel to the pattern of inputs presented to it. There are also no separate memory addresses for storing data. Instead, information is contained in the overall activation 'state' of the network. 'Knowledge' is thus represented by the network itself, which is quite literally more than the sum of its individual components.
- Fig. 1.1.1 shows artificial neural network.

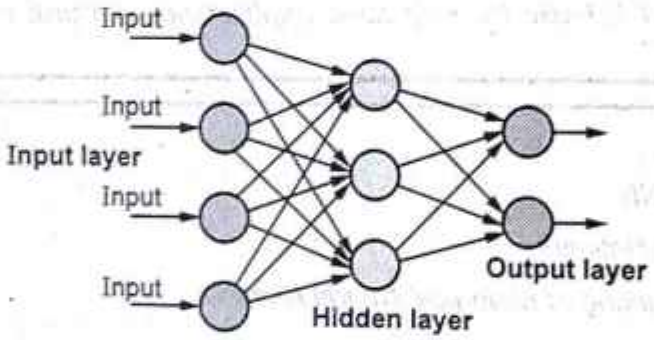


Fig. 1.1.1 Artificial neural network

- Elements of ANN are processing units, topology and learning algorithm.
- Tasks to be solved by artificial neural networks :
  1. Controlling the movements of a robot based on self-perception and other information;
  2. Deciding the category of potential food items in an artificial world;
  3. Recognizing a visual object;
  4. Predicting where a moving object goes, when a robot wants to catch it.
- Characteristics of artificial neural networks :
  1. Large number of very simple processing neuron-like processing elements.
  2. Large number of weighted connections between the elements.
  3. Distributed representation of knowledge over the connections.
  4. Knowledge is acquired by network through a learning process.

1.2 History of Neural Network

- Neural networks were first proposed in 1943 by Warren McCullough and Walter Pitts, two University of Chicago researchers who moved to MIT in 1952 as founding members of what's sometimes called the first cognitive science department.
- Pitts and McCulloch researched and wrote a paper describing how neurons might work and then developed a simple neural network using electrical circuits.
- By deploying logic gates, they were able to demonstrate how outputs could be activated when specified inputs are active, to model how neurons work in the brain. As a model of the brain, the functionality provided was considerably simpler compared to the billions of neurons in the human brain. In addition, their model was only able to process binary inputs and outputs, providing a very simplified model of a neuron.
- Frank Rosenblatt proposed the idea of a Perceptron in 1958, calling it Mark I Perceptron. It was a system with a simple input output relationship, modeled on a McCulloch-Pitts neuron. A McCulloch-Pitts neuron takes in inputs, takes a weighted sum and returns '0' if the result is below threshold and '1' otherwise.
- The beauty of Mark I Perceptron lay in the fact that its weights would be 'learnt' through successively passed inputs, while minimizing the difference between desired and actual output.
- In 1959 at Stanford, Bernard Widrow and Marcian Hoff developed the first neural network successfully applied to a real world problem. These systems were named ADALINE and MADALINE after their use of Multiple ADAptive LINear Elements.
- Backpropagation along with Gradient Descent forms the backbone and powerhouse of neural networks. While Gradient Descent constantly updates and moves the weights and bias towards the minimum of the cost function, backpropagation evaluates the gradient of the cost w.r.t. weights and biases, the magnitude and direction of which is used by gradient descent to evaluate the size and direction of the corrections to weights and bias parameters.
- By the 1980s, however, researchers had developed algorithms for modifying neural nets' weights and thresholds that were efficient enough for networks with more than one layer, removing many of the limitations identified by Minsky and Papert. The field enjoyed a renaissance.

1.3 Structure and Working of Biological Neural Network

- Artificial neural systems are inspired by biological neural systems. The elementary building block of biological neural systems is the neuron.

- The brain is a collection of about 10 billion interconnected neurons. Each neuron is a cell [right] that uses biochemical reactions to receive, process and transmit information. Fig. 1.3.1 shows biological neural systems.

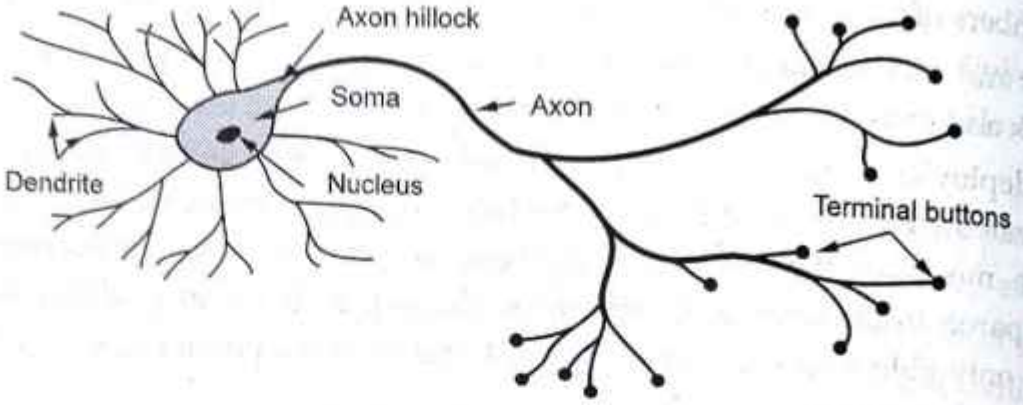


Fig. 1.3.1 Schematic of biological neuron

- The single cell neuron consists of the cell body or soma, the dendrites and the axon. The dendrites receive signals from the axons of other neurons. The small space between the axon of one neuron and the dendrite of another is the synapse. The afferent dendrites conduct impulses toward the soma. The efferent axon conducts impulses away from the soma.

Basic Components of Biological Neurons

1. The majority of ***neurons*** encode their activations or outputs as a series of brief electrical pulses (i.e. spikes or action potentials).
2. The neuron's ***cell body (soma)*** processes the incoming activations and converts them into output activations.
3. The neuron's ***nucleus*** contains the genetic material in the form of DNA. This exists in most types of cells, not just neurons.
4. ***Dendrites*** are fibres which emanate from the cell body and provide the receptive zones that receive activation from other neurons.
5. ***Axons*** are fibres acting as transmission lines that send activation to other neurons.
6. The junctions that allow signal transmission between the axons and dendrites are called ***synapses***. The process of transmission is by diffusion of chemicals called ***neurotransmitters*** across the synaptic cleft.

Comparison between Biological NN and Artificial NN

Biological NN	Artificial NN
soma	unit
Axon, dendrite	dendrite
synapse	weight
potential	weighted sum
threshold	bias weight
signal	activation

1.4 Introduction to Neural Network

- Neural networks consists of many numbers of simple elements (neurons) connected between them in system. Whole system is able to solve of complex tasks and to learn for it like a natural brain.
- Neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths and the processing performed at computing elements or nodes.
- For user, NN is black box with input vector (source data) and output vector (result).
  - A Neural network is usually structured into an input layer of neurons, one or more hidden layers one output layer.
  - Neurons belonging to adjacent layers are usually fully connected and the various types and architectures are indentified both by the different topologies adopted for the connections as well as by the choice of the activation function.
  - The values of the functions associated with the connections are called "weights".
  - The whole game of using NNs is in fact that, in order for the network to yield appropriate outputs for given inputs, the weight must be set to suitable values. The way this is obtained allows a further distinction among modes of operations.
  - A neural network is a processing device, either an algorithm or actual hardware, whose design was motivated by the design and functioning of human brains and components thereof.

- Most neural networks have some sort of "training" rule whereby the weights of connections are adjusted on the basis of presented patterns.
- In other words, neural networks "learn" from example, just like children learn to recognize dogs from examples of dogs and exhibit some structural capability for generalization.
- Neural networks normally have great potential for paralleism, since the computations of the components are independent of each other.
- Neural networks are a different paradigm for computing :
  1. Von Neumann machines are based on the processing/memory abstraction of human information processing.
  2. Neural networks are based on the parallel architecture of animal brains.

- Neural networks are a form of multiprocessor computer system, with
  a. Simple processing elements
  b. A high degree of interconnections
  c. Simple scalar messages
  d. Adaptive interaction between elements
- The advantages of neural networks are due to its adaptive and generalization ability.
  a) Neural networks are adaptive methods that can learn without any prior assumption of the underlying data.
  b) Neural network, namely the feed forward multilayer perception and radial basis function network have been proven to be universal functional approximations.
  c) Neural networks are non-linear model with good generalization ability.
- **Useful properties and capabilities of neural network**
  1. **Nonlinearity :** An artificial neuron can be linear or nonlinear. A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear.
  2. **Adaptivity :** Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment.

3. **Contextual information :** Knowledge is represented by the very structured and activation state of a neural network.
4. **Evidential response :** In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to select, but also about the confidence in the decision made.
5. **Uniformity of analysis and design :** Neural networks enjoy universality as information processors.
6. **VLSI implement-ability :** The massively parallel nature of a neural network makes it potentially fast for the computation of certain tasks.

1.4.1 Advantages of Neural Network

- The advantages of neural networks are due to its adaptive and generalization ability.
  a) Neural networks are adaptive methods that can learn without any prior assumption of the underlying data.
  b) Neural network, namely the feed forward multilayer perception and radial basis function network have been proven to be universal functional approximations.
  c) Neural networks are non-linear model with good generalization ability.

1.4.2 Application of Neural Network

Neural network applications can be grouped in following categories :

1. **Clustering :** A clustering algorithm explores the similarity between patterns and places similar patterns in a cluster. Best known applications include data compression and data mining.
2. **Classification/Pattern recognition :** The task of pattern recognition is to assign an input pattern (like handwritten symbol) to one of many classes. This category includes algorithmic implementations such as associative memory.
3. **Function approximation :** The tasks of function approximation is to find an estimate of the unknown function f() subject to noise. Various engineering and scientific disciplines require function approximation.
4. **Prediction/Dynamical systems :** The task is to forecast some future values of a time-sequenced data. Prediction has a significant impact on decision support systems. Prediction differs from function approximation by considering time factor.

1.4.3 Difference between Digital Computer and Neural Networks

Sr. No.	Digital Computers	Neural Networks
1.	Deductive reasoning : We apply known rules input data to produce output.	Inductive reasoning : Given input and output data (training examples), we construct the rules.
2.	Computation is centralized, synchronous and serial.	Computation is collective asynchronous and parallel.
3.	Memory is packetted, literally stored and location addressable.	Memory is distributed, internalized and content addressable.
4.	Not fault toerant. One transistor goes and it no longer works.	Fault tolerant, redundancy and sharing of responsibilities.
5.	Fast. Measured in millionths of a second.	Slow. Measured in thousands of a second.
6.	Exact.	Inexact.
7.	Static connectivity.	Dynamic connectivity.
8.	Applicable if well defined rules with precise input data.	Applicable if rules are unknown or complicated or if data is noisy or partial.

1.5 Neural Net Architecture

1.5.1 Single Layer Feed Forward Network

- The architecture of the neural network refers to the arrangement of the connection between neurons, processing element, number of layers and the flow of signal in the neural network.
- There are mainly two category of neural network architecture :
  a. Feed-forward
  b. Feedback (recurrent) neural networks.

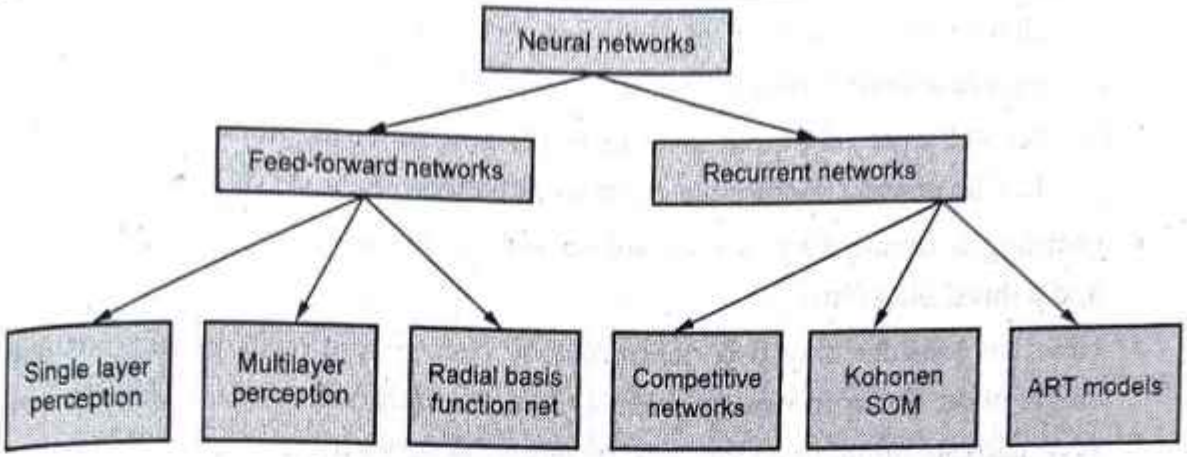


Fig. 1.5.1

1. Architecture and Learning Rule

- In late 1950s, Frank Rosenblatt introduced a network composed of the units that were enhanced version of McCulloch-Pitts Threshold Logic Unit (TLU) model.
- Rosenblatt's model of neuron, a perceptron, was the result of merger between two concepts from the 1940s, McCulloch-Pitts model of an artificial neuron and Hebbian learning rule of adjusting weights.
- In addition to the variable weight values, the perceptron model added an extra input that represents bias. Thus, the modified equation is now as follows :

$$\text{Sum} = \sum_{i=1}^{N} I_i W_i + b,$$

where **b** represents the bias value.

- Fig. 1.5.2 shows a typical perception setup for pattern recognition applications, in which visual patterns are represented as matrices of elements between 0 and 1.

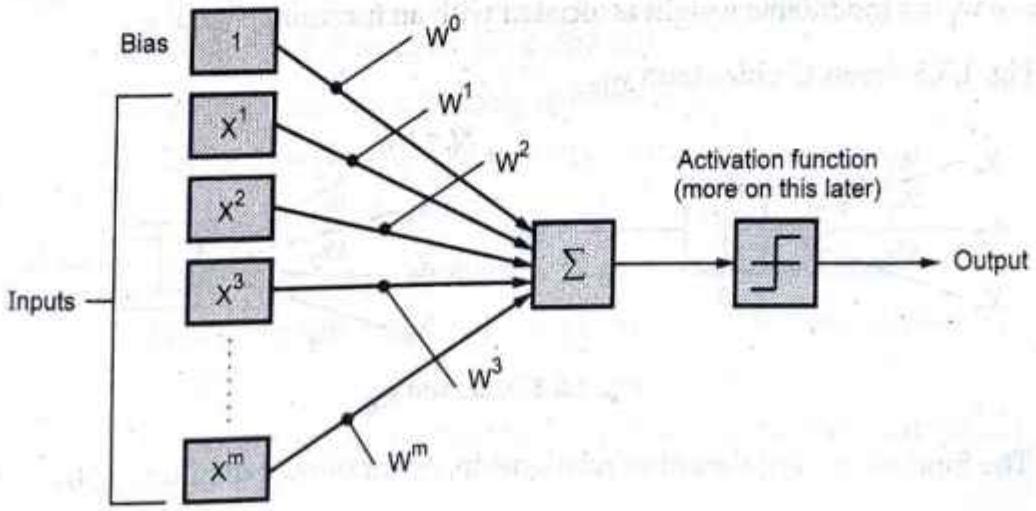


Fig. 1.5.2 Perception setup

1. First layer act as a set of feature detectors that are hardwired to the input signals to detect specific features.
2. Second layer i.e. output layer takes the outputs of the feature detectors in the first layer and classifies the given input pattern.

- Learning is initiated by making adjustments to the relevant connection strengths and a threshold value.
- Here we consider only two class problem. Here output layer usually has only a single node. For an n-class problem (n > 3), the output layer usually has n-nodes, each corresponding to a class and the output node with the largest value indicates which class the input vector belongs to.
- In the first stage, the linear combination of inputs is calculated. Each value of input array is associated with its weight value, which is normally between 0 and 1. Also, the summation function often takes an extra input value Theta with weight value of 1 to represent threshold or bias of a neuron.
  a. The term x_i is referred to as **active or excitatory** if its value is 1.
  b. If the value is 0 then it is **inactive**.
  c. If the value is – 1 then it is **inhibitory**.
- The output unit is a linear threshold element with a threshold value :

$$0 = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

$$= f\left(\sum_{i=1}^{n} w_i x_i - w_0\right), w_0 \equiv -\theta$$

$$= f\left(\sum_{i=1}^{n} w_i x_i\right), x_0 \equiv 1$$

where w_i is a modifiable weight associated with an incoming signal x_i.

- Fig. 1.5.3 shows the bias term w_0.

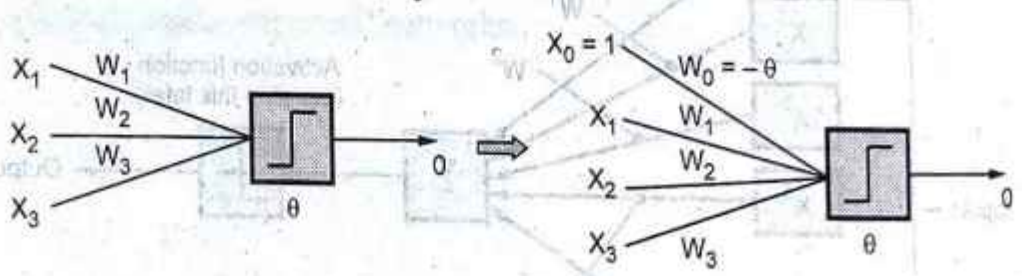


Fig. 1.5.3 Bias term w_0

- The function y = f(x) describes relationship, an input-output mapping from x to y.

- The equation (1.5.1), the f(.) is the activation function of the perceptron and it is typically either a signum function sgn(x) or step function step(x) :

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{otherwise} \end{cases}$$

$$\text{step}(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases} \quad \ldots (1.5.1)$$

- The sum-of-product value is then passed into the second stage to perform the activation function which generates the output from the neuron. The activation function "squashes" the amplitude of the output in the range of [0, 1] or [– 1, 1] alternately. The behavior of the activation function will describe the characteristics of an artificial neuron model.
- The basic learning algorithm for a single layer perceptron repeats the following steps until the weights converge :
  1. Select an input vector x from the training data set.
  2. If the perceptron gives an incorrect response, modify all connection weights w_i according to

$$\Delta w_i = \eta t_i X_i$$

Where t_i is a target output and η is a learning state.

Perceptron Convergence Theorem

Theorem : If there is a set of weights that correctly classify the (linearly separable) training patterns, then the learning algorithm will find one such weight set, w^* in a finite number of iterations.

Assumptions :

1. At least one such set of weights, w^*, exists and
2. There are a finite number of training patterns.
3. The threshold function is uni-polar (output is 0 or 1).

2. Exclusive OR problem

- XOR problem is a pattern recognition problem in neural network.
- Neural networks can be used to classify boolean functions depending on their desired outputs.
- For a two input binary XOR problem, the desired output is given in the form of truth table.

	X	Y	Class
Desired I/O pair 1	0	0	0
Desired I/O pair 2	0	1	1
Desired I/O pair 3	1	0	1
Desired I/O pair 4	1	1	0

- The XOR problem is not linearly separable. We cannot use a single layer perceptron to construct a straight line to partition the two dimensional input space into two regions, each containing only data points of the same class.
- Let us consider following four equations :

$$0 \times w_1 + 0 \times w_2 + w_0 \leq 0 \Leftrightarrow w_0 \leq 0,$$

$$0 \times w_1 + 1 \times w_2 + w_0 \, 0 \Leftrightarrow w_0 > -w_2$$

$$1 \times w_1 + 0 \times w_2 + w_0 > 0 \Leftrightarrow w_0 > -w_1$$

$$1 \times w_1 + 1 \times w_2 + w_0 \leq 0 \Leftrightarrow w_0 \leq -w_1 - w_2$$

1.5.2 Multi-Layer Feed Forward Network

- A multilayer feed-forward neural network is a network consisting of multiple layers of units, all of which are adaptive. The network is not allowed to have cycles from later layers back to earlier layers, hence the name "feed-forward". Let us consider a network with a single complete hidden layer. i.e., the network consists of some input nodes, some output nodes and a set of hidden nodes. Every hidden node takes inputs from each of the input nodes and feeds into each of the output nodes.

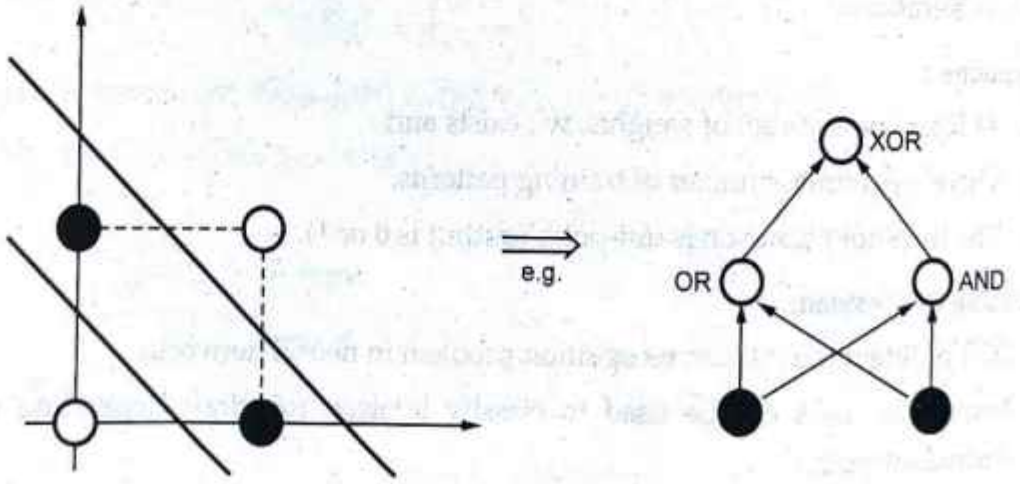


Fig. 1.5.4

- In multi-layer feed forward neural networks, the sigmoid activation function, defined by $g(x) = \frac{1}{1+e^{-x}}$ is normally used.
- A Multi-Layer Perceptron (MLP) has the same structure of a single layer perceptron with one or more hidden layers. An MLP is a network of simple neurons called perceptrons.
- A typical multilayer perceptron network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes.
- It is not possible to find weights which enable single layer perceptrons to deal with non-linearly separable problems like XOR :
- Multi-layer perceptrons are able to cope with non-linearly separable problems.
- Each neuron in one layer has direct connections to all the neurons of the subsequent layer. MLP can implement nonlinear discriminants (for classification) and nonlinear regression functions (for regression).
- Historically, the problem was that there were no known learning algorithms for training MLPs. Fortunately; it is now known to be quite straightforward. The procedure for finding a gradient vector in the network structure is generally referred to as **backpropagation**. Because the gradient vector is calculated in the direction opposite to the flow of the output of each node.
- Procedure of backpropagation :
  1. The output values are compared with the target to compute the value of some predefined error function.
  2. The error is then feedback through the network.
  3. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function.
- Continue this process until the connection weights in the network have been adjusted so that the network output has converged, to an acceptable level, with the desired output.
- If we use the gradient vector in a simple steepest descent method, the resulting learning paradigm is often referred to as the **backpropagation** learning rule. Backpropagation works by approximating the non-linear relationship between the input and the output by adjusting the weight values internally.

- Generally, the backpropagation network has two stages, training and testing. During the training phase, the network is "shown" sample inputs and the correct classifications. For example, the input might be an encoded picture of a face and the output could be represented by a code that corresponds to the name of the person.
- Fig. 1.5.5 shows three most commonly used activation functions in backpropagation MLPs.

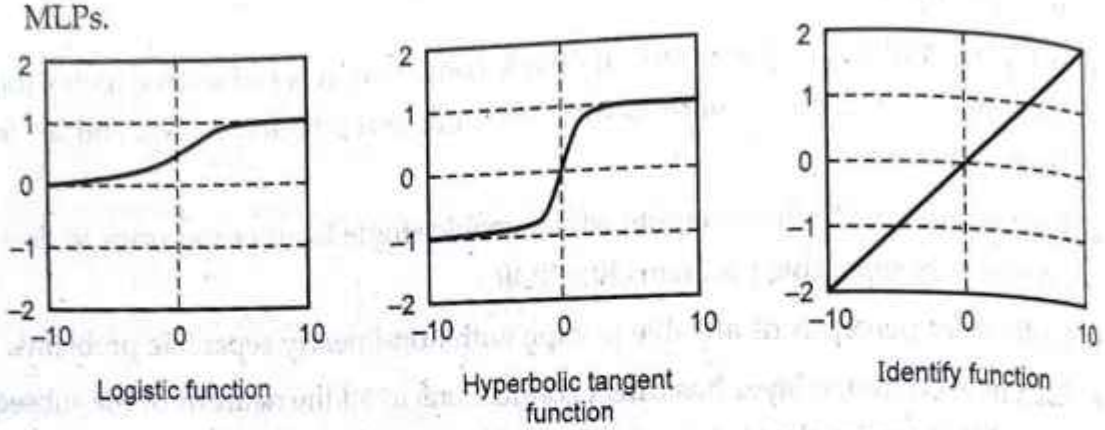


Fig. 1.5.5 Activation function

Logistic function :

$$f(x) = \frac{1}{1+e^{-x}}$$

Hyperbolic tangent function :

$$f(x) = \tanh(x/2) = \frac{1-e^{-x}}{1+e^{-x}}$$

Identity function :

$$f(x) = x$$

- Both the hyperbolic tangent function and logistic function approximate the signum and step function respectively. Sometimes these two function are referred to as **squashing functions** since the inputs to these functions are squashed to the range [0, 1] or [– 1, 1].
- These functions are also called **sigmoidal functions** because their S-shaped curves exhibits smoothness and asymptotic properties.
- A learning process is organized through a learning algorithm, which is a process of updating the weights in such a way that a machine learning tool implements a given input/output mapping with no errors or with some minimal acceptable error.

- Any learning algorithm is based on a certain learning rule, which determines how the weights shall be updated if the error occurs.

Backpropagation Learning Rule

- The net input of a node is defined as the weighted sum of the incoming signals plus a bias term. Fig. 1.5.6 shows the backpropagation MLP for node j. The net input and output of node j is as follows :

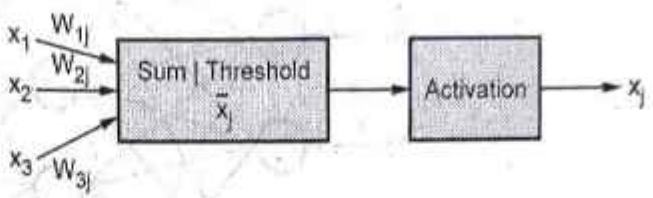


Fig. 1.5.6 Backpropagation MLP for node j

$$\bar{X}_j = \Sigma_i + W_{ij} + W_j$$

$$x_j = f(\bar{X}_j) = \frac{1}{1 + \exp(-\bar{X}_j)}$$

Where

x_i is the output of node i located in any one of the pervious layers,

W_{ij} is the weight associated with the link connecting nodes i and j,

W_j is the bias of node j.

- Internal parameters associated with each node j is the weight W_{ij}. So changing the weights of the node will change the behaviour of the whole backpropagation MLP.
- Fig. 1.5.7 shows two layer backpropagation MLP.
- The above backpropagation MLP will refer to as a 3-4-3 network, corresponding to the number of nodes in each layer.
- The backward error propagation also known as the backpropagation (BP) or the Generalized Delta Rule (GDR). A squared error measure for the pth input-output pair is defined as

$$E_p = \sum_k (d_k - x_k)^2$$

Where d_k is the desired output for node k and x_k is the actual output for node k when the input part of the pth data pair is presented.

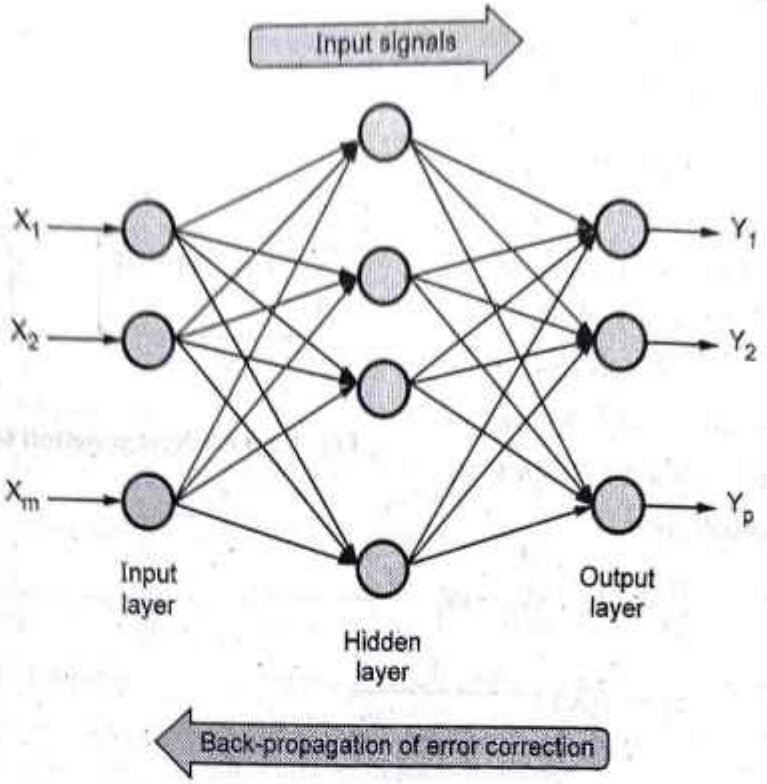


Fig. 1.5.7 Two layer backpropagation MLP

- To find the gradient vector, an error term $\bar{\epsilon}_i$ for node i is defined as :

$$\bar{\epsilon}_i = \frac{\partial + E_p}{\partial \bar{X}_i}$$

- The partial derivative can be rewritten as product of two terms using chain rule for partial differentiation :

$$\frac{\partial E(t)}{\partial W_{ij}(t)} = \frac{\partial E(t)}{\partial a_i(t)} \bullet \frac{\partial a_i(t)}{\partial w_{ij}(t)}$$

- Features of the delta rule are as follows :
  1. Simplicity
  2. Distributed learning : Learning is not reliant on central control of the network.
  3. Online learning : Weights are updated after presentation of each pattern.

Rules for Feedforward Multilayer Perceptron

- The training algorithm is called Error Back Propagation (EBP) training algorithm. If a submitted pattern provides an output far from desired value, the weights and thresholds are adjusted so that the current mean square classification error is reduced.

- The training is repeated for all patterns until the training set provide an acceptable overall error. Usually the mapping error is computed over the full training set.
- Error back propagation algorithm is working in two stages :
  1. The trained network operates feed-forward to obtain output of the network
  2. The weight adjustment propagate backward from output layer through hidden layer toward input layer.

1.5.3 Recurrent Neural Network

- A recurrent neural network is a type of neural network that contains loops, allowing information to be stored within the network.
- A RNN is particularly useful when a sequence of data is being processed to make a classification decision or regression estimate but it can also be used on non-sequential data. Recurrent neural networks are typically used to solve tasks related to time series data.
- Applications of recurrent neural networks include natural language processing, speech recognition, machine translation, character-level language modeling, image classification, image captioning, stock prediction, and financial engineering.
- Fig. 1.5.8 shows architecture of recurrent neural network.

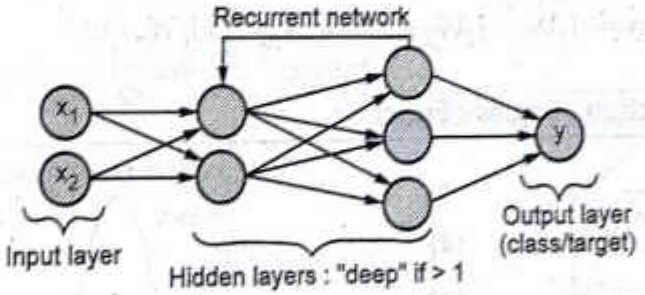


Fig. 1.5.8 Architecture of RNN

- Recurrent Neural Networks can be thought of as a series of networks linked together. They often have a chain-like architecture, making them applicable for tasks such as speech recognition, language translation, etc.
- An RNN can be designed to operate across sequences of vectors in the input, output, or both. For example, a sequenced input may take a sentence as an input and output a positive or negative sentiment value. Alternatively, a sequenced output may take an image as an input and produce a sentence as an output.

1.6 Activation Functions

- Activation functions also known as transfer function is used to map input nodes to output nodes in certain fashion.
- The activation function is the most important factor in a neural network which decided whether or not a neuron will be activated or not and transferred to the next layer.
- Activation functions help in normalizing the output between 0 to 1 or – 1 to 1. It helps in the process of backpropagation due to their differentiable property. During backpropagation, loss function gets updated and activation function helps the gradient descent curves to achieve their local minima.
- Activation function basically decides in any neural network that given input or receiving information is relevant or it is irrelevant.
- These activation function makes the multilayer network to have greater representational power than single layer network only when non-linearity is introduced.
- The input to the activation function is sum which is defined by the following equation.

$$\text{sum} = I_1W_1 + I_2W_2 + \ldots + I_nW_n = \sum_{j=1}^{n} I_j W_j + b$$

- **Activation Function : Logistic Function**

$$f(\text{sum}) = \frac{1}{(1+e^{-s*\text{sum}})}$$

$$= (1+e^{-s\times\text{sum}})^{-2}$$

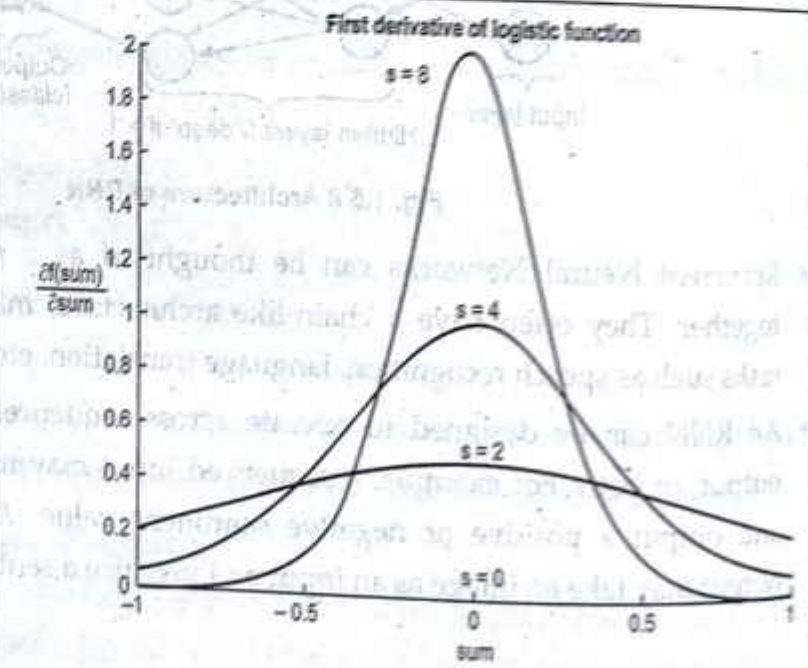


Fig. 1.6.1

- Logistic function monotonically increases from a lower limit (0 or – 1) to an upper limit (+ 1) as sum increases. In which values vary between 0 and 1, with a value of 0.5 when I is zero.

- **Activation Function : Arc Tangent**

$$f(\text{sum}) = \frac{2}{\pi}\tan^{-1}(s \times \text{sum})$$

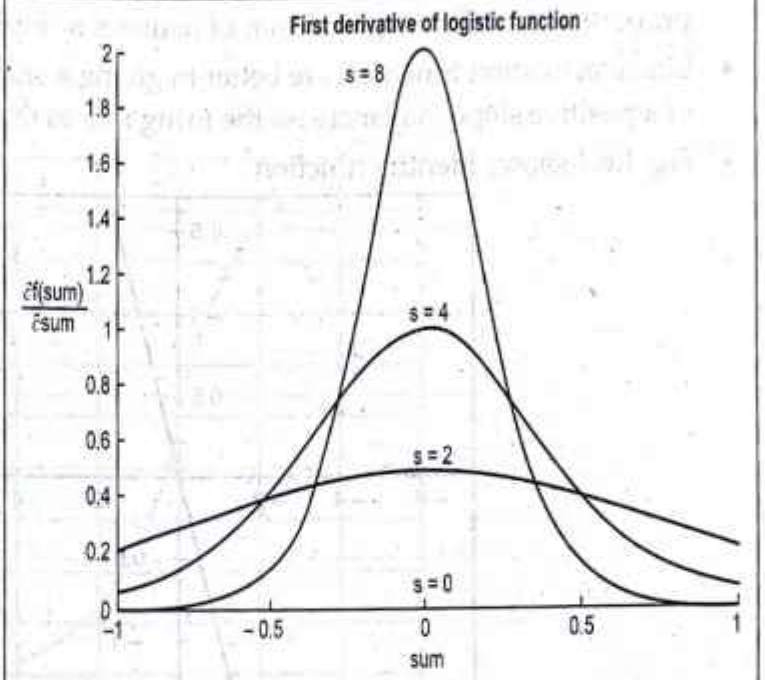


Fig. 1.6.2

- **Activation Function : Hyperbolic Tangent**

$$f(\text{sum}) = \tanh(s * I) = \frac{e^{s \times \text{sum}} - e^{-s \times \text{sum}}}{e^{s \times \text{sum}} + e^{-s \times \text{sum}}}$$

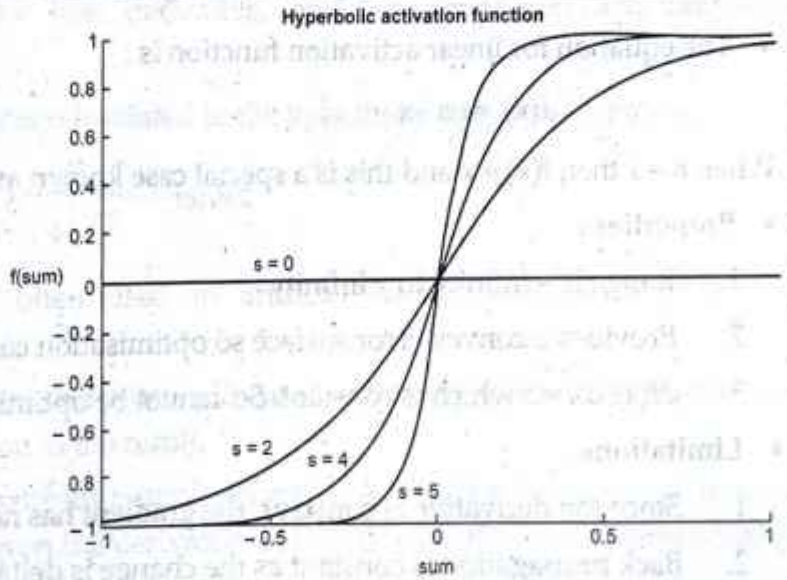


Fig. 1.6.3

1.6.1 Identity or Linear Activation Function

- A linear activation is a mathematical equation used for obtaining output vectors with specific properties.
- It is a simple straight line activation function where our function is directly proportional to the weighted sum of neurons or input.
- Linear activation functions are better in giving a wide range of activations and a line of a positive slope may increase the firing rate as the input rate increases.
- Fig. 1.6.4 shows identity function.

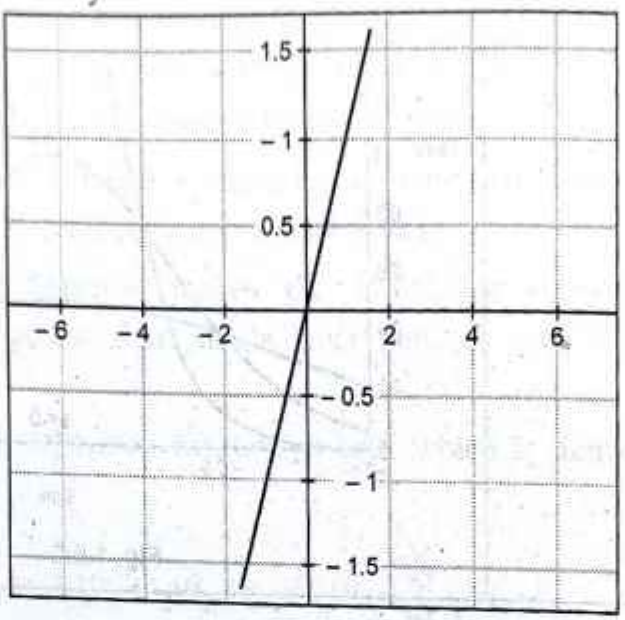


Fig. 1.6.4 Identity function

- The equation for linear activation function is :

$$f(x) = a \cdot x$$

When a = 1 then f(x) = x and this is a special case known as identity.

- **Properties :**
  1. Range is – infinity to + infinity
  2. Provides a convex error surface so optimisation can be achieved faster.
  3. df(x)/dx = a which is constant. So cannot be optimised with gradient descent.
- **Limitations :**
  1. Since the derivative is constant, the gradient has no relation with input.
  2. Back propagation is constant as the change is delta x.
  3. Activation function does not work in neural networks in practice.

1.6.2 Sigmoid

- A sigmoid function produces a curve with an "S" shape. The example sigmoid function shown on the left is a special case of the logistic function, which models the growth of some set.

$$\text{sig}(t) = \frac{1}{1+e^{-t}}$$

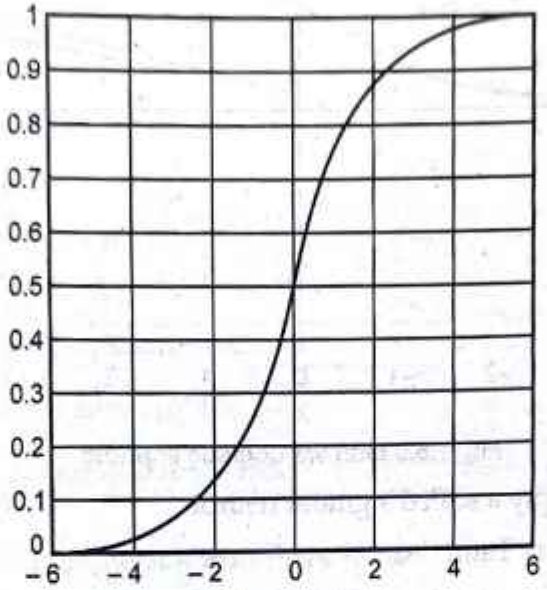


Fig. 1.6.5

- In general, a sigmoid function is real-valued and differentiable, having a non-negative or non-positive first derivative, one local minimum and one local maximum.
- The logistic sigmoid function is related to the hyperbolic tangent as follows :

$$1 - 2\text{sig}(x) = 1 - 2\frac{1}{1+e^{-x}} = -\tanh\frac{x}{2}$$

- Sigmoid functions are often used in artificial neural networks to introduce nonlinearity in the model.
- A neural network element computes a linear combination of its input signals, and applies a sigmoid function to the result.
- A reason for its popularity in neural networks is because the sigmoid function satisfies a property between the derivative and itself such that it is computationally easy to perform.

$$\frac{d}{dt}\text{sig}(t) = \text{sig}(t)(1 - \text{sig}(t))$$

- Derivatives of the sigmoid function are usually employed in learning algorithms.

1.6.3 Tanh and ReLU

- Tanh is also like logistic sigmoid but better. The range of the tanh function is from (– 1 to 1). Tanh is also sigmoidal (s - shaped).
- Fig. 1.6.6 shows tanh v/s Logistic sigmoid.

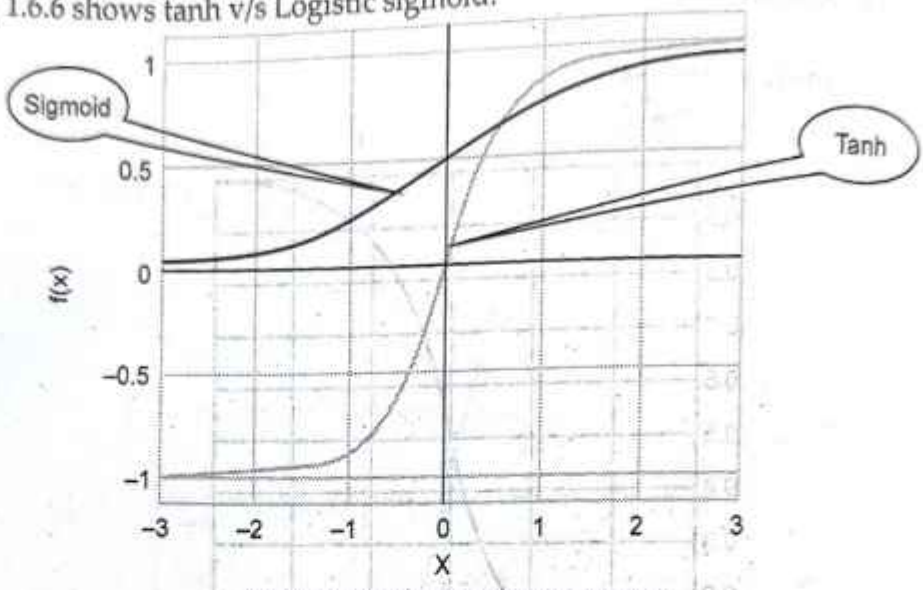


Fig. 1.6.6 tanh v/s Logistic sigmoid

- Tanh neuron is simply a scaled sigmoid neuron.
- Problems resolved by Tanh
  1. The output is not zero centered 2. Small gradient of sigmoid function
- ReLU (Rectified Linear Unit) is the most used activation function in the world right now. Since, it is used in almost all the convolution neural networks or deep learning.
- Fig. 1.6.7 shows ReLU v/s Logistic sigmoid.

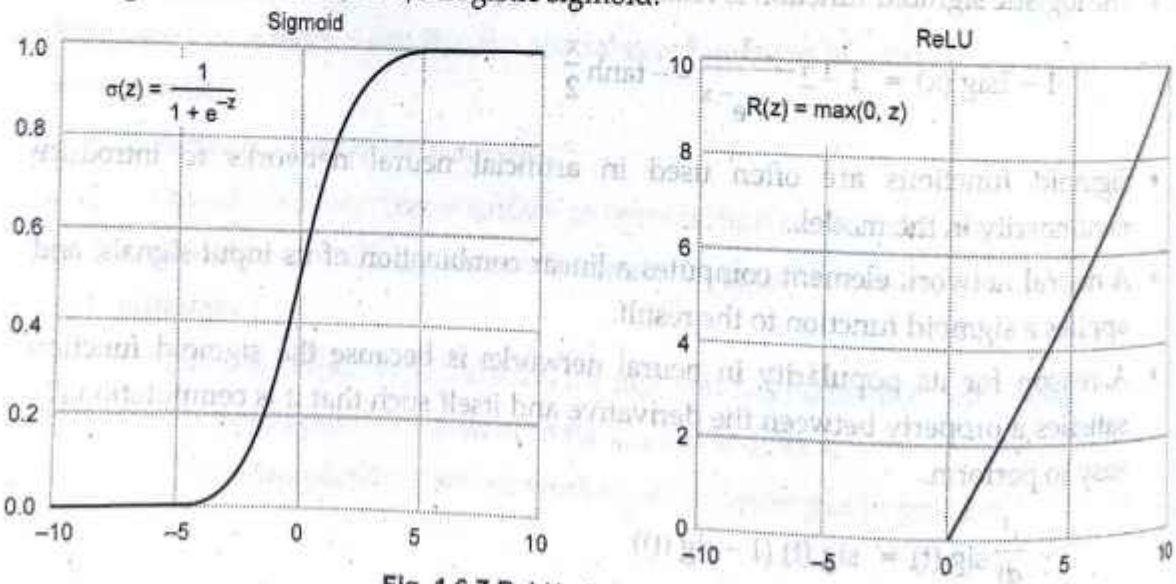


Fig. 1.6.7 ReLU v/s Logistic Sigmoid

- As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.
- Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLUcan be implemented by simply thresholding a matrix of activations at zero.

Function	Advantages	Disadvantages
Sigmoid	1. Output in range (0,1)	1. Saturated neurons 2. Not zero entered 3. Small gradient 4. Vanishing gradient
Tanh	1. Zero centered, 2. Output in range (– 1,1)	1. Saturated neurons
ReLU	1. Computational efficiency, 2. Accelerated convergence	1. Dead neurons, 2. Not zero centered

1.7 Models of Neuron-McCulloch and Pitts Model

- The first mathematical model of a biological neuron was presented by McCulloch and Pitts. This model is known as McCulloch Pitt model. It is basic building block of neural network.
- Directed weight graph is used for connecting neurons.
- McCulloch and Pitts describe a neuron as a logical threshold element with two possible states. Such a threshold element has "N" input channels and one output channel. An input channel is either active (input 1) or silent (input 0).
- The activity states of all input channels thus encode the input information as a binary sequence of N bits. The state of the threshold element is then given by linear summation of all a different input signals x_i and comparison of the sum with a threshold value s.
- The system of neurons is static and acts synchronously. A processor (system) with multiple inputs and a single output.
- Effective input : Weighted sum of all inputs.
- Bias or threshold : If the effective input is larger than the bias, the neuron outputs a one, otherwise, it outputs a zero.

- Fig. 1.7.1 shows McCulloch Pitt model.

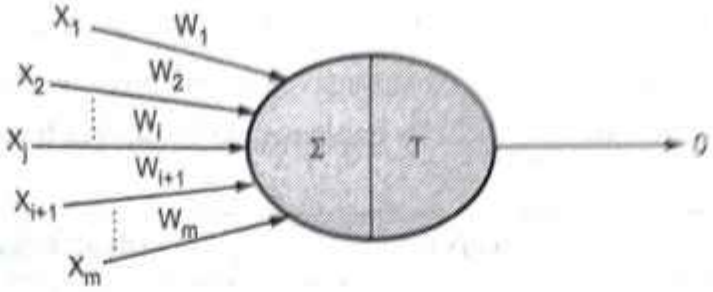


Fig. 1.7.1 McCulloch Pitt model

- This model can be described in a mathematical formalism as follows :

$$0 = \theta(a)$$

Where

$$a = \sum W_j X_j - T$$

And

$\theta(x)$ is a function such that $\theta(x) = 1$ if $x > 0$, otherwise $\theta(x) = 1$.

- The parameters used to scale the inputs are called the weights. The effective input is the weighted sum of the inputs. The parameter to measure the switching level is the threshold or bias. Neuron fires (output of one) when its net input excitation exceeds a certain value called 'threshold.' Threshold is the minimum value of the sum of the weighted active inputs needed for the postsynaptic neuron to fire.
- The function for producing the final output is called the activation function, which is the step function in the McCulloch-Pitts model.

$$0 = f\left(\sum_{j=1}^{N} W_j X_j - T\right)$$

$$f(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Their "neurons" operated under the following assumptions :

1. They are binary devices (0, 1) .
2. Each neuron has a fixed threshold (theta).
3. The neuron receives inputs from excitatory synapses, all having identical weights.
4. Inhibitory inputs have an absolute veto power over any excitatory inputs.

5. At each time step the neurons are simultaneously (synchronously) updated by summing the weighted excitatory inputs and setting the output to 1 iff the sum is greater than or equal to the threshold AND if the neuron receives no inhibitory input.

- In general, there are many different kinds of activation functions. The step function used in the McCulloch-Pitts model is simply one of them. Because the activation function takes only two values, this model is called discrete neuron.
- To make the neuron learnable, some kind of continuous function is often used as the activation function. This kind of neurons is called continuous neurons. Typical functions used in an artificial neuron are sigmoid functions, radial basis function, sinusoidal functions, etc.

Problems with McCulloch-Pitts neurons

1. Weights and thresholds are analytically determined. We cannot learn.
2. It is very difficult to minimize size of a network.

1.8 Perceptron

- The perceptron is a feed-forward network with one output neuron that learns a separating hyper-plane in a pattern space.
- The "n" linear Fx neurons feed forward to one threshold output Fy neuron. The perceptron separates linearly separable set of patterns.

1.8.1 Single Layer Perceptron

- The perceptron is a feed-forward network with one output neuron that learns a separating hyper-plane in a pattern space. The "n" linear Fx neurons feed forward to one threshold output Fy neuron. The perceptron separates linearly separable set of patterns.
- SLP is the simplest type of artificial neural networks and can only classify linearly separable cases with a binary target (1, 0).
- We can connect any number of McCulloch-Pitts neurons together in any way we like. An arrangement of one input layer of McCulloch-Pitts neurons feeding forward to one output layer of McCulloch-Pitts neurons is known as a **Perceptron**.
- A single layer feed-forward network consists of one or more output neurons, each of which is connected with a weighting factor W_{ij} to all of the inputs X_i.
- The Perceptron is a kind of a single-layer artificial network with only one neuron. The Perception is a network in which the neuron unit calculates the linear

combination of its real-valued or boolean inputs and passes it through a threshold activation function. Fig. 1.8.1 shows Perceptron.

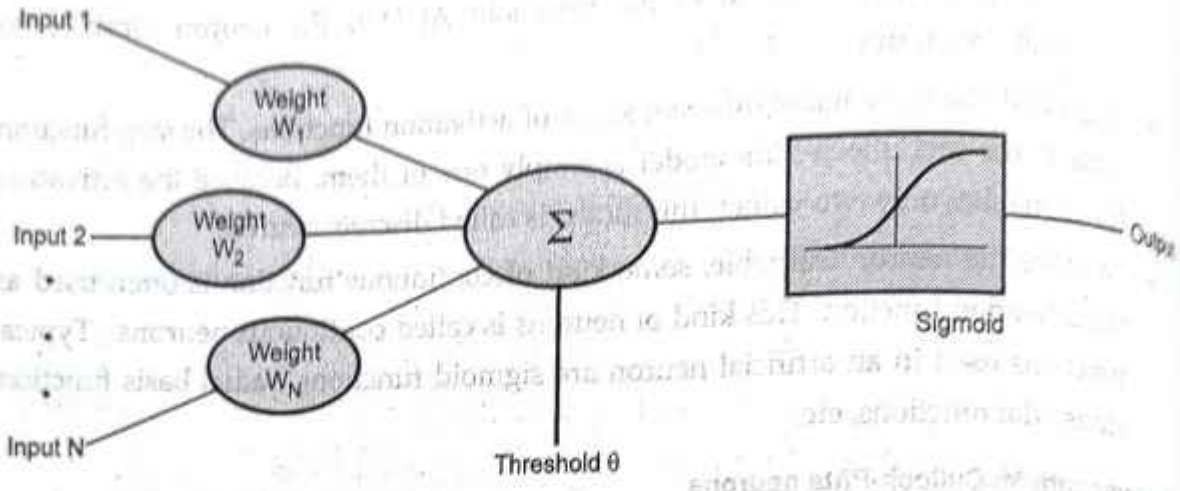


Fig. 1.8.1 Perceptron

- The Perceptron is sometimes referred to a Threshold Logic Unit (TLU) since it discriminates the data depending on whether the sum is greater than the threshold value.
- In the simplest case the network has only two inputs and a single output. The output of the neuron is :

$$y = f\left(\sum_{i=1}^{2} W_i X_i + b\right)$$

- Suppose that the activation function is a threshold then

$$f = \begin{cases} 1 & \text{if } s > 0 \\ -1 & \text{if } s \le 0 \end{cases}$$

- The Perceptron can represent most of the primitive boolean functions : AND, OR, NAND and NOR but can not represent XOR.
- In single layer perceptron, initial weight values are assigned randomly because it does not have previous knowledge. It sum all the weighted inputs. If the sum is greater than the threshold value then it is activated i.e. output = 1.

Output

$$W_1X_1 + W_2X_2 + \ldots + W_nX_n > \theta \Rightarrow 1$$

$$W_1X_1 + W_2X_2 + \ldots + W_nX_n \le \theta \Rightarrow 0$$

- The input values are presented to the perceptron, and if the predicted output is the same as the desired output, then the performance is considered satisfactory and no changes to the weights are made.
- If the output does not match the desired output, then the weights need to be changed to reduce the error.
- The weight adjustment is done as follows :

$$\Delta W = \eta \times d \times x$$

Where

x = Input data

d = Predicted output and desired output

η = Learning rate

- If the output of the perceptron is correct then we do not take any action. If the output is incorrect then the weight vector is $W \rightarrow W + \Delta W$.
- The process of weight adaptation is called **learning**.
- Perceptron Learning Algorithm :

1. Select random sample from training set as input.
2. If classification is correct, do nothing.
3. If classification is incorrect, modify the weight vector W using

$$W_i = W_i + \eta_d (n) X_i (n)$$

Repeat this procedure until the entire training set is classified correctly.

1.8.2 Multilayer Perceptron

- The perceptron is very useful for classifying data sets that are linearly separable.
- The Multilayer Perceptron (MLP) model features multiple layers that are interconnected in such a way that they form a feed-forward neural network. Each neuron in one layer has directed connections to the neurons of a separate layer.
- It consists of three types of layers: the input layer, output layer and hidden layer.
- Fig. 1.8.2 shows multilayer perceptron model.
- The **input layer** receives the input signal to be processed. The input layer distributes the values to each of the neurons in the hidden layer. In addition to the predictor variables, there is a constant input of 1.0, called the bias that is fed to each of the hidden layers; the bias is multiplied by a weight and added to the sum going into the neuron.

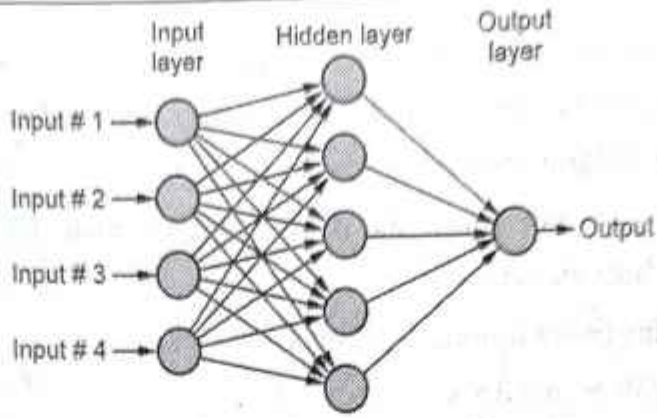


Fig. 1.8.2 Multilayer perceptron model

- **Hidden layer :** Arriving at a neuron in the hidden layer, the value from each input neuron is multiplied by a weight and the resulting weighted values are added together producing a combined value. The weighted sum is fed into a transfer function, which outputs a value. The outputs from the hidden layer are distributed to the output layer.
- **Output layer :** Arriving at a neuron in the output layer, the value from each hidden layer neuron is multiplied by a weight, and the resulting weighted values are added together producing a combined value. The weighted sum is fed into a transfer function, which outputs a value. The output values are the outputs of the network.
- The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP.
- The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. The major use cases of MLP are pattern classification, recognition, prediction and approximation.
- The perceptron is very useful for classifying data sets that are linearly separable. They encounter serious limitations with data sets that do not conform to this pattern as discovered with the XOR problem. The XOR problem shows that for any classification of four points that there exists a set that are not linearly separable.
- The MultiLayer Perceptron breaks this restriction and classifies datasets which are not linearly separable. They do this by using a more robust and complex architecture to learn regression and classification models for difficult datasets.

- **Deciding how many neurons to use in the hidden layers :**

  a) One of the most important characteristics of a perceptron network is the number of neurons in the hidden layer(s). If an inadequate number of neurons are used, the network will be unable to model complex data, and the resulting fit will be poor.

  b) If too many neurons are used, the training time may become excessively long, and, worse, the network may over fit the data. When overfitting occurs, the network will begin to model random noise in the data. The result is that the model fits the training data extremely well, but it generalizes poorly to new, unseen data. Validation must be used to test for this.

1.8.3 ADALINE Network

- ADALINE (Adaptive Linear Neuron) is an early single-layer artificial neural network.
- An important generalized of the perceptrons training algorithm was presented by Widrow and Hoff as the least mean square learning procedure also known as the delta rule.
- The learning rule was applied to the "**adaptive linear element**" also named Adaline.
- The perceptron learning rule uses the output of the thersold function for learning. The delta rule uses the net output without further mapping into output values – 1 or + 1.
- Fig. 1.8.3 shows adaline.

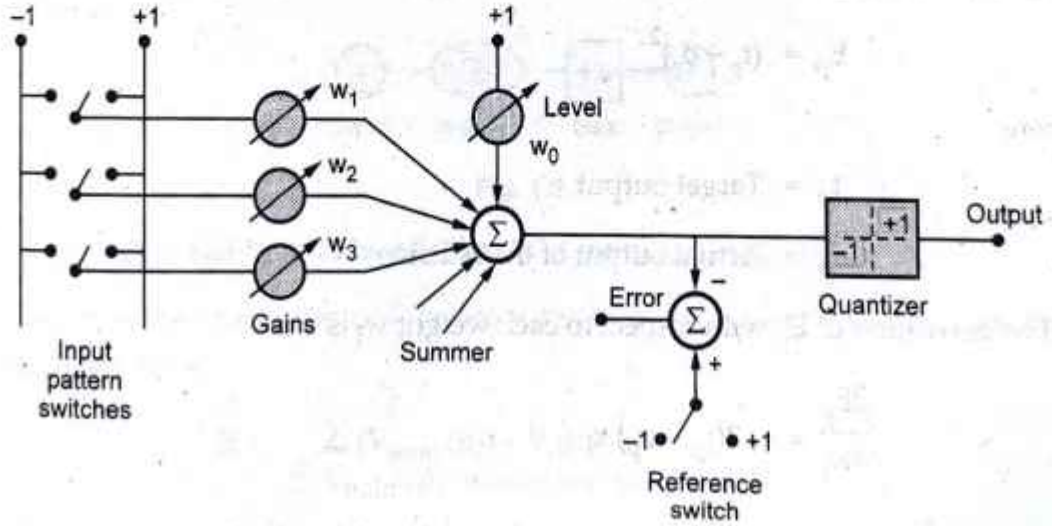


Fig. 1.8.3 Adaline

If the input conductances are denoted by w_i where $m_i = 0, 1, 2, \ldots, n$ and input and output signals by x_i and y respectively, then the output of the central block is defined to be :

$$y = \sum_{i=1}^{n} w_i x_i + \theta$$

Where, $\theta \equiv w_0$

- In a simple physical implementation, this device consists of a set of controllable resistors connected to a circuit which can sum up currents caused by the input voltage signals. Usually the central block, the summer is also followed by a quantizer which outputs + 1 of – 1, depending on the polarity of the sum.
- The problem is to determine the coefficients w_i, where i = 0, 1 ..., n, in such way that the input output response is correct for a large number of arbitrarily chosen signal sets.
- If an exact mapping is not possible the average error must be minimized, for instance, in the sense of least squares.
- An adaptive operation means that there exists a mechanism by which the w_i can be adjusted, usually iteratively to attain the correct values.
- For the Adaline, Widrow introduced the delta rule to adjust the weights.
- For the pth input-output pattern, the error measure of a single-output Adaline can be expressed as,

$$E_p = (t_p - o_p)^2$$

Where

t_p = Target output

o_p = Actual output of the Adaline

- The derivation of E_p with respect to each weight w_i is

$$\frac{\partial E_p}{\partial w_i} = -2(t_p - o_p)\, x_i$$

- To decrease E_p by gradient descent, the update formula for w_i on the pth input output pattern is

$$\Delta_p w_i = \eta(t_p - o_p) x_i$$

- The delta rule tries to minimize squared errors, it is also referred to as the least mean square learning procedure or Widrow - Hoff learning rule.

1.9 Basic Learning Laws

By learning rule, we mean a procedure for modifying the weights and biases of a network. The purpose of the learning rule is to train the network to perform some tasks. There are many types of neural network learning rules. They fall into three broad categories : Supervised learning, Unsupervised learning and Reinforcement learning.

1.9.1 Weights and Bias

- Weights and biases are the learnable parameters that help a neural network correctly learn a function. A machine learning model uses lots of examples to learn the correct weights and bias to assign to each feature in a dataset to help it correctly predict outputs.
- Weight is the parameter within a neural network that transforms input data within the network's hidden layers. A neural network is a series of nodes, or neurons. Within each node is a set of inputs, weight and a bias value.
- As an input enters the node, it gets multiplied by a weight value and the resulting output is either observed or passed to the next layer in the neural network. Often the weights of a neural network are contained within the hidden layers of the network.

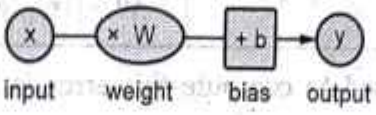


Fig. 1.9.1

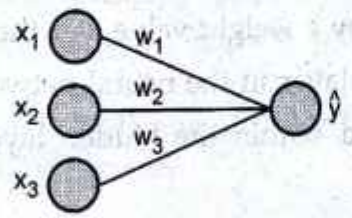
- Choose the weights w_i to best fit the set of training examples.
- Minimize the squared error E between the train values and the values predicted by the hypothesis

$$E = \sum_{(b,\ V_{train}(b)) \in \text{ training examples}} (V_{train}(b)) - \hat{V}((b))^2$$

- Require an algorithm that will incrementally refine weights as new training values.
- Least Mean Squares (LMS) is one such algorithm.

Weight vs. Bias

- A teachable neural network will randomize both the weight and bias values before learning initially begins. As training continues, both parameters are adjusted toward the desired values and the correct output.
- The two parameters differ in the extent of their influence upon the input data. Simply, bias represents how far off the predictions are from their intended value.
- Biases make up the difference between the function's output and its intended output. A low bias suggest that the network is making more assumptions about the form of the output, whereas a high bias value makes less assumptions about the form of the output.
- Weights, on the other hand, can be thought of as the strength of the connection. Weight affects the amount of influence a change in the input will have upon the output. A low weight value will have no change on the input and alternatively a larger weight value will more significantly change the output.
- In most learning networks, error is calculated as the difference between the actual output y and the predicted output y ? Loss functions are quantitative measures of how satisfactory the model predictions are.



Output :

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b = w^T x + b$$

Neural network parameters

$$W = \{ | w_1, w_2, w_3 | b \}$$

- The function that is used to compute this error is known as Loss Function also known as Cost function.

Mean Squared Error (MSE)

- The mean squared error calculates the average squared error between the outputs from the neural network and the targets in the data set.
- Mean Squared Error is one of the most common loss functions. MSE loss function is widely used in linear regression as the performance measure

$$\text{Mean Squared Error} = \frac{\sum (\text{Outputs} - \text{Targets})^2}{\text{Samples Number}}$$

Example 1.9.1 *For a given data having 100 examples, if saturated errors SE_1, SE_2 and SE_3 are 13.3, 3.33 and 4.00 respectively. Calculate Mean Squared Error (MSE). State the formula for MSE.*

Solution :

$$\text{Mean Squared Error} = \frac{\text{Squared Error}_1 + \text{Squared Error}_2 + \ldots + \text{Squared Error}_N}{\text{Number of data samples}}$$

$$= \frac{13.3 + 3.33 + 4.00}{100} = 0.2066$$

❑❑❑

Unit II

2 Learning Algorithms

Syllabus

Learning and Memory, Learning Algorithms, Numbers of hidden nodes, Error Correction and Gradient Decent Rules, Perceptron Learning Algorithms, Supervised Learning Backpropagation, MultilayeredNetwork Architectures, Back propagation Learning Algorithm, Feed forward and feedback neural networks,example and applications.

Contents

2.1 Learning and Memory

2.2 Hebbian Learning

2.3 Competitive Learning

2.4 Error Correction

2.5 Gradient Decent Rules

2.6 Supervised Learning

2.7 Backpropagation

2.8 Feed Forward Neural Networks

2.1 Learning and Memory

- One of the primary functions of the brain is associative memory. Learning can be considered as a process of forming associations between related patterns. The associative memory is composed of a cluster of units which represent a simple model of a real biological neuron.
- An associative memory, also known as Content-Addressable Memory (CAM) can be searched for a value in a single memory cycle rather than using a software loop.
- Associative memories can be implemented using networks with or without feedback. Such associative neural networks are used to associate one set of vectors with another set of vectors, say input and output patterns.
- The aim of an associative memory is, to produce the associated output pattern whenever one of the input patterns is applied to the neural network. The input pattern may be applied to the network either as input or as initial state and the output pattern is observed at the outputs of some neurons constituting the network.
- Associative memories belong to class of neural network that learn according to a certain recording algorithm. They require information a priori and their connectivity matrices most often need to be formed in advance. Writing into memory produces changes in the neural interconnections. Reading of the stored info from memory named recall, is a transformation of input signals by the network.
- All memory information is spatially distributed throughout the network. Associative memory enables a parallel search within a stored data. The purpose of search is to output one or all stored items that matches the search argument and retrieve it entirely or partially.
- The Fig. 2.1.1 shows a block diagram of an associative memory.

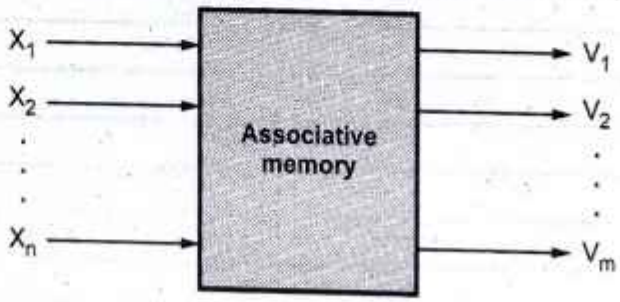


Fig. 2.1.1 Block diagram of an associative memory

- In the initialization phase of the associative memory no information is stored; because the information is represented in the w weights they are all set to zero.

- The advantage of neural associative memories over other pattern storage algorithms like lookup tables of hash codes is that the memory access can be fault tolerant with respect to variation of the input pattern.
- In associative memories many associations can be stored at the same time. There are different schemes of superposition of the memory traces formed by the different associations. The superposition can be simple linear addition of the synaptic changes required for each association (like in the Hopfield model) or nonlinear.
- The performance of neural associative memories is usually measured by a quantity called information capacity, that is, the information content that can be learned and retrieved, divided by the number of synapses required.
- An associative memory is a content-addressable structure that maps specific input representations to specific output representations. It is a system that "associates" two patterns (X, Y) such that when one is encountered, the other can be recalled.
- Associative network memory can be static or dynamic.
  - **Static :** Networks recall an output response after an input has been applied in one feed-forward pass and theoretically without delay. They were termed instantaneous.
  - **Dynamic :** Memory networks produce recall as a result of output/input feedback interaction, which requires time.
- There are two classes of associative memory : Auto-associative and hetero-associative.
- Whether auto-or hetero-associative, the net can associate not only the exact pattern pairs used in training, but is also able to obtain associations if the input is similar to one on which it has been trained.

2.1.1 Auto-associative Memory

- Auto-associative networks are a special subset of the hetero-associative networks, in which each vector is associated with itself, i.e. $y^i = x^i$ for i = 1, ..., m. The function of such networks is to correct noisy input vectors.
- Fig. 2.1.2 shows auto-associative memory.
- Auto-associative memories are content based memories which can recall a stored sequence when they are presented with a fragment or a noisy version of it. They are very effective in de-noising the input or removing interference from the input which makes them a promising first step in solving the cocktail party problem.
- The simplest version of auto-associative memory is linear associator which is a two-layer feed-forward fully connected neural network where the output is constructed in a single feed-forward computation.

Fig. 2.1.2 Auto-associative memory

- Artificial neural networks can be used as associative memories. One of the simplest artificial neural associative memory is the linear associator. The Hopfield model and Bidirectional Associative Memory (BAM) models are some of the other popular artificial neural network models used as associative memories

2.1.2 Hetero-associative Memory Network

- Hetero-associative networks map "m" input vectors $X^1, X^2, \ldots, X^m$ in n-dimensional space to m output vectors $y^1, y^2, \ldots, y^m$ in k-dimensional space, so that $X^i \rightarrow y^i$.
- If $\|\tilde{X} - X^i\| < \epsilon$ then $\tilde{x} - y^i$. This should be achieved by the learning algorithm, but becomes very hard when the number m of vectors to be learned is too high.
- Fig. 2.1.3 shows block diagram of hetero-associative network.

Fig. 2.1.3 Auto-associative memory

- Fig. 2.1.4 shows the structure of a hetero-associative network without feedback.

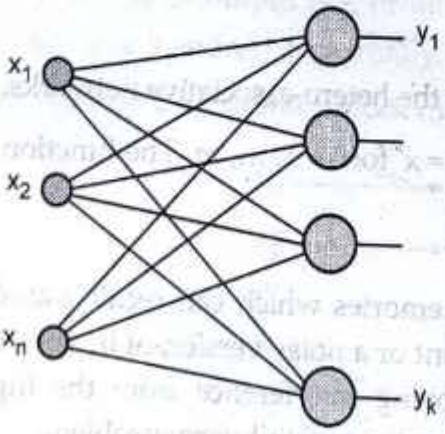


Fig. 2.1.4 Hetero-associative network without feedback

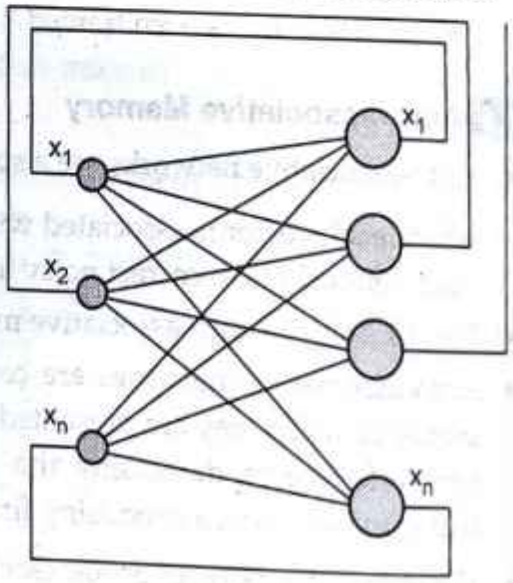


Fig. 2.1.5 Hetero-associative network with feedback

2.1.3 The Hopfield Network

- The **Hopfield** model is a single-layered recurrent network. Like the associative memory, it is usually initialized with appropriate weights instead of being trained.
- Hopfield Neural Network (HNN) is a model of auto-associative memory. It is a single layer neural network with feedbacks. Fig. 2.1.6 shows Hopfield network of three units. The Hopfield network is created by supplying input data vectors, or pattern vectors, corresponding to the different classes. These patterns are called class patterns.

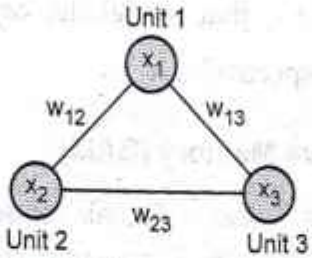


Fig. 2.1.6 Hopfield network of three units

- Hopfield model consists of a single layer of processing elements where each unit is connected to every other unit in the network other than itself.
- The output of each neuron is a binary number in {– 1, 1}. The output vector is the state vector. Starting from an initial state (given as the input vector), the state of the network changes from one to another like an automaton. If the state converges, the point to which it converges is called the attractor.
- In its simplest form, the output function is the sign function, which yields 1 for arguments ≥ 0 and – 1 otherwise.
- The connection weight matrix W of this type of network is square and symmetric. The units in the Hopfield model act as both input and output units.
- A Hopfield network consists of "n" totally coupled units. Each unit is connected to all other units except itself. The network is symmetric because the weight w_{ij} for the connection between unit i and unit j is equal to the weight w_{ij} of the connection from unit j to unit i. The absence of a connection from each unit to itself avoids a permanent feedback of its own state value.
- Hopfield networks are typically used for classification problems with binary pattern vectors.
- Hopfield model is classified into two categories :
  1. Discrete Hopfield Model
  2. Continuous Hopfield Model

- In both discrete and continuous Hopfield network weights trained in a one-shot fashion and not trained incrementally as was done in case of Perceptron and MLP.
- In the discrete Hopfield model, the units use a slightly modified bipolar output function where the states of the units, i.e., the output of the units remain the same if the current state is equal to some threshold value.
- The continuous Hopfield model is just a generalization of the discrete case. Here, the units use a continuous output function such as the sigmoid or hyperbolic tangent function. In the continuous Hopfield model, each unit has an associated capacitor C_i and resistance r_i that model the capacitance and resistance of real neuron's cell membrane, respectively.

2.1.4 Bidirectional Associative Memory (BAM)

- BAM consists of two layers, x and y. Signals are sent back and forth between both layers until an equilibrium is reached. Equilibrium is reached if the x and y vectors no longer change. Given an x vector the BAM is able to produce the y vector and vice versa.
- BAM consists of bi-directional edges so that information can flow in either direction. Since the BAM network has bidirectional edges, propagation moves in both directions, first from one layer to another and then back to the first layer. Propagation continues until the nodes are no longer changing values.
- Fig. 2.1.7 shows BAM network.

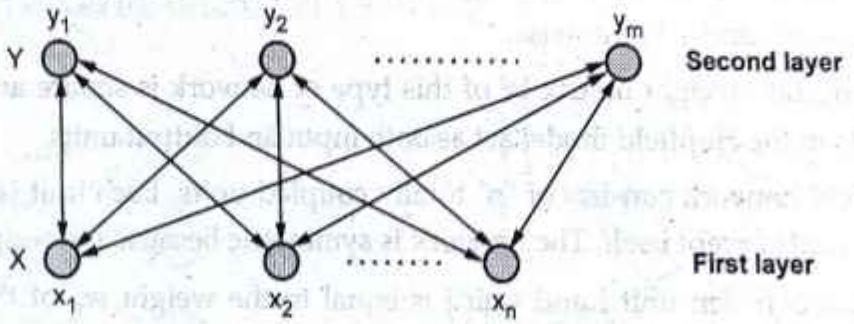


Fig. 2.1.7 BAM network

- Since the BAM also uses the traditional Hebb's learning rule to build the connection weight matrix to store the associated pattern pairs, it too has a severely low memory capacity.

- BAM can be classified into two categories :
  1. **Discrete BAM :** The network propagates an input pattern X to the Y layer where the units in the Y layer will compute their net input.
  2. **Continuous BAM :** The units use the sigmoid or hyperbolic tangent output function. The units in the X layer have an extra external input I_i, while the units in the Y layer have an extra external input J_j for i = 1, 2, ..., m and j = 1, 2, ..., n.
- These extra external inputs lead to a modification in the computation of the net input to the units.

2.1.5 Difference between Auto-associative Memory and Hetero-Associative Memory

Auto-associative memory	Hetero-associative memory
The inputs and output vectors s and t are the same.	The inputs and output vectors s and t are different.
Recalls a memory of the same modality as the one that evoked it.	Recalls a memory that is different in character from the input.
A picture of a favorite object might evoke a mental image of that object in vivid detail.	A particular smell or sound, for example, might evoke a visual memory of some past event.
An auto-associative memory retrieves the same pattern.	Hetero-associative memory retrieves the stored pattern.
Example : color correction, color constancy	Example : 1. Space transforms : Fourier, 2. Dimensionality reduction : PCA

2.2 Hebbian Learning

- In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as **Hebb's Law**. Hebb's Law states that if neuron i is near enough to excite neuron j and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron j becomes more sensitive to stimuli from neuron i.
- Hebb's Law can be represented in the form of two rules :
  1. If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.
  2. If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

- Hebb's Law provides the basis for learning without a teacher. Learning here is a **local phenomenon** occurring without feedback from the environment.
- Using Hebb's Law we can express the adjustment applied to the weight w_{ij} at iteration p in the following form :

$$\Delta w_{ij}(p) = F[y_i(p), x_i(p)]$$

- As a special case, we can represent Hebb's Law as follows :

$$\Delta w_{ij}(p) = \alpha\, y_i(p)\, x_i(p)$$

where α is the learning rate parameter. This equation is referred to as the **activity product rule**.

- Hebbian learning implies that weights can only increase. To resolve this problem, we might impose a limit on the growth of synaptic weights. It can be done by introducing a non-linear **forgetting factor** into Hebb's Law :

$$\Delta w_{ij}(p) = \alpha\, y_i(p)\, x_i(p) - \varphi\, y_i(p)\, w_{ij}(p)$$

where φ is the forgetting factor.

- Forgetting factor usually falls in the interval between 0 and 1, typically between 0.01 and 0.1, to allow only a little "forgetting" while limiting the weight growth.

Hebbian learning algorithm

Step 1 : Initialisation

Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].

Step 2 : Activation

Compute the neuron output at iteration p

$$y_j(p) = \sum_{i=1}^{n} x_i(p)\, w_{ij}(p) - \theta_j$$

where n is the number of neuron inputs, and is the threshold value of neuron j.

Step 3 : Learning

Update the weights in the network :

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where $w_{ij}(p)$ is the weight correction at iteration p. The weight correction is determined by the generalised activity product rule :

$$w_{ij}(p) = \varphi\, y_i(p)\, [\lambda\, x_i(p) - w_{ij}(p)]$$

Step 4 : Iteration

Increase iteration p by one, go back to Step 2.

Hebbian learning example

- To illustrate Hebbian learning, consider a fully connected feed forward network with a single layer of five computation neurons. Each neuron is represented by a McCulloch and Pitts model with the sign activation function. The network is trained on the following set of input vectors :

$$x_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad x_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad x_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad x_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Initial and final weight matrices

$$\begin{array}{cc} & \text{Output layer} \\ & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \text{Input layer} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{cc} & \text{Output layer} \\ & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \text{Input layer} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \end{array}$$

- A test input vector, or probe, is defined as

$$X = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- When this probe is presented to the network, we obtain :

$$Y = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \right\} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.4940 \\ 0.2661 \\ 0.0907 \\ 0.9478 \\ 0.0737 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

2.3 Competitive Learning

- Fig. 2.3.1 shows basic architecture of a competitive learning system.

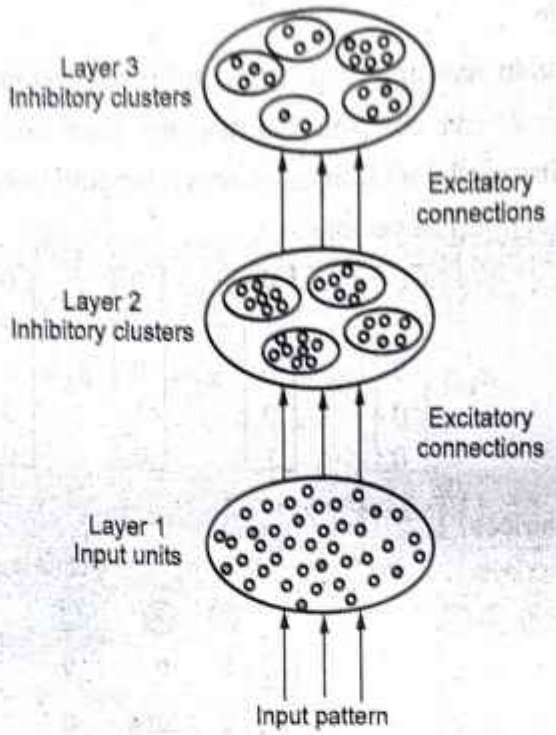


Fig. 2.3.1 Competitive learning system

- It consists of a set of hierarchically layered units in which each layer connects, via excitatory connections, with the layer immediately above it, and has inhibitory connections to units in its own layer.
- In the most general case, each unit in a layer receives an input from each unit in the layer immediately below it and projects to each unit in the layer immediately above it.
- Moreover, within a layer, the units are broken into a set of inhibitory clusters in which all elements within a cluster inhibit all other elements in the cluster.
- Thus the elements within a cluster at one level compete with one another to respond to the pattern appearing on the layer below. The more strongly any particular unit responds to an incoming stimulus, the more it shuts down the other members of its cluster.
- Units are represented in the diagram as dots. Units may be active or inactive. Active units are represented by filled dots, inactive ones by open dots.
- **Properties :**

  1. The units in a given layer are broken into several sets of non-overlapping clusters. Each unit within a cluster inhibits every other unit within a cluster.

Within each cluster, the unit receiving the largest input achieves its maximum value while all other units in the cluster are pushed to their minimum value 1. We have arbitrarily set the maximum value to 1 and the minimum value to 0.

2. Every unit in every cluster receives inputs from all members of the same set of input units.
3. A unit learns if and only if it wins the competition with other units in its cluster.
4. A stimulus pattern Sj consists of a binary pattern in which each element of the pattern is either active or inactive. An active element is assigned the value 1 and an inactive element is assigned the value 0.
5. Each unit has a fixed amount of weight (all weights are positive) that is distributed among its input lines.

- There are several characteristics of a competitive learning mechanism :
  1. Each cluster classifies the stimulus set into M groups, one for each unit in the cluster. Each of the units captures roughly an equal number of stimulus patterns.
  2. If there is structure in the stimulus patterns, the units will break up the patterns along structurally relevant lines.
  3. If the stimuli are highly structured, the classifications are highly stable. If the stimuli are less well structured, the classifications are more variable, and a given stimulus pattern will be responded to first by one and then by another member of the cluster.
  4. The particular grouping done by a particular cluster depends on the starting value of the weights and the sequence of stimulus patterns actually presented.
  5. To a first approximation, the system develops clusters that minimize within-cluster distance, maximize between-cluster distance, and balance the number of patterns captured by each cluster.

2.4 Error Correction

- Learning is a process where unknown ANN parameters are adapted through continuous process of stimulation from the environment. Learning is determined by the way how the change of parameters takes place. A set of rules that are solution to the learning problem is called a learning algorithm.
- Error correction belongs to the supervised learning paradigm. Fig. 2.4.1 shows error correction learning.

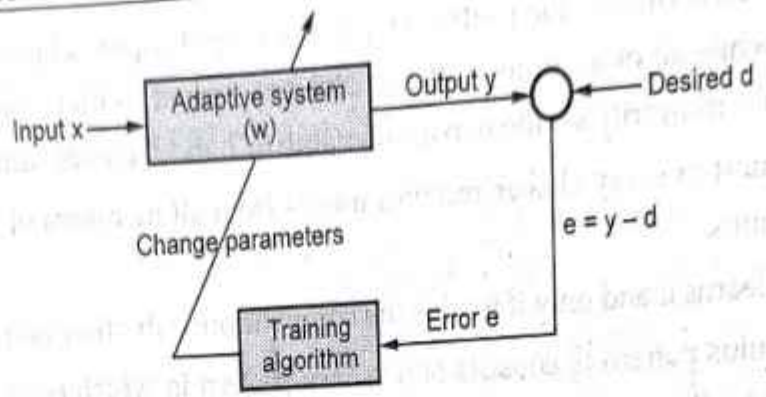


Fig. 2.4.1 Error correction learning

- Neuron (k) is driven by a signal vector x(n) produced by one or more layers of hidden neurons, which are themselves driven by an input vector applied to the source nodes of the neural network.
- Let $d_k(n)$ be desired output of neuron k at moment n
- Let $y_k(n)$ be obtained output of neuron k at moment n
- Output $y_k(n)$ is obtained using input vector x(n).
- Input vector x(n) and desired output $d_k(n)$ represent an example that is presented to ANN at moment n.
- Error is the difference between desired and obtained output of neuron k at moment n :

$$e_k(n) = d_k(n) - y_k(n)$$

- Error signal $e_k(n)$ actuates a control mechanism.
- The goal of error-correction learning is to minimize an error function derived from errors $e_k(n)$ so that the obtained output of all neurons approximates the desired output in some statistical sense. A frequently used error function is mean square error :

$$J = E\left[\frac{1}{2}\sum_k e_k^2(n)\right]$$

where E[.] is the statistical expectation operator and summation is for all neurons in the output layer.

2.5 Gradient Decent Rules

- Much of machine learning can be written as an optimization problem.
- Example loss functions : Logistic regression, linear regression, principle component analysis, neural network loss.

- A very efficient way to train logistic models is with **Stochastic Gradient Descent (SGD).**
- One challenge with training on power law data (i.e. most data) is that the terms in the gradient can have very different strengths.
- The idea behind stochastic gradient descent is iterating a weight update based on the gradient of loss function :

$$\bar{w}(k+1) = \bar{w}(k) - \gamma \nabla L(\bar{w})$$

- Logistic regression is designed as a **binary classifer** (output say {0, 1}) but actually **outputs the probability** that the input instance is in the "1" class.
- A logistic classifier has the form :

$$p(X) = \frac{1}{1 + \exp(-X\beta)}$$

where $X = (X_1, \ldots, X_n)$ is a vector of features.

- Stochastic gradient has some serious limitations however, especially if the gradients vary widely in magnitude. Some coefficients change very fast, others very slowly.
- This happens for text, user activity and social media data (and other power-law data), because gradient magnitudes scale with feature frequency, i.e. over several orders of magnitude.
- It is not possible to set a single learning rate that trains the frequent and infrequent features at the same time.
- An example of stochastic gradient descent with perceptron loss is shown as follows :

from sklearn.linear_model import SGDClassifier

2.5.1 Finding the Optimal Hyper-Parameters through Grid Search

- In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed.
- In any machine learning algorithm, these parameters need to be initialized before training a model.
- **Model hyperparameters** *are the properties that govern the entire training process.*
- Hyperparameters are important because they directly control the behaviour of the training algorithm and have a significant impact on the performance of the model is being trained.

- Choosing appropriate hyperparameters plays a crucial role in the success of our neural network architecture. Since it makes a huge impact on the learned model.
- For example, if the learning rate is too low, the model will miss the important patterns in the data. If it is high, it may have collisions.
- Choosing good hyperparameters gives two benefits :
  1. Efficiently search the space of possible hyperparameters.
  2. Easy to manage a large set of experiments for hyperparameter tuning.
- **The process of finding most optimal hyperparameters in machine learning is called hyperparameter optimisation.**
- Grid search is a very traditional technique for implementing hyperparameters. It brute force all combinations. Grid search requires to create two set of hyperparameters.

  1. Learning rate 2. Number of layers
- Grid search trains the algorithm for all combinations by using the two set of hyperparameters and measures the performance using "Cross Validation" technique.
- This validation technique gives assurance that our trained model got most of the patterns from the dataset.
- One of the best methods to do validation by using "K-Fold Cross Validation" which helps to provide ample data for training the model and ample data for validations.
- With this technique, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluating each model and selecting the architecture which produces the best results.
- For example, say you have two continuous parameters α and β, where manually selected values for the parameters are the following :

$$\alpha \in \{0, 1, 2\}$$

$$\beta \in \{.25, .50, .75\}$$

- Then the pairing of the selected hyperparametric values, H, can take on any of the following :

  $H \in \{(0, .25), (0, .50), (0, .75), (1, .25), (1, .50), (1, .75), (2, .25), (2, .50), (2, .75)\}$
- Grid search will examine each pairing of and to determine the best performing combination. The resulting pairs, *H*, are simply each output that results from taking the Cartesian product of α and β.

- While straightforward, this "brute force" approach for hyperparameter optimization has some drawbacks. Higher-dimensional hyperparametric spaces are far more time consuming to test than the simple two-dimensional problem presented here.
- Also, because there will always be a fixed number of training samples for any given model, the model's predictive power will decrease as the number of dimensions increases. This is known as Hughes phenomenon.

2.5.2 Vanishing Gradient Problem

- When back-propagation is used, the earlier layers will receive very small updates compared to the later layers. This problem is referred to as the vanishing gradient problem.
- The vanishing gradient problem is essentially a situation in which a deep multilayer feed-forward network or a Recurrent Neural Network (RNN) does not have the ability to propagate useful gradient information from the output end of the model back to the layers near the input end of the model.
- Weight initialization is one technique that can be used to solve the vanishing gradient problem. It involves artificially creating an initial value for weights in a neural network to prevent the backpropagation algorithm from assigning weights that are unrealistically small.
- The most important solution to the vanishing gradient problem is a specific type of neural network called Long Short-Term Memory Networks (LSTMs).
- Indication of vanishing gradient problem :
  a) The parameters of the higher layers change to a great extent, while the parameters of lower layers barely change.
  b) The model weights could become 0 during training.
  c) The model learns at a particularly slow pace and the training could stagnate at a very early phase after only a few iterations.
- Some methods that are proposed to overcome the vanishing gradient problem :
  a) Residual neural networks (ResNets)
  b) Multi-level hierarchy
  c) Long short term memory (LSTM)
  d) Faster hardware
  e) ReLU
  f) Batch normalization

2.6 Supervised Learning

- Supervised learning is the machine learning task of inferring a function from supervised training data. The training data consist of a set of training examples. The task of the supervised learner is to predict the output behavior of a system for any set of input values, after an initial training phase.
- Supervised learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs are usually provided by an external teacher.
- Human learning is based on the past experiences. A computer does not have experiences.
- A computer system learns from data, which represent some "past experiences" of an application domain.
- To learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not-approved and high-risk or low risk. The task is commonly called : Supervised learning, Classification or inductive learning.
- Training data includes both the input and the desired results. For some examples the correct results (targets) are known and are given in input to the model during the learning process. The construction of a proper training, validation and test set is crucial. These methods are usually fast and accurate.
- Have to be able to generalize : give the correct results when new data are given in input without knowing a priori the target.
- Supervised learning is the machine learning task of inferring a function from supervised training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object and a desired output value.
- A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a classifier or a regression function. Fig. 2.6.1 shows supervised learning process.

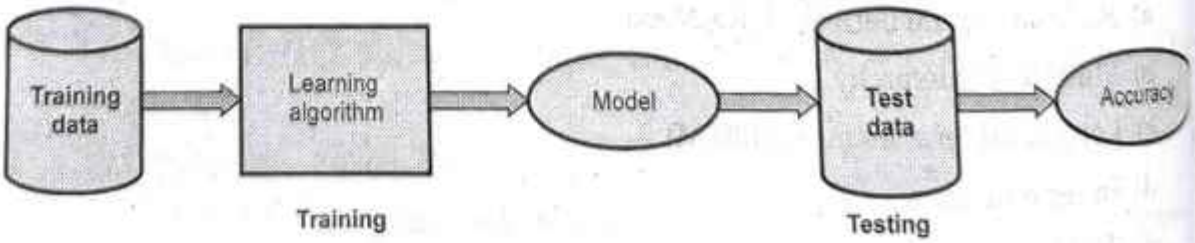


Fig. 2.6.1 Supervised learning process

- The learned model helps the system to perform task better as compared to no learning.
- Each input vector requires a corresponding target vector.

  Training Pair = (Input Vector, Target Vector)
- Fig. 2.6.2 shows input vector.
- Supervised learning denotes a method in which some input vectors are collected and presented to the network. The output computed by the net-work is observed and the deviation from the expected answer is measured. The weights are corrected according to the magnitude of the error in the way defined by the learning algorithm.

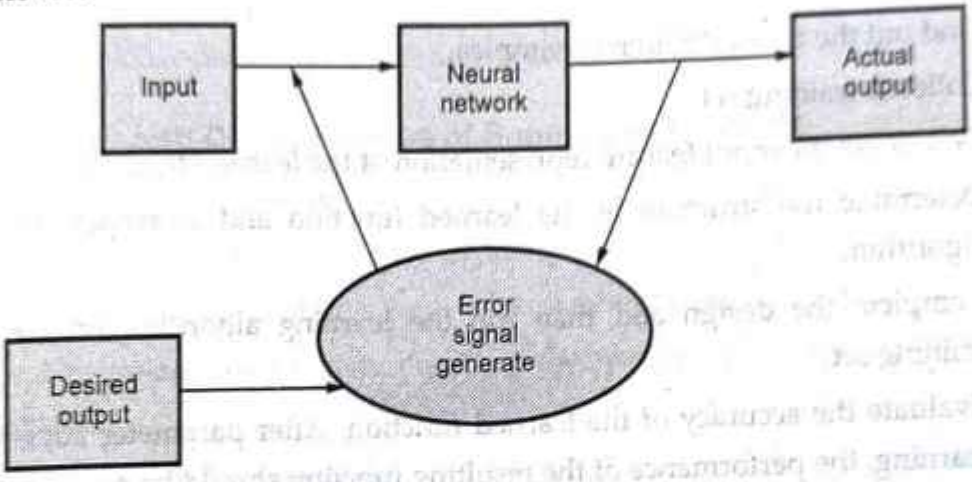


Fig. 2.6.2 Input vector

- Supervised learning is further divided into methods which use reinforcement or error correction. The perceptron learning algorithm is an example of supervised learning with reinforcement.

Data formats in supervised learning :

- Supervised learning always uses a dataset to define finite set of real vectors with m features each :

$$X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\} \text{ where } \bar{X}_i \in \mathbb{R}^m$$

- Considering that user approach is always probabilistic, we need to consider each X as drawn from a statistical multivariate distribution D. It is also useful to add an important condition upon the whole dataset X. Here we consider that all samples to be independent and identically distributed. This means all variables belong to the same distribution D and considering an arbitrary subset of m values, it happens that :

$$P(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m) = \prod_{i=1}^{m} P(\bar{x}_i)$$

- The corresponding output values can be both numerical - continuous and categorical. In the first case, the process is called regression, while in the second, it is called classification.
- Example : Dataset contains city populations by year for the past 100 years and user want to know what the population of a specific city will be four years from now. The outcome uses labels that already exist in the data set : **population, city and year.**
- In order to solve a given problem of supervised learning, following steps are performed :
  1. Find out the type of training examples.
  2. Collect a training set.
  3. Determine the input feature representation of the learned function.
  4. Determine the structure of the learned function and corresponding learning algorithm.
  5. Complete the design and then run the learning algorithm on the collected training set.
  6. Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.
- Supervised learning is divided into two types : Classification and Regression.

1. Classification :

- Classification predicts categorical labels (classes), prediction models continuous-valued functions. Classification is considered to be supervised learning.
- Classifies data based on the training set and the values in a classifying attribute and uses it in classifying new data. **Prediction** means models continuous-valued functions, i.e., predicts unknown or missing values.
- Preprocessing of the data in preparation for classification and prediction can involve data cleaning to reduce noise or handle missing values, relevance analysis to remove irrelevant or redundant attributes and data transformation, such as generalizing the data to higher level concepts or normalizing data.
- Numeric prediction is the task of predicting continuos values for given input. For example, we may wish to predict the salary of college employee with 15 years of work experience or the potential sales of a new product given its price.

- Some of the classification methods like back - propagation, support vector machines and k-nearest-neighbor classifiers can be used for prediction.

2. Regression :

- For an input x, if the output is continuous, this is called a regression problem. For example, based on historical information of demand for tooth paste in supermarket, user are asked to predict the demand for the next month.
- Regression is concerned with the prediction of continuous quantities. Linear regression is the oldest and most widely used predictive model in the field of machine learning. The goal is to minimize the sum of the squared errors to fit a straight line to a set of data points.
- Regression algorithm used in supervised learning is linear regression, Bayesian linear regression, polynomial regression, regression tree etc.

2.6.1 Advantages and Disadvantages of Supervised Learning

1. Advantages of supervised learning

- It performs classification and regression tasks.
- It allows estimating or mapping the result to a new sample.
- We have complete control over choosing the number of classes we want in the training data.

2. Disadvantages of supervised learning

- Supervised learning cannot handle all complex tasks in Machine Learning.
- Computation time is vast for supervised learning.
- It requires a labelled data set.
- It requires a training process.

2.6.2 Difference between Supervised and Unsupervised Learning

Sr. No.	Supervised learning	Unsupervised learning
1.	Desired output is given.	Desired output is not given.
2.	It is not possible to learn larger and more complex models than with supervised learning.	It is possible to learn larger and more complex models with unsupervised learning.
3.	Use training data to infer model.	No training data is used.
4.	Every input pattern that is used to train the network is associated with an output pattern.	The target output is not presented to the network.

5.	Trying to predict a function from labeled data.	Try to detect interesting relations in data.
6.	Supervised learning requires that the target variable is well defined and that a sufficient number of its values are given.	For unsupervised learning typically either the target variable is unknown or has only been recorded for too small a number of cases.
7.	Example : Optical character recognition.	Example : Find a face in an image.
8.	We can test our model.	We can not test our model.
9.	Supervised learning is also called classification.	Unsupervised learning is also called clustering.

2.7 Backpropagation

- Backpropagation is a training method used for a multi-layer neural network. It is also called the generalized delta rule. It is a gradient descent method which minimizes the total squared error of the output computed by the net.
- The backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.
- Backpropagation is a systematic method for training multiple layer ANN. It is a generalization of Widrow-Hoff error correction rule. 80 % of ANN applications use backpropagation.
- Fig. 2.7.1 shows backpropagation network.

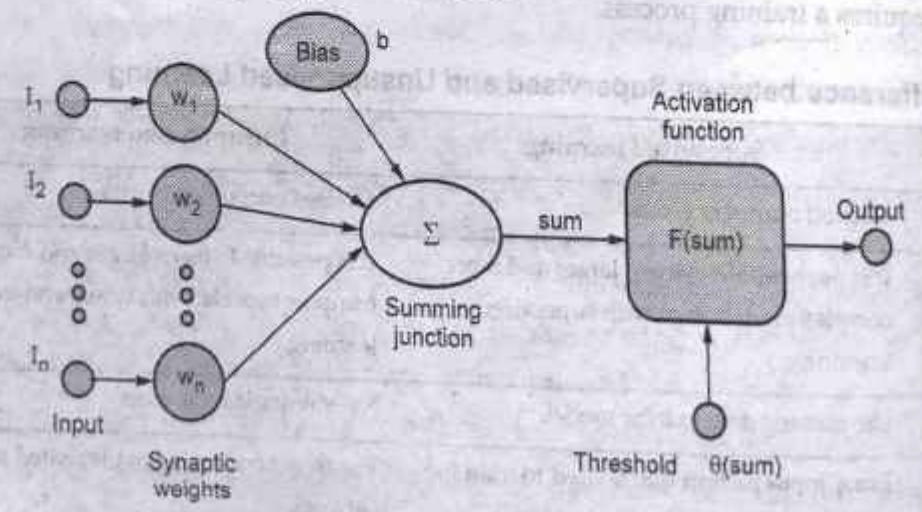


Fig. 2.7.1 Backpropagation network

- Consider a simple neuron :
  a. Neuron has a summing junction and activation function.
  b. Any non linear function which differentiable everywhere and increases everywhere with sum can be used as activation function.
  c. Examples : Logistic function, Arc tangent function, Hyperbolic tangent activation function.
- These activation function makes the multilayer network to have greater representational power than single layer network only when non-linearity is introduced.
- **Need of hidden layers :**
  1. A network with only two layers (input and output) can only represent the input with whatever representation already exists in the input data.
  2. If the data is discontinuous or non-linearly separable, the innate representation is inconsistent, and the mapping cannot be learned using two layers (Input and Output).
  3. Therefore, hidden layer(s) are used between input and output layers.
- **Weights** connects unit (neuron) in one layer only to those in the next higher layer. The output of the unit is scaled by the value of the connecting weight and it is fed forward to provide a portion of the activation for the units in the next higher layer.
- Backpropagation can be applied to an artificial neural network with any number of hidden layers. The training objective is to adjust the weights so that the application of a set of inputs produces the desired outputs.
- **Training procedure :** The network is usually trained with a large number of input-output pairs.
  1. Generate weights randomly to small random values (both positive and negative) to ensure that the network is not saturated by large values of weights.
  2. Choose a training pair from the training set.
  3. Apply the input vector to network input.
  4. Calculate the network output.
  5. Calculate the error, the difference between the network output and the desired output.
  6. Adjust the weights of the network in a way that minimizes this error.
  7. Repeat steps 2 - 6 for each pair of input-output in the training set until the error for the entire system is acceptably low.

Forward pass and backward pass :

- Backpropagation neural network training involves two passes.
  1. In the forward pass, the input signals moves forward from the network input to the output.
  2. In the backward pass, the calculated error signals propagate backward through the network, where they are used to adjust the weights.
  3. In the forward pass, the calculation of the output is carried out, layer by layer, in the forward direction. The output of one layer is the input to the next layer.
- In the reverse pass,
  a. The weights of the output neuron layer are adjusted first since the target value of each output neuron is available to guide the adjustment of the associated weights, using the delta rule.
  b. Next, we adjust the weights of the middle layers. As the middle layer neurons have no target values, it makes the problem complex.
- **Selection of number of hidden units :** The number of hidden units depends on the number of input units.
  1. Never choose h to be more than twice the number of input units.
  2. You can load p patterns of I elements into $\log_2$ p hidden units.
  3. Ensure that we must have at least 1/e times as many training examples.
  4. Feature extraction requires fewer hidden units than inputs.
  5. Learning many examples of disjointed inputs requires more hidden units than inputs.
  6. The number of hidden units required for a classification task increases with the number of classes in the task. Large networks require longer training times.

Factors influencing backpropagation training

- The training time can be reduced by using :
  1. **Bias :** Networks with biases can represent relationships between inputs and outputs more easily than networks without biases. Adding a bias to each neuron is usually desirable to offset the origin of the activation function. The weight of the bias is trainable similar to weight except that the input is always +1.
  2. **Momentum :** The use of momentum enhances the stability of the training process. Momentum is used to keep the training process going in the same general direction analogous to the way that momentum of a moving object behaves. In backpropagation with momentum, the weight change is a combination of the current gradient and the previous gradient.

2.7.1 Advantages and Disadvantages

Advantages of backpropagation :

1. It is simple, fast and easy to program.
2. Only numbers of the input are tuned and not any other parameter.
3. No need to have prior knowledge about the network.
4. It is flexible.
5. A standard approach and works efficiently.
6. It does not require the user to learn special functions.

Disadvantages of backpropagation :

1. Backpropagation possibly be sensitive to noisy data and irregularity.
2. The performance of this is highly reliant on the input data.
3. Needs excessive time for training.
4. The need for a matrix-based method for backpropagation instead of mini-batch.

2.8 Feed Forward Neural Networks

- Feed Forward Neural Network is an artificial neural network in which the connections between nodes does not form a cycle. The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.
- They are called feed forward because information only travels forward in the network (no loops), first through the input nodes, then through the hidden nodes (if present) and finally through the output nodes.
- Feed-forward networks tends to be simple networks that associates inputs with outputs. It can be used in pattern recognition. This type of organization is represented as bottom-up or top-down.
- Fig. 2.8.1 shows basic structure of a Feed Forward (FF) Neural Network.
- Input layer contains one or more input nodes. For example, suppose we want to predict whether it will rain tomorrow and base our decision on two variables, humidity and wind speed. In that case, our first input would be the value for humidity and the second input would be the value for wind speed.

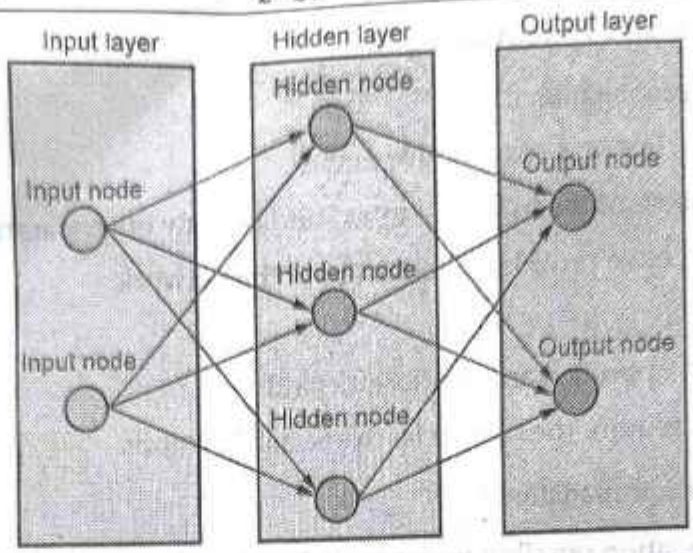


Fig. 2.8.1 Basic structure of a Feed Forward Neural Network

- Hidden layer : This layer contains an activation function.
- Output layer contains one or more output nodes.
- Feed foward neural networks are primarily used for supervised learning in cases where the data to be learned is neither sequential nor time-dependent.
- Feed-forward networks have the following characteristics :
  1. Perceptron's are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world and hence are called hidden layers.
  2. Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is constantly "fed forward" from one layer to the next and this explains why these networks are called feed-forward networks.
  3. There is no connection among perceptron's in the same layer.

2.8.1 Feedback Neural Networks

- Feedback networks also known as recurrent neural network or interactive neural network are the deep learning models in which information flows in backward direction. It allows feedback loops in the network. Feedback networks are dynamic in nature, powerful and can get much complicated at some stage of execution.
- Fig. 2.8.2 shows feedback neural network.
- Signals can travel in both the directions in feedback neural networks. Feedback neural networks are very powerful and can get very complicated. Feedback neural networks are dynamic.

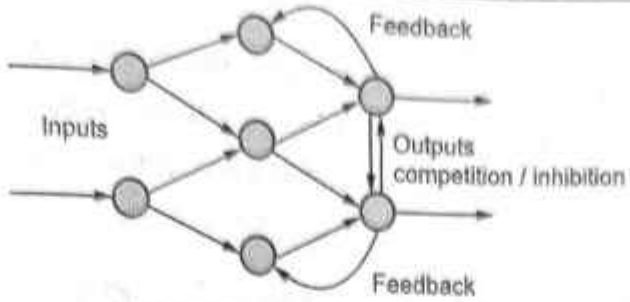


Fig. 2.8.2 Feedback neural network

- The 'state' in such network keep changing until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.
- Feedback neural network architecture is also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations. Feedback loops are allowed in such networks. They are used in content addressable memories.

2.8.3 Example and Applications of Feedforward NN

- **Physiological feedforward system :** during this, the feedforward management is epitomized by the conventional prevenient regulation of heartbeat prior to work out by the central involuntary.
- **Gene regulation and feedforward :** during this, a motif preponderantly seems altogether the illustrious networks and this motif has been shown to be a feedforward system for the detection of the non-temporary modification of atmosphere.
- **Automation and machine management :** Feedforward control may be discipline among the sphere of automation controls.
- **Parallel feedforward compensation with derivative :** This a rather new technique that changes the part of AN open-loop transfer operates of a non-minimum part system into the minimum part.

2.8.4 Difference between RNNs and feed-forward NN

- In a feed-forward neural network, the information only moves in one direction from the input layer, through the hidden layers, to the output layer. The information moves straight through the network.
- Feed-forward neural networks have no memory of the input they receive and are bad at predicting what's coming next. Because a feed-forward network only

considers the current input, it has no notion of order in time. It simply can remember anything about what happened in the past except its training.

- In a RNN the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.

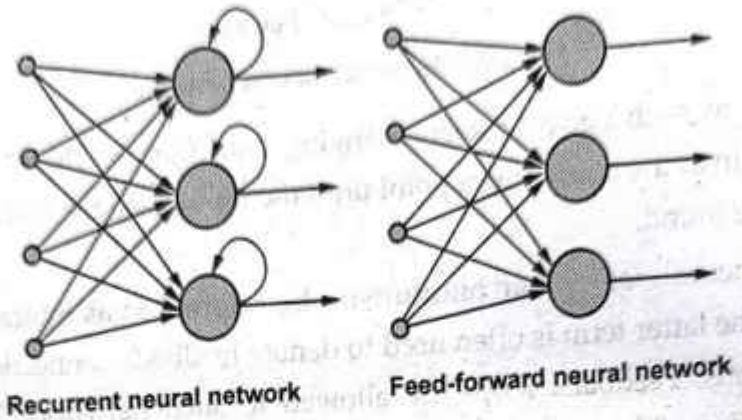


Fig. 2.8.3

Unit III

3 Associative Learning

Syllabus

Introduction, Associative Learning, Hopfield network, Error Performance in Hopfield networks, simulated annealing, Boltzmann machine and Boltzmann learning, State transition diagram and false minima problem, stochastic update, simulated annealing. Basic functional units of ANN for pattern recognition tasks: Pattern association, pattern classification and pattern mapping tasks.

Contents

3.1 Associative Learning, Hopfield Networks, Simulated Annealing, Boltzman Machine

3.2 Simulated Annealing

3.3 Boltzmann Machine and Boltzmann Learning

3.4 Basic Functional Units of ANN for Pattern Recognition Tasks

3.1 Associative Learning, Hopfield Networks, Simulated Annealing, Boltzman Machine

3.1.1 Associative Learning

- Associative learning is **a form of conditioning.** It is a matter of **temporal relationship between a signal and a sequence** that can have positive or negative consequences for the organism. Neuronal connections in the brain are created through associative learning of these connections.
- The physiological basis for associative learning was explained and proven by the psychologist Donald Oldung Hebb in his work on the function of Neurons with common synapses. His Hebb's learning rule describes a mapping of learning into neural networks. The term is used to describe a process in which the frequent interaction of two neurons increases their action potentials in relation to each other.
- **People learn much faster with positive associations than with negative or neutral associations.** Thus, if someone evaluates the sequence of a signal as profitable for him, this leads more quickly to a learning success. The term "strength of association" is also used here, which can lie between zero and a finite value.
- In contrast to the approach of learning via linkages, the **non-associative learning** about information that stands alone. With this information, the organism does not find a connection to a subsequent event.
- From the point of view of psychology, learning is about enduring changes in behaviour that result from experience. Accordingly, it is a process that enables living beings to react to situations through previous experiences and by incorporating further experiences.
- Associative learning is classical conditioning. This type of learning **functions through a conditioned response triggered by a conditioned stimulus.** It goes back to the Russian Ivan Pavlov and his still famous discovery when feeding dogs.
- To support Ivan Pavlov's experiments, the success of this learning method was also confirmed by tests with sea snails and rabbits. Rodents, for example, can be conditioned to new situations with the help of maze or olfactory learning methods.
- **People also learn more easily when new information can be linked to existing personal impressions, such as places, memories or emotions.** This makes it easier to remember something than when new information occurs in isolation.

- An associative neural network (ASNN) is an ensemble-based method inspired by the function and structure of neural network correlations in brain. The method operates by simulating the short- and long-term memory of neural networks. The long-term memory is represented by ensemble of neural network weights, while the short-term memory is stored as a pool of internal neural network representations of the input pattern.
- An associative neural network (ASNN) is a combination of an ensemble of the feed-forward neural networks and the K-nearest neighbor technique.
- The introduced network uses correlation between ensemble responses as a measure of distance among the analyzed cases for the nearest neighbor technique and provides an improved prediction by the bias correction of the neural network ensemble both for function approximation and classification. Actually, the proposed method corrects a bias of a global model for a considered data case by analyzing the biases of its nearest neighbors determined in the space of calculated models. An associative neural network has a memory that can coincide with the training set. If new data become available the network can provide a reasonable approximation of such data without a need to retrain the neural network ensemble.

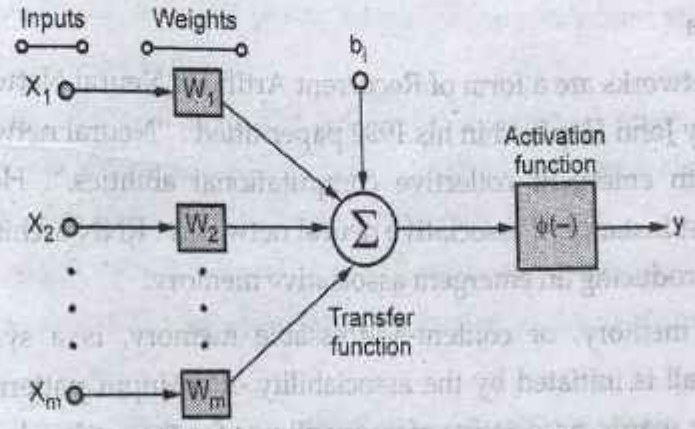


Fig. 3.1.1 Associate neural network

- The traditional multi-layer neural network (MLP) is a memory-less approach. This means that after training is complete all information about the input patterns is stored in the neural network weights and input data are no longer needed, that is there is no explicit storage of any presented example in the system complicated, there is no guarantee that all details of f, i.e. its fine structure, will be represented. Thus, the global model can be inadequate because it does not describe equally well the entire state space with poor performance of the method being mainly due to a high bias of the global model in some particular regions of space. The same problem

of bias is also pertinent if neural networks are used for classification. The MLP variance can also contribute to poor performance of this method. However, the variance can be decreased by analyzing a large number of networks, that is using artificial neural network ensemble and taking, for example, a simple average of all networks as the final model. The problem of bias of MLP cannot be so easily addressed simply by using larger neural networks since such networks can fall in a local minimum and thus can still have a considerable bias.

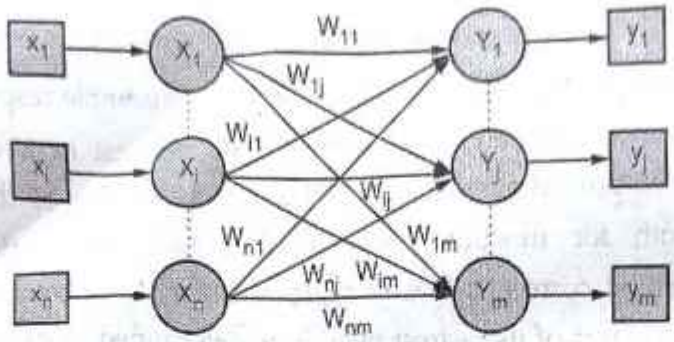


Fig. 3.1.2 Associative neural network

3.1.2 Hopfield Network

3.1.2.1 Introduction

- Hopfield networks are a form of Recurrent Artificial Neural Networks (RNNs), first described by John Hopfield in his 1982 paper titled : "Neural networks and physical systems with emergent collective computational abilities.". Hopfield Networks were the first instance of associative neural networks : RNN architectures which are capable of producing an emergent associative memory.
- Associative memory, or content-addressable memory, is a system in which memory recall is initiated by the associability of an input pattern to a memorized one. In other words, associative memory allows for the retrieval and completion of a memory using only an incomplete or noisy portion of it.
- As an example, a person might see a picture they like and be 'brought back' to the memory of their first time watching it. The context, people, surroundings, and emotions in that memory can be retrieved via subsequent exposure to only a portion of the original stimulus : the song alone. These features of Hopfield Networks (HNs) made them a good candidate for early computational models of human associative memory, and marked the beginning of a new era in neural computation and modeling.

- Hopfield's unique network architecture was based on the Ising model, a physics model that explains the emergent behavior of the magnetic fields produced by ferromagnetic materials. The model is usually described as a flat graph, where nodes represent magnetic dipole moments of regular repeating arrangements. Each node can occupy one of two states (i.e. spin up or spin down, + 1 or – 1), and the agreement of states between adjacent nodes is energetically favorable. As each dipole 'flips', or evolves to find local energetic minima, the whole material trends toward a steady-state, or global energy minimum. HNs can be thought of in a similar way where the network is represented as a 2D graph and each node (or neuron) occupies one of two states: active or inactive. Similar to Ising models, the dynamic behavior of a HN is ultimately determined according to a metaphorical notion of 'system energy', and the network converges to a state of 'energetic minimum', which in the case of a learned network happens to be a memory.

3.1.2.2 Characteristics of Hopfield Network - Qualities and Attributes

- Each neuron in the network has three qualities namely,
  1. **Connections to other neurons -** Each neuron in the network is connected to all other neurons, and each connection has a unique strength, or weight, analogous to the strength of a synapse. These connection strengths are stored inside a matrix of weights.
  2. **Activation -** This is computed via the net input from other neurons and their respective connection weights, loosely analogous to the membrane potential of a neuron. The activation takes on a single scalar value.
  3. **Bipolar state -** This is the output of the neuron, computed using the neuron's activation and a thresholding function, analogous to a neuron's 'firing state.' In this case, – 1 and + 1.
- The *information* or *memories* of a Hopfield Network are stored in the strength of connections between neurons, much in the same way that information is thought to be stored in the brain according to models of long term memory. The connections, and their respective weights, directly affect the activity of the network with each generation, and ultimately determine the state to which the network converges. You can picture the weight between 2 neurons, neuron-a and neuron-b, as the extent to which the output of neuron-a will contribute to the activation of neuron-b, and vice versa. More on activation and its effect on neuronal state below.

- Importantly, the connections between neurons in a HN are symmetric. This means that if neuron-a is connected to neuron-b with a strength of + 1, neuron-b's connection to neuron-a is also + 1. One can store the weights of a network with n neurons in a square symmetric matrix of dimensions $n \times n$.
- Because neurons do not connect back to themselves, the weights on the matrix's diagonal are effectively null. It's important to note that the network structure, including the number of neurons and their connections in the network, is fixed.
- The weights of those connections are the tunable parameter. By 'network learning', it means it is the fine-tuning of each weight in the network in order to achieve some desirable output. In the case of an HN, the initial state configuration can be thought of as an input, and its final state configuration as the output. Because each neuron is updated depending on linear combinations of its inputs, the entire network is really one massive mathematical function: a dynamical system, which, with each update generation, takes its past output as its current input. The parameters of the function, the weights, determine the way the network evolves over time.

3.12.3 Activation Function and Thresholding Function

- Any neuron's instantaneous activation can be calculated by taking the weighted sum of inputs from the neurons it's connected to :

$$y_i = \sum_j y_i w_{ij}$$

Equation 1 : The activation function of a neuron in a binary Hopfield Network,

Where y_i represents the activation of neuron-i in the network, y_j represents a vector of the respective outputs of all neurons inputting to neuron-i, and w_{ij} the symmetric weight of the connection between neurons i and j.

The activation of a neuron is used to determine the state or output, of the neuron according to a thresholding function :

$$s_i = \begin{cases} +1 & \text{if } y_i > 0 \\ -1 & \text{if } y_i < 0 \end{cases}$$

Equation 2 : The binary-threshold output function,

Where s_i represents a neuron-i's given state and output.

- As the state of each neuron in the network is updated, its contributions to the activities of other neurons change; this in turn feeds back and modulates more activity and so on, until the network reaches a stable formation.

- If a neuron's activation during a given generation is positive, its output will be + 1 and vice versa. Another way to think of this is that a given neuron is receiving a 'field' of inputs analogous to its activation. Each and every input the neuron receives is weighted, and summed, to yield a net activation, be it positive or negative. If the sign of that field differs from the sign of the neuron's current output, its state will flip to align itself. If the sign of the field of input matches the sign of the neuron's current output, it will stay the same.
- If the behavior of the overall network is considered according to these rules, the quality of recurrence becomes clear; each network update, or output, becomes the input for the next timestep.

Translation to the distributed storage and representation of information in the network

- While each neuron can represent one of two states at a given time, the overall state of the network, termed s, can represent a string of binary information. For example consider the pattern,

$$\text{Pattern} = [0, 1, 0, 1, 0, 1, 0, 1, 0]$$

Fig. 3.1.3 A binary string of information

- This pattern is a 1×9 binary vector which can be reshaped to a 3×3 matrix as below,

Pattern.reshape (3, 3) = [[0, 1, 0], [1, 0, 1], [0, 1, 0]] → when drawn looks like :

Fig. 3.1.4 A binary string of information is reshaped and visualized as an image

- An image is just a matrix of numerical pixel values, so any image which can be represented with pixel values occupying binary states can be accurately represented by the states of a binary HN. There are many forms of information can be represented in a binary / bipolar fashion.
- If a pattern takes the form of an image, the pixels themselves are represented using individual neurons. In the example above there is a 3×3 matrix of pixels, so 9 neurons are required to fully represent the image. Another way to visualize this is to represent the network as a graph, as shown below,

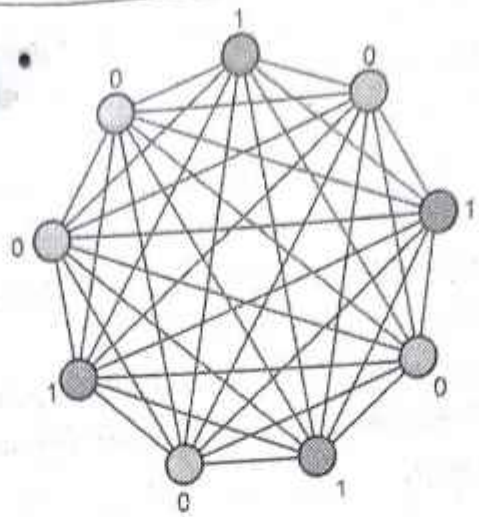


Fig. 3.1.5 Binary string representation using Hopfield Network

- In this network graph, the nodes represent neurons and the edges represent the connections between them. The black dot (outside the network image) indicates the neuron whose state represents the first element, 1, in the pattern vector. Following the state of the neurons in clockwise order yields the rest of the pattern.
- In a computer, the state of each neuron is represented as the individual elements in a 1×9 vector, S. Here the information to be stored is directly represented using the respective states of neurons in the network :

$$S = [0, 1, 0, 1, 0, 1, 0, 1, 0].$$

- Any arbitrary information string of n bits can be represented by a network of n neurons. It is the combined activity of the neurons in the network which collectively represent one's target information in a distributed fashion. In most HNs, the states of neurons are bipolar (– 1 or + 1). For the sake of simplicity, the example above uses binary 0 or 1, from here on out assume that the neurons occupy bipolar states.

3.1.2.4 Evolution Towards a Memorized Pattern in Hopfield Network

- HNs are often referred to as 'attractor networks,' because they tend to evolve to, or are attracted to, a "steady state". To see this in action, consider the example of binary pattern above. Assuming that the network has already learned the pattern, initialize the network to a random state, and let the network run according to the update rules described above, one neuron at a time.

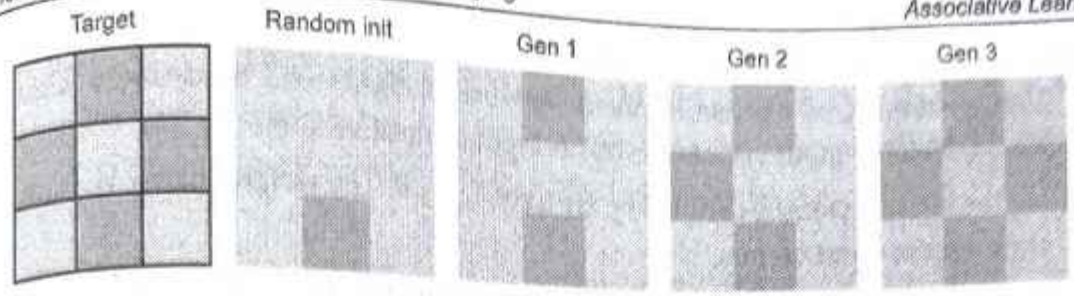


Fig. 3.1.6 The evolution of a Hopfield Network which has learned the target pattern, and is allowed to evolve one neuron at a time

- Here it is seen that despite the network's randomly initialized state, it was able to restore the target memory in just three update generation steps - the overall state of the network was attracted to the memory state.

3.1.2.5 Types of Hopfield Networks (Discrete Hopfield Network and Continuous Hopfield Network)

- The Hopfield network is of two types,

1. Discrete hopfield network
2. Continuous hopfield network

1. Discrete hopfield network

- When the network is operated in discrete line fashion it is called as discrete hopfield network.
- The network takes two-valued inputs : binary (0, 1) or bipolar (+ 1, – 1); the use of bipolar inputes makes the analysis easier. The network has symmetrical weights with no self-connections, i.e.,

$$W_{ij} = W_{ji};$$

$$W_{ij} = 0 \quad \text{if } i = j$$

Architecture of discrete hopfield network

- The hopfield's model consists of processing elements with two outputs, one inverting and the other non-inverting. The outputs from each processing element are fed back to the input of other processing elements but not to itself.

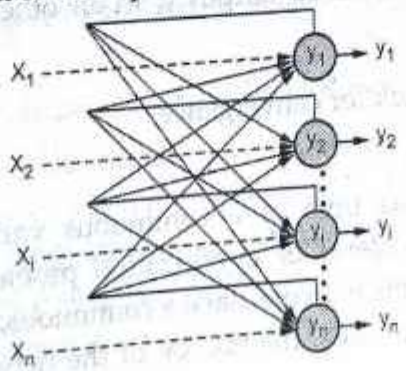


Fig. 3.1.7 Discrete Hopfield network

Training algorithm of discrete hopfield network

- During training of discrete hopfield network, weights will be updated. One can have the binary input vectors as well as bipolar input vectors.
- Let the input vectors be denoted by s(p), p = 1, ..., P. The weight matrix W is used to store a set of input vectors.
- In case of input vectors being binary, the weight matrix $W = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_{p=1}^{P} [2s_i(p) - 1] [2s_j(p) - 1] \text{ for } i \neq j$$

- When the input vectors are bipolar, the weight matrix $W = \{w_{ij}\}$ can be defined as

$$w_{ij} = \sum_{p=1}^{P} [s_i(p)] [s_j(p)] \text{ for } i \neq j$$

Testing algorithm of discrete hopfield net

Step 0 : Initialize the weights to store patterns, i.e., weights obtained from training algorithm using Hebb rule.

Step 1 : When the activations of the net are not converged, then perform steps 2 - 8.

Step 2 : Perform steps 3 - 7 for each input vector X.

Step 3 : Make the initial activations of the net equal to the external input vector X :

$$y_i = x_i \text{ for } i = 1 \text{ to } n$$

Step 4 : Perform steps 5 - 7 for each unit y_i. (Here, the units are updated in random order.)

Step 5 : Calculate the net input of the network :

$$y_i n_i = x_i + \sum_j y_j w_{ji}$$

Step 6 : Apply the activations over the net input to calculate the output :

$$y_i = y_i 0 \text{ if } y_i n_i > \theta_i \quad y_i n_i = \theta_i \text{ if } y_i n_i < \theta_i$$

where θ_i is the threshold and is normally taken as zero.

Step 7 : Now feedback the obtained output y_i to all other units. Thus, the activation vectors are updated.

Step 8 : Finally, test the network for convergence.

2. Continuous hopfield network

- Continuous network has time as a continuous variable and can be used in associative memory problems or optimization problems like traveling salesman problem. The nodes of this network have a continuous, graded output rather than two state binary output. Thus, the energy of the network decreases continuously with time.

- Unlike the discrete hopfield networks, here the time parameter is treated as a continuous variable. So, instead of getting binary / bipolar outputs, one can obtain values that lie between 0 and 1. It can be used to solve constrained optimization and associative memory problems. The output is defined as :

$$v_i = g(u_i)$$

where,

v_i = Output from the continuous hopfield network

u_i = Internal activity of a node in continuous hopfield network

3.1.2.6 Energy Function

- The hopfield networks have an energy function associated with them. It either diminishes or remains unchanged on update (feedback) after every iteration. The energy function for a continuous hopfield network is defined as :

$$E = 0.5 \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} v_i w_j + \sum_{i=1}^{n} \theta_i v_i$$

- To determine if the network will converge to a stable configuration, one need to see if the energy function reaches its minimum by :

$$\frac{d}{dt} E \leq 0$$

- The network is bound to converge if the activity of each neuron with respect to time is given by the following differential equation :

$$\frac{d}{dt} u_i = \frac{-u_i}{\tau} + \sum_{j=1}^{n} w_{ij} v_j + \theta_i$$

3.1.2.7 Error Performance in Hopfield Networks

- One of the critical shortcomings of Hopfield Neural Network (HNN) is that the network may not always converge to a fixed point. HNN, is limited to local optimization during training to achieve network stability. In the Hopfield network a solution of an optimization problem is obtained after the network is relaxed to an equilibrium state.
- Other major performance issues of Hopfield networks are as follows,
  1. **Memory capacity :** The number m of training patterns should be about the number of neurons n or less (according to Hopfield, it should be less than 0.15 n ; according to some new results, m ε 0.5 n/log n). This means that the memorizing capacity of a Hopfield network is severely limited. Catastrophic "forgetting" may occur if we try to memorize more patterns than the network is supposed to handle.

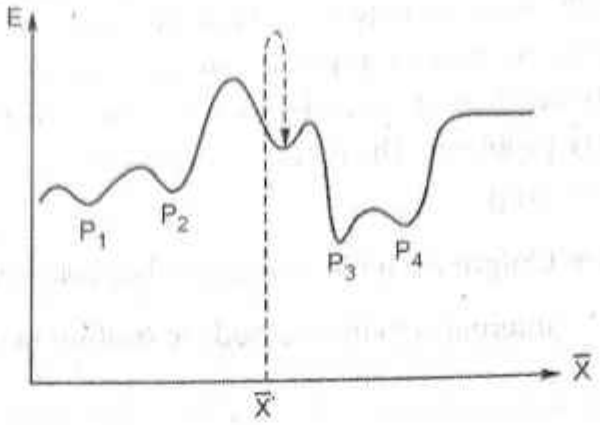


Fig. 3.1.8 A network may relax into a "spurious" state, which phenomenon is called the local minima problem

2. **Discrepancy limitation :** The new pattern to be recognized as one of the training patterns should not differ from any training pattern by more than about 25 %.
3. **Orthogonality between patterns :** The more orthogonal (dissimilar) the training patterns, the better the recognition.
4. **Spurious states of attraction :** Sometimes the network learns some patterns (creates basins of attraction) called spurious states, which are not presented in the set of training patterns. An example of such a state is shown in Fig. 3.1.8. In order to overcome this limitation, Hopfield introduced a "little bit of unlearning" for every learning pattern :

$$\Delta w_{ij} = \alpha x_i^{(p)} \cdot x_j^{(p)} \text{ for the } p^{th} \text{ pattern } x^{(p)}$$

Spurious states may be useful, if the Hopfield network is used for partial matching and approximate reasoning. The input values are between 0 and 1 represent the certainty of the facts or fuzzy membership degrees.

5. **Weight symmetry :** The weight matrix has to be symmetrical in order for the network to reach an equilibrium. The symmetrical synaptic weights are not at all biologically plausible, but they are a useful limitation here.

3.2 Simulated Annealing

3.2.1 Introduction

- Simulated annealing is a **method for solving unconstrained and bound constrained optimization problems.** The method models the physical process of heating a material and then slowly lowering the temperature to decrease defects thus minimizing the system energy.

- The simulated annealing algorithm is based upon physical annealing in real life. Physical annealing is the process of heating up a material until it reaches an **annealing temperature** and then it will be **cooled down** slowly in order to change the material to a desired structure. When the material is hot, the molecular structure is weaker and is more susceptible to change. When the material cools down, the molecular structure is harder and is less susceptible to change.
- Another important part of this analogy is the following equation from Thermal Dynamics,

$$P(\Delta E) = e^{-\frac{\Delta E}{k * t}}$$

- This equation calculates the probability that the energy magnitude will increase. One can calculate this value given some energy magnitude and some temperature t along with the Boltzmann constant k.
- **From AI point of view, simulated annealing** is a stochastic global search optimization algorithm.
- This means that it makes use of randomness as part of the search process. This makes the algorithm appropriate for nonlinear objective functions where other local search algorithms do not operate well.
- Simulated Annealing (SA) mimics the physical annealing process but is used for optimizing parameters in a model. This process is very useful for situations where there are a lot of local minima such that algorithms like Gradient Descent would be stuck at.
- It modifies a single solution and searches the relatively local area of the search space until the local optima is located. It may accept worse solutions as the current working solution.
- The likelihood of accepting worse solutions starts high at the beginning of the search and decreases with the progress of the search, giving the algorithm the opportunity to first locate the region for the global optima, escaping local optima, then hill climb to the optima itself.

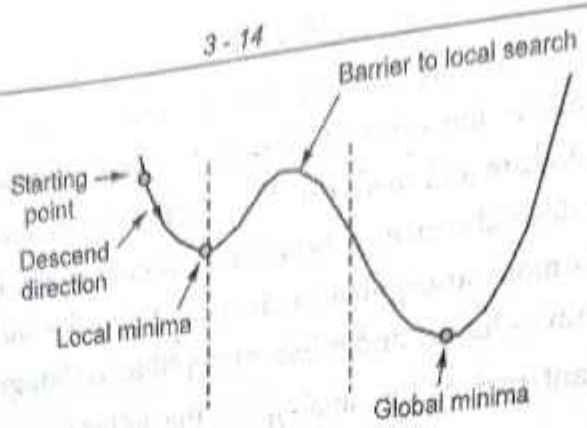


Fig. 3.2.1

- In problems like the one in above figure, if gradient descent started at the starting point indicated, it would be stuck at the local minima and not be able to reach the global minima.

3.2.2 Simulated Annealing terms

Objective function

The objective function is the function that is to be optimized.

Temperature

- The *temperature* is a parameter in simulated annealing that affects two aspects of the algorithm :
  - The distance of a trial point from the current point.
  - The probability of accepting a trial point with higher objective function value.
- Temperature can be a vector with different values for each component of the current point. Typically, the initial temperature is a scalar.
- Temperature decreases gradually as the algorithm proceeds. One can specify the initial temperature as a positive scalar or vector in the InitialTemperature option. One can specify the temperature as a function of iteration number as a function handle in the TemperatureFcn option. The temperature is a function of the Annealing parameter, which is a proxy for the iteration number. The slower the rate of temperature decrease, the better the chances are of finding an optimal solution but the longer the run time.

Annealing parameter

- The *annealing parameter* is a proxy for the iteration number. The algorithm can raise temperature by setting the annealing parameter to a lower value than the current iteration. One can specify the temperature schedule as a function handle with the TemperatureFcn option.

Reannealing

- *Annealing* is the technique of closely controlling the temperature when cooling a material to ensure that it reaches an optimal state. Reannealing raises the temperature after the algorithm accepts a certain number of new points, and starts the search again at the higher temperature. Reannealing avoids the algorithm getting caught at local minima. Specify the reannealing schedule with the ReannealInterval option.

3.2.3 Simulated Annealing Algorithm

Step 1 : First start with an initial solution $s = S_0$. This can be any solution that fits the criteria for an acceptable solution. Let the initial temperature $t = t_0$.

Step 2 : Setup a temperature reduction function *alpha*. There are usually 3 main types of temperature reduction rules :

1. Linear reduction rule : $t = t - \alpha$
2. Geometric reduction rule : $t = t * \alpha$
3. Slow-decrease rule : $t = \frac{t}{1+\beta t}$

- Each reduction rule reduces the temperature at a different rate and each method is better at optimizing a different type of model. For the 3rd rule, beta is an arbitrary constant.

Step 3 : Starting at the initial temperature, loop through *n* iterations of step 4 and then decrease the temperature according to *alpha*. Stop this loop until the *termination conditions* are reached. The termination conditions could be reaching some end temperature, reaching some acceptable threshold of performance for a given set of parameters, etc. The mapping of time to temperature and how fast the temperature decreases is called the **Annealing Schedule**.

Step 4 : Given the neighbourhood of solutions N(s), pick one of the solutions and calculate the difference in cost between the old solution and the new neighbour solution. The neighbourhood of a solution are all solutions that are close to the solution. For example, the neighbourhood of a set of five parameters might be if one of the five parameters were to change but kept the remaining four the same.

Step 5 : If the difference in cost between the old and new solution is greater than 0 (the new solution is better), then accept the new solution. If the difference in cost is less than 0 (the old solution is better), then generate a random number between 0 and 1 and accept it if it's under the value calculated from the energy magnitude equation from before.

- In the simulated annealing case, the equation has been altered to the following :

$$P = \begin{cases} 1 & \text{if } \Delta c \le 0 \\ e^{\frac{-\Delta c}{t}} & \text{if } \Delta c > 0 \end{cases}$$

Where the *delta c* is the change in cost and the *t* is the current temperature.

- The *P* calculated in this case is the probability that one should accept the new solution.

High vs. Low Temperature

- Due to the way the probability is calculated, when the temperature is higher, it is more likely that the algorithm accepts a worse solution. This promotes **Exploration** of the search space and allows the algorithm to more likely travel down a sub optimal path to potentially find a global maximum.

Change	Temperature	Acceptance probability
0.2	0.9	0.8007
0.4	0.9	0.6412
0.6	0.9	0.5134
0.8	0.9	0.4111

- When the temperature is lower, the algorithm is less likely or will not to accept worse solution. This promotes **Exploitation** which means that once the algorithm is in the right search space, there is no need to search other sections of the search space and should instead try to converge and find the global maximum.

Change	Temperature	Acceptance probability
0.2	0.1	0.1353
0.4	0.1	0.0183
0.6	0.1	0.0025
0.8	0.1	0.0003

Problems - to be optimized with simulated annealing

- Travelling Salesman Problem (TSP)
- Scheduling problems
- Task allocations
- Graph colouring and partitioning
- Non-linear function optimizations

- Consider an example of a difficult optimization task is the chip floor planning problem. Assume that an engineer at circuit manufacturing company has given a task to design the layout for an integrated circuit. There are set of modules of different shapes / sizes and a fixed area on which the modules can be placed. There are a number of objectives need to achieve: maximizing ability for wires to connect components, minimize net area, minimize chip cost, etc. With these in mind, engineer had to create a cost function, taking all, say, 1000 variable configurations and returning a single real value representing the 'cost' of the input configuration. This can be called as the objective function, since the goal is to minimize its value.
- A naive algorithm would be a complete space search - to search all possible configurations until one finds the minimum. This may suffice for functions of few variables, but the problem one has in mind would entail such a brute force algorithm to fun in O(n!).
- Due to the computational intractability of problems like these, and other NP-hard problems, many optimization heuristics have been developed in an attempt to yield a good, albeit potentially suboptimal, value. In current scenario one doesn't necessarily need to find a strictly optimal value - finding a near-optimal value would satisfy the goal. One widely used technique is simulated annealing, by which one can introduce a degree of stochasticity, potentially shifting from a better solution to a worse one, in an attempt to escape local minima and converge to a value closer to the global optimum.
- Simulated annealing is based on metallurgical practices by which a material is heated to a high temperature and cooled. At high temperatures, atoms may shift unpredictably, often eliminating impurities as the material cools into a pure crystal. This is replicated via the simulated annealing optimization algorithm, with energy state corresponding to current solution.
- In this algorithm, an initial temperature is defined, often set as 1 and a minimum temperature, on the order of 10^{-4}. The current temperature is multiplied by some fraction alpha and thus decreased until it reaches the minimum temperature. For each distinct temperature value, the core optimization routine is run a fixed number of times. The optimization routine consists of finding a neighboring solution and accepting it with probability $e^{(f(c) - f(n))}$ where c is the current solution and n is the neighboring solution. A neighboring solution is found by applying a slight perturbation to the current solution. This randomness is useful to escape the common pitfall of optimization heuristics - getting trapped in local minima. By

potentially accepting a less optimal solution than the current one, and accepting with probability inverse to the increase in cost, the algorithm is more likely converge near the global optimum. Designing a neighbor function is quite tric and must be done on a case by case basis, but below are some ideas for findi neighbors in locational optimization problems,

- Move all points 0 or 1 units in a random direction
- Shift input elements randomly
- Swap random elements in input sequence
- Permute input sequence
- Partition input sequence into a random number of segments and perm segments

3.2.4 Advantages vs. Disadvantages of Simulated Annealing

Advantages

- It is easy to implement and use.
- It provides optimal solutions to a wide range of problems.

Disadvantages

- It can take a long time to run if the annealing schedule is very long.
- In the algorithm there are a lot of parameters those need to be tuned for bet performance.

3.3 Boltzmann Machine and Boltzmann Learning

3.3.1 Introduction

- A Boltzmann machine is a **type of recurrent neural network in which nodes ma binary decisions with some bias.** Boltzmann machines can be strung together make more sophisticated systems such as deep belief networks. A Boltzman machine is also known as a stochastic Hopfield network with hidden units.
- Several Boltzmann machines can be collaborated together to make even mo sophisticated systems such as a deep belief network. Coined after the famou Austrian scientist Ludwig Boltzmann, who based the foundation on the idea Boltzmann distribution in the late 20th century, this type of network was furth developed by Stanford scientist Geoffrey Hinton.

- It derives its idea from the world of thermodynamics to conduct work toward desired states. It consists of a network of symmetrically connected, neuron-like units that make decisions stochastically whether to be active or not.
- A Boltzmann machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm that allows them to discover interesting features in datasets composed of binary vectors. The learning algorithm is very slow in networks with many layers of feature detectors, but it can be made much faster by learning one layer of feature detectors at a time.
- Boltzmann machines are used to solve two quite different computational problems. For a search problem, the weights on the connections are fixed and are used to represent the cost function of an optimization problem. The stochastic dynamics of a Boltzmann machine then allow it to sample binary state vectors that represent good solutions to the optimization problem.
- For a learning problem, the Boltzmann machine is shown a set of binary data vectors and it must find weights on the connections so that the data vectors are good solutions to the optimization problem defined by those weights.
- To solve a learning problem, Boltzmann machines make many small updates to their weights, and each update requires them to solve many different search problems.

3.3.2 Objective of Boltzmann Machine and Architecture

- The main purpose of Boltzmann machine is to optimize the solution of a problem. It is the work of Boltzmann machine to optimize the weights and quantity related to that particular problem.

Architecture

- A Boltzmann machine has two kinds of nodes
  - Visible nodes : These are nodes that can be measured and are measured.
  - Hidden nodes : These are nodes that cannot be measured or are not measured.
- According to some experts, a Boltzmann machine can be called a stochastic Hopfield network which has hidden units. It has a network of units with an 'energy' defined for the overall network.
- Boltzmann machines seek to reach thermal equilibrium. It essentially looks to optimize global distribution of energy. But the temperature and energy of the system are relative to laws of thermodynamics and are not literal.

- A Boltzmann machine is made up of a learning algorithm that enables it to disco interesting features in datasets composed of binary vectors. The learning algori tends to be slow in networks that have many layers of feature detectors but it possible to make it faster by implementing a learning layer of feature detectors.
- They use stochastic binary units to reach probability distribution equilibrium minimize energy). It is possible to get multiple Boltzmann machines to collabo together to form far more sophisticated systems like deep belief networks.
- The following diagram shows the architecture of Boltzmann machine. It is cl from the diagram, that it is a two-dimensional array of units. Here, weights interconnections between units are – **p** where **p > 0**. The weights of self-connectio are given by **b** where **b > 0**.

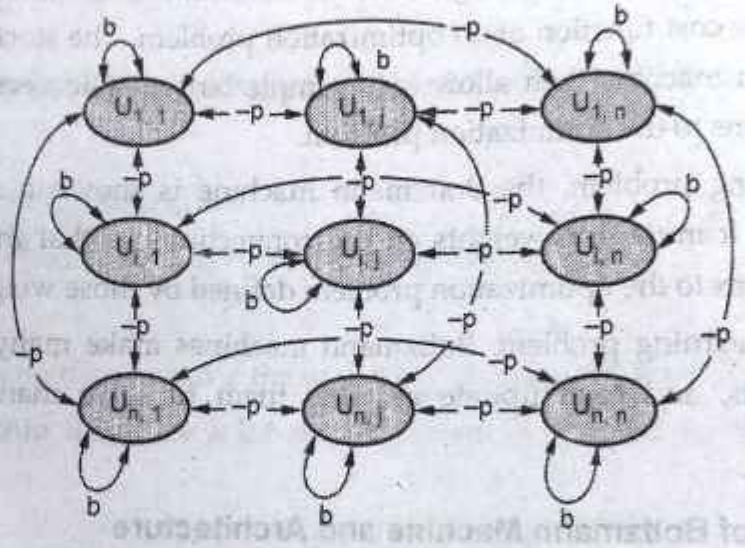


Fig. 3.3.1 Boltzmann machine architecture

- The architecture of the Boltzmann machine comprises a shallow, two-layer neu network that also constitutes the building blocks of the deep network. The fi layer of this model is called the visible or input layer and the second is the hidd layer.
- They consist of neuron-like units called a node and nodes are the areas whe calculations take place. These nodes are interconnected to each other across laye but no two nodes of the same layer are linked.
- Therefore there is no intra-layer communication and hence being one of th restrictions in a Boltzmann machine. Each node through computation process input, and makes stochastic decisions about whether to transmit that input or n When data is fed as input, these nodes learn all the parameters, their patterns a

correlation between them, on their own and form an efficient system. Hence a Boltzmann machine is also termed as an unsupervised deep learning model.

- This model can then be trained to get ready to monitor and study abnormal behaviour depending on what it has learnt. The coefficients that modify the inputs are randomly initialized. Each visible node takes a low-level feature from an item present in the dataset to be learned.
- The result of those two operations is fed into an activation function which in turn produces the node's output also known as the strength of the signal passing through it. The outputs of the first hidden layer would be passed as inputs to the second hidden layer, and from there through as many hidden layers created, until they reach a final classifying layer.
- For simple feed-forward movements, the nodes function as an autoencoder. Learning is typically very slow in Boltzmann machines with a high number of hidden layers as large networks may take a long time to approach their equilibrium distribution, especially when the weights are large and the distribution being highly multimodal.

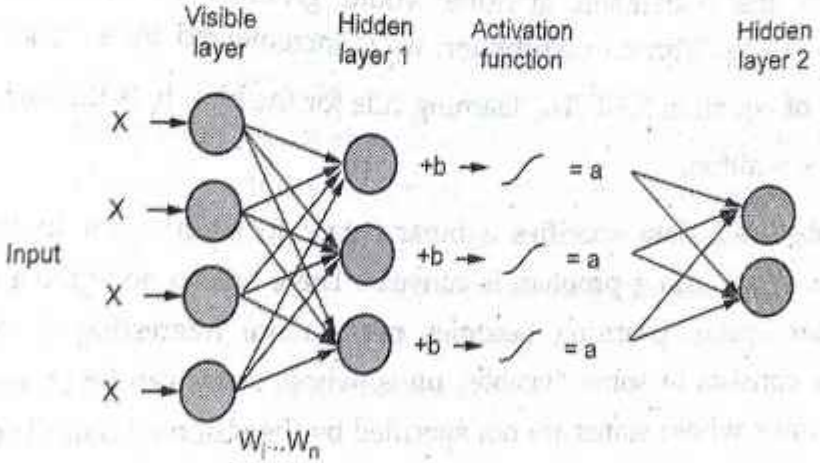


Fig. 3.3.2 Multiple hidden layers

- Boltzmann Machines consist of a learning algorithm that helps them to discover interesting features in datasets composed of binary vectors. The learning algorithm is generally slow in networks with many layers of feature detectors but can be made faster by implementing a learning layer of feature detectors.
- Boltzmann machines are typically used to solve different computational problems such as, for a search problem, the weights present on the connections can be fixed and are used to represent the cost function of the optimization problem. Similarly, for a learning problem, the Boltzmann machine can be presented with a set of binary data vectors from which it must find the weights on the connections so that

the data vectors are good solutions to the optimization problem defined by weights. Boltzmann machines make many small updates to their weights, and update requires solving many different search problems.

3.3.3 Learning in Boltzmann Machine

- Given a training set of state vectors (the data), learning consists of finding we and biases (the parameters) that make those state vectors good. More specifi the aim is to find weights and biases that define a Boltzmann distribution in w the training vectors have high probability. By differentiating and using the fac $\partial E(v)/\partial w_{ij} = s_i^v s_j^v$ it can be shown that

$$\sum_{v \in \text{data}} \frac{\partial \log P(v)}{\partial w_{ij}} = (s_i s_j)_{\text{data}} - (s_i s_j)_{\text{model}} \quad \text{...(3.}$$

where $(s_i s_j)_{data}$ is the expected value of $s_i s_j$ in the data distribution and $(s_i s_j)_{model}$ is excepted value when the Boltzmann machine is sampling state vectors from equilibrium distribution at a temperature of 1. To perform gradient ascent in the probability that the Boltzmann machine would generate the observed data w sampling from its equilibrium distribution w_{ij} is incremented by a small learning times the RHS of equation 3.3.1. The learning rule for the bias, b_i is the same as equa 3.3.1, but with s_j omitted.

- If the observed data specifies a binary state for every unit in the Boltzm machine, the learning problem is convex : There are no non-global optima in parameter space. Learning becomes much more interesting if the Boltzm machine consists of some "visible" units, whose states can be observed data hidden units whose states are not specified by the observed data. The hidden u acts as latent variables (features) that allow the Boltzmann machine to mo distributions over visible state vectors that cannot be modelled by direct pairw interactions between the visible units. A surprising property of Boltzm machines is that of Boltzmann machines is that, even with hidden units learning rule remains unchanged. This makes it possible to learn binary featu that capture higher-order structure in the data. With hidden units, the expecta $(s_i\ s_j)_{data}$ is the average, over all data vectors, of the expected value of $s_i s_j$ whe data vector is clamped on the visible units and the hidden units are repeat updated until they reach equilibrium with the clamped data vector.

- It is surprising that the learning rule is so simple because $\partial \log P(v)/\partial w_{ij}$ depends on all the other weights in the network. Fortunately, the difference in the two correlations in equation 3.3.1 tells w_{ij} everything it needs to know about the other weights. This makes it unnecessary to explicitly propagate error derivatives, as in the backpropagation algorithm.

3.3.4 Uses of Boltzmann Machine

- The main purpose of the Boltzmann machine is to optimize the solution of a problem. It optimizes the weights and quantities related to the particular problem assigned to it. This method is used when the main objective is to create mapping and learn from the attributes and target variables in the data. When the objective is to identify an underlying structure or the pattern within the data, unsupervised learning methods for this model are considered to be more useful. Some of the most popular unsupervised learning methods are clustering, dimensionality reduction, anomaly detection and creating generative models.
- Each of these techniques has a different objective of detecting patterns such as identifying latent grouping, finding irregularities in the data, or generating new samples from the available data. These networks can also be stacked layer-wise to build deep neural networks that capture highly complicated statistics. The use of restricted Boltzmann machines has gained popularity in the domain of imaging and image processing as well since they are capable of modelling continuous data that are common to natural images. They also are being used to solve complicated quantum mechanical many-particle problems or classical statistical physics problems like the Ising and Potts classes of models.

3.3.5 Energy-Based Models

- The Boltzmann machine's sampling distribution uses the Boltzmann distribution. The following equation controls the Boltzmann distribution -

$$P_i = \frac{e^{(-\epsilon_i/kT)}}{\sum e^{(-\epsilon_j/kT)}}$$

k - Boltzmann's constant

ϵ_i - The energy of the system in state i

P_i - The probability of the system being in state i

$\sum e^{(-\epsilon_i/kT)}$ - The sum of values for all possible states of the system

T - The temperature of the system

- The Boltzmann distribution defines many system states, and Boltzmann mach use this distribution to generate various machine states. According to the equa above, the likelihood that a system will be in state I diminish as system energy r In its lowest energy state, the system is, therefore, the most stable (gas is the m stable when it spreads). In this case, synapses' weights are used to define system's energy in Boltzmann's machines. The system always strives to find lowest energy state by modifying the weights after it has been trained and weights have been established.

3.3.6 Types of Boltzmann Machines (Restricted BMs, Deep Belief Netwo Deep BMs)

1. Restricted Boltzmann Machines (RBMs)

- In a full Boltzmann machine, each node is connected to every other node and he the connections grow **exponentially.** This is the reason RBMs are used. restrictions in the node connections in RBMs are as follows :-
  - Hidden nodes cannot be connected to one another.
  - Visible nodes connected to one another.
- A restricted term means that one is not permitted to connect two types of layers are of the same type to one another. In other words, the two hidden layers or layers of neurons are unable to form connections with one another. However, th may be connections between the apparent and hidden layers.
- Since there is no output layer in the machine, it is unclear how the weights wo be detected and modified, and determine whether or not the prediction was ri **One response fits all the questions : Restricted Boltzmann Machine.**
- The neural network that is a part of the energy-based model is called RBM. It generative, unsupervised, probabilistic deep learning algorithm. Finding the probability distribution that maximizes the log-likelihood function is the goa RBM. RBM only has two layers : the input layer and the hidden layer and undirected. All of the hidden nodes are linked to all of the visible nodes. RBM also referred to as an asymmetrical bipartite graph since it has two layers : a visi or input layer and a hidden layer. The visible nodes don't have any connecti inside the same layer. The concealed nodes are not connected intralayer either. O the input and hiding nodes have connections.

- All of the nodes in the original Boltzmann machine are connected. RBM is referred to as a Restricted Boltzmann Machine since it restricts intralayer connectivity.
- RBMs do not modify their weights through backpropagation and gradient descent since they are undirected. They change their weights using a technique known as contrastive divergence. The visible nodes' weights are initially created at random and utilized to create the hidden nodes. Then, these concealed nodes recreate exposed nodes using the same weights. All throughout, the same weights were utilized to reconstruct the visible nodes. Due to their lack of connectivity, the created nodes are different from one another.

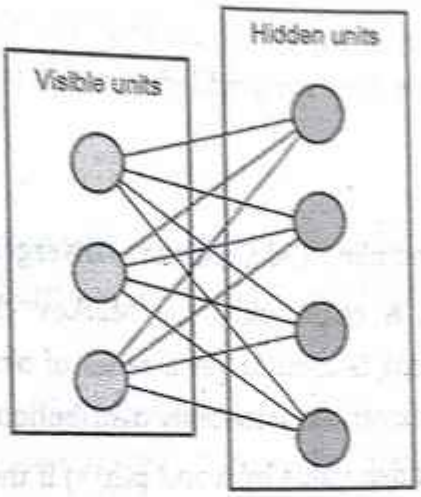


Fig. 3.3.3 RBMs

- Some key characteristics of the Restricted Boltzmann machine are :
  - There are no connections between the layers.
  - They employ symmetric and recurring structures.
  - It is an algorithm for unsupervised learning, meaning that it draws conclusions from the input data without labeled replies.
  - In their learning process, RBMs attempt to link low energy states with high probability ones and vice versa.

Working of RBM

- A low-level feature from a learning target item in the dataset is used by each visible node. The hidden layer's node 1 multiplies x by weight and adds it to a bias. These two procedures' outcomes are fed into an activation function, which, given an input of x, creates the output of the node, or the signal strength traveling through it.

- One can examine how many inputs would mix at a single hidden node. The o of the node is created by multiplying each x by a distinct weight, summi products, adding the sum to a bias and then passing the final result once agai an activation function.
- Each input x is multiplied by its corresponding weight w at each buried no other words, a single input x would have three weights in this situation, totali weights (4 input nodes x 3 hidden nodes). The weights between the two layer always create a matrix with input nodes in the rows and output nodes i columns.
- The four inputs are sent to each hidden node, multiplied by each weight, hidden node receives one output as a result of the activation algorithm afte sum of these products is once more added to a bias (which compels at least activations to occur).

Training of RBM

- Two methods - Gibbs sampling and Contrastive divergence are used to train RB
- When direct sampling is challenging, the Markov chain Monte Carlo me known as Gibbs sampling is used to get a series of observations that are rou drawn from a given multivariate probability distribution.
- The prediction is the hidden value by h and p(h|v) if the input is represented b P(v|h) is utilized for the regenerated input values' prediction when the hid values are known. Let's say that after k rounds, v k is acquired from input valu after this process has been performed k times.
- In order to approximate the graph slope, a graphical slope showing the relation between a network's errors and its weights is called the gradient in contra divergence. Contrastive divergence is a rough maximum-likelihood lear approach and is employed when one needs to approximate the learning gradie the algorithm and choose which direction to go in because one cannot dir evaluate a set of probabilities or a function.
- Weights are updated on CD. The gradient is first determined from the reconstru input, and the old weights are updated by adding the delta.

Advantages of RBM

- The hidden layer's activations can be included in other models as valuable features to boost performance.
- Due to the limitations on connections between nodes, it is faster than a standard Boltzmann machine.
- Efficiently computed and expressive enough to encode any distribution.

Disadvantages of RBM

- The backpropagation algorithm is more well known than the CD-k algorithm, which is utilized in RBMs.
- Because it is challenging to calculate the energy gradient function, training is more challenging.
- Weight modification.

Applications of RBMs

- Restricted Boltzmann machines are the most successful type of Boltzmann machines. They have been used in a wide range of applications.

i. Dimensionality reduction

- The RBM's architecture is convenient for such type of problems since their hidden nodes are of lower dimension than their visible nodes, hence they have a reduced representation of the data.

ii. Data reconstruction

- RBM is an undirected graphical model. However, it can be shown to be equivalent to a directed graphical model with infinite number of layers.

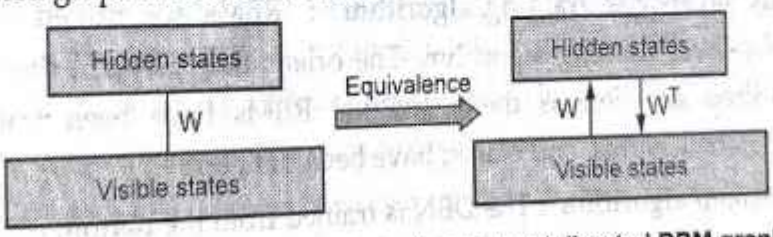


Fig. 3.3.4 Equivalence between undirected and directed RBM graph

- With this idea in mind, and with the idea that Auto-encoders (AE) are used to reconstruct data, one can unfold an RBM to approximate the AE architecture to perform the same task.

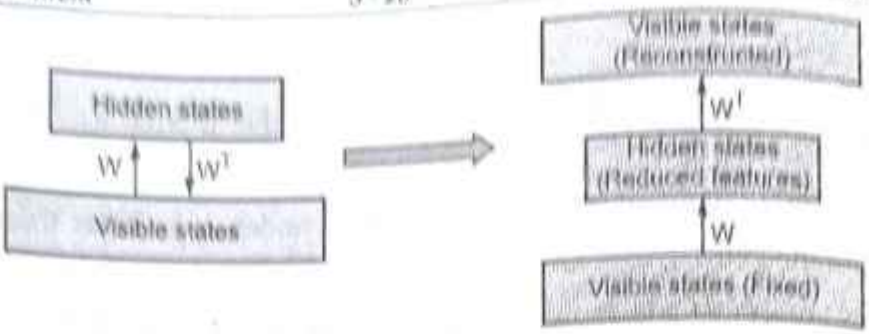


Fig. 3.3.5 Using RBM to approximate auto-encoder

iii. Recommendation engines

- One of the most known applications for RBM is collaborative filtering where the network should provide a recommendation to the end user.
- The basic idea behind recommender systems is to create a different training instance and a different RBM for each user, depending on the ratings provided by the users. All RBMs share weights.
- It is also important to notice that the ratings range from 1 to 5, hence one should use softmax nodes instead of sigmoid nodes, in order to correspond to the rating values.

2. Deep Belief Networks (DBNs)

- Consider stacking numerous RBMs so that the outputs of the first RBM serve as the input for the second RBM, and so forth. Deep Belief Networks are the name given to these networks. Each layer's connections are undirected (as each layer is an RBM). Those between the strata are simultaneously directed (except for the top two layers – whose connections are undirected). The DBNs can be trained in two different ways :
  - **Greedy layer-wise training algorithm** : RBMs are trained using a greedy layer-by-layer training algorithm. The orientation between the DBN layers is established as soon as the individual RBMs have been trained (i.e., the parameters, weights, and biases, have been defined).
  - **Wake-sleep algorithm** : The DBN is trained from the bottom up using a wake-sleep algorithm (connections going up indicate wake), and then from the bottom up using connections indicating sleep.

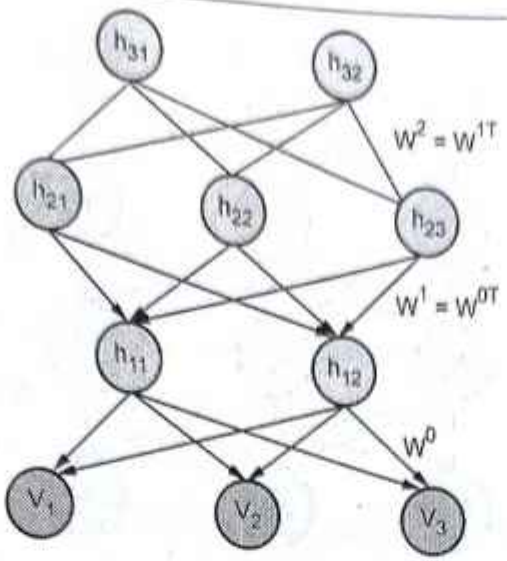


Fig. 3.3.6 DBNs

- Deep Belief Networks are pre-trained using a greedy algorithm like all other networks. The algorithm applies a layer-by-layer approach to learn the top-down weights, and these weights determine the dependencies between each layer. In this network, numerous steps of Gibbs sampling run on the top two hidden layers. This step essentially creates a sample from the Restricted Boltzmann Machine defined by the top two hidden layers. Later, a single pass of the previous sampling is used to find a piece from the visible layer throughout the rest of the model.
- In order to ensure that the layer connections only work downwards, the RBMs can be stacked, trained, and then do so (except for the top two layers).

3. Deep Boltzmann Machines (DBMs)

- Similar to DBNs, DBMs also have undirected connections between the layers in addition to the connections inside the levels (unlike DBN, in which the layer connections are directed). DBMs can be utilized for more challenging jobs since they can extract more sophisticated or complex features.
- The Deep Boltzmann Machine works by greedy layer-by-layer training. It depends on learning stacks of restricted Boltzmann machines. In layer-wise training, increase the input twice for lower-level and top-level Restricted Boltzmann Machine. The lower level Restricted Boltzmann Machine inputs are doubled to compensate for the deficiency of top-down information into the first hidden layer. Likewise, the hidden units are doubled for the top-level Restricted Boltzmann Machine to pay for the lack of bottom-up input.

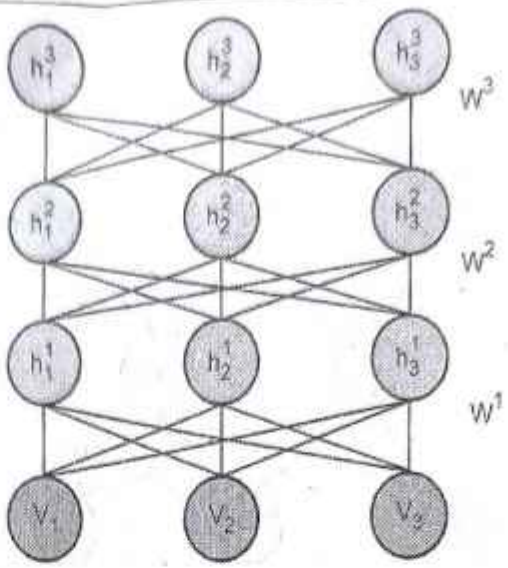


Fig. 3.3.7 DBMs

4. Conditional Boltzmann Machines

- Boltzmann machines are used to model distributions of data vectors. In order to do so they learn automatically the existing patterns in the given data which makes them more suitable for unsupervised learning tasks. However, they can also be used in **supervised learning** tasks where the goal is to predict the output that is conditioned on the input.
- The equations will be the same, the only difference is in the learning process. When generating the **model-centric samples,** instead of randomly initializing the visible and the hidden nodes, a **subset** from the **visible** nodes will be **clamped.** The latter will act as the inputs and the remaining visible nodes will act as outputs.
- Now the network will be maximizing the log probability of the observed output conditioned on the input vectors (the clamped visible nodes).

Higher-order Boltzmann Machines

- Boltzmann machine is a non-linear network that models interactions between pairwise stochastic binary nodes. However, the learning algorithm of a BM can be extended to higher order interaction. This type of networks is used to faster the learning process. For example for order $l = 3$, there is a weight w_{ijk} that is shared between three nodes as shown in the Fig. 3.3.8 below.

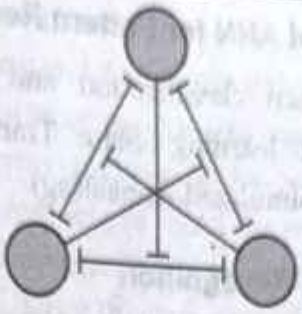


Fig. 3.3.8 Third order interaction between three nodes

- Since a Boltzmann machine is symmetric, it should have :

$$w_{ijk} = w_{jik} = w_{ikj} = w_{kji}$$

For simplicity reasons the biases in the below equations are ignored.

The equation of the energy function E will become :

$$E = -\frac{1}{3}\sum_i\sum_j\sum_k w_{ijk} s_i s_j s_k$$

In Gibbs sampling, the conditional probability will be computed just like in standard BM, by applying the **sigmoid** function on the energy gap for node **i**. However the equation of the latter will be modified like the following :

$$\Delta E_i = \sum_j\sum_k w_{ijk} s_j s_k$$

Finally, the update rule will be the same except for the term $s_i s_j$ that will be replaced by $s_i s_j s_k$.

3.3.7 Limitations of Boltzmann Machines

- Boltzmann machines suffer from many limitations that make them unsuitable for practical use.
  - The learning process is very slow especially when the weights are large and the equilibrium distribution is multimodal.
  - The generation of data using Boltzmann machine is challenging compared to other generative models, because of the cyclic dependency between the nodes.
  - Boltzmann machine suffer from the **variance-trap** that causes the activations to saturate. This is harmful for the network because the nodes no longer outputs values between **0** and **1**.

3.4 Basic Functional Units of ANN for Pattern Recognition Tasks

- Pattern association, pattern classification and pattern mapping tasks (Using Boltzmann machine and learning, State Transition Diagram, Flase Minima, Stochastic Update, Using Simulated Annealing)

3.4.1 Introduction to Pattern Recognition

- Human problem solving is basically a pattern processing problem and not a data processing problem. In any pattern recognition task humans perceive patterns in the input data and manipulate the pattern directly.
- Pattern recognition is a process of finding regularities and similarities in data using machine learning data. These similarities can be found based on statistical analysis, historical data, or the already gained knowledge by the machine itself.
- A pattern is a regularity in the world or in abstract notions. If music is being discussed, a description of a type would be a pattern. If a person keeps watching music videos, Video playing website wouldn't recommend the person sports videos.
- The act of recognition can be divided into two broad categories : recognizing concrete items and recognizing abstract items. The recognition of concrete items involves the recognition of spatial and temporal items. Examples of spatial items are fingerprints, weather maps, pictures and physical objects. Examples of temporal items are waveforms and signatures. Recognition of abstract items involves the recognition of a solution to a problem, an old conversation or argument, etc. In other words, recognizing items that do not exist physically.

Features of pattern recognition

- Pattern recognition system should recognize familiar patterns quickly and accurate
- Recognize and classify unfamiliar objects
- Accurately recognize shapes and objects from different angles
- Identify patterns and objects even when partly hidden
- Recognize patterns quickly with ease and with automaticity.

3.4.2 Pattern Recognition Tasks

- The inherent differences in information handling by human beings and machines in the form of patterns and data and in their functions in the form of understanding and recognition. Below are pattern recognition tasks,

1. **Pattern association** - Pattern association problem involves storing a set of patterns or a set of input-output pattern pairs in such a way that when test data are presented, the pattern or pattern pair corresponding to the data is recalled. This is purely a memory function to be performed for patterns and pattern pairs. Typically, it is desirable to recall the correct pattern even though the test data are noisy or incomplete. The problem of storage and recall of patterns is called autoassociation. Since this is a content addressable memory function, the system should display accretive behaviour, that is, should recall the stored pattern closest to the given input. It is also necessary to store as many patterns or pattern pairs as possible in a given system. Printed characters or any set of fixed symbols could be considered as examples of patterns for these tasks. Note that the test patterns are the same as the training patterns, but with some noise added or some portions missing. In other words, the test data are generated from the same source in an identical manner as the training data.

2. **Pattern mapping** - In pattern mapping, given a set of input patterns and the corresponding output pattern or class label, the objective is to capture the implicit relationship between the patterns and the output, so that when a test input is given, the corresponding output pattern or the class label is retrieved. Note that the system should perform some kind of generalization as opposed to memorizing the information. This can also be viewed as a pattern classification problem belonging to supervised learning category. Typically, in this case the test patterns belonging to a class are not the same as the training patterns, although they may originate from the same source. Speech spectra of steady vowels generated by a person or hand-printed characters, could be considered as examples of patterns for pattern mapping problems. Pattern mapping generally displays interpolative behaviour, whereas pattern classification displays accretive behaviour.

3. **Pattern grouping** - In this case, given a set of patterns, the problem is to identify the subset of patterns possessing similar distinct features and group them together. Since the number of groups and the features of each group are not explicitly stated, this problem belongs to the category of unsupervised learning or pattern clustering. Note that this is possible only when the features are unambiguous as in the case of hand-printed characters or steady vowels. In the pattern mapping problem the patterns for each group are given separately, and the implicit, although distinct, features have to be captured through the mapping. In pattern grouping on the other hand, patterns belonging to several groups are given, and the system has to resolve the groups.

4. **Feature mapping** - In several patterns the features are not unambiguous. In fact the features vary over a continuum, and hence it is difficult to form groups of patterns having some distinct features. In such cases, it is desirable to display the feature changes in the patterns directly. This again belongs to the unsupervised learning category. In this case what is learnt is the feature map of a pattern and not the group or class to which the pattern may belong. This occurs, for example, in the speech spectra for vowels in continuous speech. Due to changes in the vocal tract shape for the same vowel occurring in different contexts, the features (formants or resonances of the vocal tract in this case) vary over overlapping regions for different vowels.
5. **Pattern variability** - There are many situations when the features in the pattern undergo unspecified distortions each time the pattern is generated by the system. This can be easily seen in the normal handwritten cursive script. Human beings are able to recognize them due to some implicit interrelations among the features, which themselves cannot be articulated precisely. Classification of such patterns falls into the category of pattern variability task.
6. **Temporal patterns** - Human beings are able to capture effortlessly the dynamic features present in a sequence of patterns. This is true, for example, in speech where the changes in the resonance characteristics of the vocal tract system (e.g. formant contours) capture the significant information about the speech message. This is also true in any dynamic scene situation. All such situations require handling sequences of static patterns simultaneously, looking for changes in the features in the subpatterns in adjacent pattern pairs.
7. **Stability-plasticity dilemma** - In any pattern recognition task the input patterns keep changing. Therefore it is difficult to freeze the categorization task based on a set of patterns used in the training set. If it is frozen, then the system cannot learn the category that a new pattern may suggest. In other words, the system lacks its plasticity. On the other hand, if the system is allowed to change its categorization continuously, based on new input patterns, it cannot be used for any application such as pattern classification or clustering, as it is not stable. This is called stability-plasticity dilemma in pattern recognition.

3.4.3 Input Processing for Pattern Recognition

- Recognition of an item involves three levels of processing : input filtering, feature extraction and classification

- **Filtering** is removing unwanted information or data from input. Depending on the application, the filter algorithm or method will change. For example, consider finger print identification. Each time one scans the fingerprints through a (non-ink) fingerprint device, the scanned output may be different. The difference may be due to a change in contrast or brightness or in the background of the image. There could be some distortion. In order to process the input, one may need only lines in the fingerprints and one may not need the other parts or background of the fingerprint. In order to filter out the unwanted portion of the image and replace it with a white background, one needs a filter mechanism. Once the image is filtered through the filter mechanism, one will get standard clean finger prints only with lines, which in turn helps with the process of feature extraction.
- **Feature extraction** is a process of studying and deriving useful information from the filtered input patterns. The derived information may be general features, which are evaluated to ease further processing. For example, in image recognition, the extracted features will contain information about gray shade, texture, shape or context of the image. This is the main information used in image processing. The methods of feature extraction and the extracted features are application dependent.
- **Classification** is the final stage of the pattern recognition. This is the stage where an automated system declares that the inputted object belongs to a particular category. There are many classification methods in the field. Classification method designs are based on the following concepts.

  i. **Member-roster concept -** Under this template-matching concept, a set of patterns belonging to a same pattern is stored in a classification system. When an unknown pattern is given as input, it is compared with existing patterns and placed under the matching pattern class.

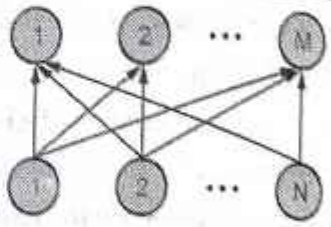
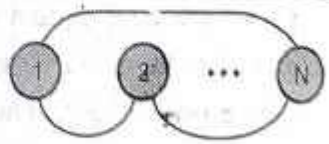
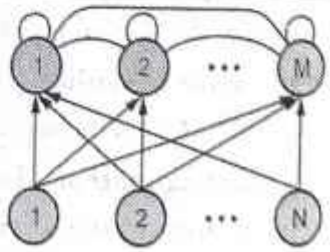
  ii. **Common property concept -** In this concept, the common properties of patterns are stored in a classification system. When an unknown pattern comes inside, the system checks its extracted common property against the common properties of existing classes and places the pattern/object under a class, which has similar, common properties.

  iii. **Clustering concept -** Here, the patterns of the targeted classes are represented in vectors whose components are real numbers. So, using its clustering properties, one can easily classify the unknown pattern. If the target vectors are far apart in geometrical arrangement, it is easy to classify the unknown patterns. If they are nearby or if there is any overlap in the cluster arrangement,

one needs more complex algorithms to classify the unknown patterns. One simple algorithm based on the clustering concept is minimum distance classification. This method computes the distance between the unknown pattern and the desired set of known patterns and determines which known pattern is closest to the unknown and, finally, the unknown pattern is placed under the known pattern to which it has minimum distance. This algorithm works well when the target patterns are far apart.

3.4.4 Functional Units of Pattern Recognition and Working of ANN

- Pattern recognition systems consist of four functional units - A feature extractor (to select and measure the representative properties of raw input data in a reduced form), a pattern matcher (to compare an input pattern to reference patterns using a distance measure), a reference templates memory (against which the input pattern is compared), and a decision maker (to make the final decision as to which reference template is the closest to the input pattern).
- Below are ANN that are used in pattern recognition

1. Feedforward ANN a) Pattern association b) Pattern classification c) Pattern mapping / classification	
2. Feedback ANN a) Auto association b) Pattern storage (LTM) c) Pattern environment storage (LTM)	
3. Feedforward and Feedback ANN a) Pattern storage (STM) b) Pattern clustering c) Features map	

3.4.4.1 Pattern Recognition Tasks by Feedforward ANN

Pattern association

- Arch : Two layers, linear processing unit, single set of weights
- Learning : Hebb (orthogonal) rule, Delta (linearly independent) rule
- Recall : Direct
- Limitation : Linear independence, # patterns restricted to dimensionality
- To overcome : Non-linear processing unit, becomes a pattern classification problem

Pattern classification

- Arch : Two layers, non-linear processing unit, geometrical interpretation
- Learning : Delta rule
- Recall : Direct
- Limitation : Linearly separable functions, hard problems
- To overcome : More layers, hard learning problems

Pattern mapping / classification

- Arch : Multilayer(hidden), non-linear units, geometric interpretation
- Learning : Generalized delta rule - backpropagation
- Recall : Direct
- Limitation : Slow learning
- To overcome : More complex architectures

i. **Pattern association -** The objective is to design a linear network that can capture the association in the pairs of vectors (a~,bz), $l = 1, 2... L$, through a set of weights to be determined by a learning or training law. The input data used in training are typically generated synthetically, like machine printed characters. The input data used for recall may be corrupted by external noise.

- The network consists of a set of weights connecting the two layers of processing units, the output function of each unit being linear. Such a network is called a linear associator network. Due to linearity of the output function of each unit, the activation values and the output signals of the units in the input layer are same as the input data values. The activation value of the ith unit in the output layer is given by,

$$y_i = \sum_{j=1}^{N} w_{ij} a_{ij}$$

- The output of the ith unit is the same as its activation value y_i, since the output function of the unit is linear. The objective is to determine a set of weights w_{ij} in such a way that the actual output b'_{li} is equal to the desired output b_{li} for all the L pattern pairs.
- If the input L pattern vectors $\{a_l\}$ are all orthogonal, then it is possible to use Hebb's learning law to determine the optimal weights of the network. It should be noted that a learning law enables updating of weights as patterns are applied one by one to the network. The optimality of the weights is determined by minimizing the mean squared error between the desired and the actual output values. The optimal weights after I pattern pairs are fed to the network are given by,

$$w_{ij}^{(l)} = w_{ij}^{l-1} + b_{li}\, a_{lj}, \quad w_{ij}^{0} = 0.$$

- The final optimal weights for pattern association task are given by,

$$w_{ij} = w_{ij}^{L}$$

- If the input vectors {as} are only linearly independent, but not necessarily orthogonal, then the optimal weights that minimize the mean squared error can be obtained using the LMS learning law.
- Once the network is trained, for any given input pattern a_l, the associated pattern b_l can be recalled using the equations,

$$y_i = \sum_{j=1}^{N} w_{ij}\, a_{lj} \quad \text{and} \quad b_{li} = y_i$$

- When noisy input patterns are used during recall, i.e., $\{a_{li}\}$, then the recalled pattern $\{b'_{li}\}$ will also be noisy. Since the given set of input pattern $\{a_l\}$, $l = 1 \ldots L$, is assumed to be linearly independent, the number of patterns in the input set is limited to the dimensionality of the input vector, namely, N. Therefore, it is not possible to store more than N pattern pairs in a linear associative network. If the number of input pattern are more than its dimension (N) or if the input set (even for $L < N$) are not linearly independent, then the resulting weight vectors are not optimal any more. In such a case the recall of the associative pattern for a given input pattern may not be correct always.

- Even if the input patterns are linearly independent and optimal weights are used, the recall may be in error if a noisy input pattern is presented to recall the associated pattern.

ii. **Pattern classification :** In an N-dimensional space if a set points could be considered as input patterns without restriction on their number, and if an output pattern, not necessarily distinct, is assigned to each of the input patterns, then the number of distinct output patterns can be viewed as distinct classes or class labels for the input patterns. Since there is no restriction on the type and number of input patterns, the input-output pattern pairs (a_l, b_l), $l = 1, 2, \ldots L$ in this case can be considered as a training set for a pattern classification problem. Typically for pattern classification problems the output patterns are points in a discrete (normally binary) M-dimensional space. The input patterns are usually from natural sources like speech and hand-printed characters. The input patterns may be corrupted by external noise at the time of recall.

- A two layer network with non-linear (threshold or hard limiting) output function for the units in the output layer, can be used to perform the task of pattern classification. This may also be identified as a single layer perceptron network.
- The network can be trained (i.e. weights can be adjusted) for the given set of input-output patterns using a delta rule.
- The corresponding learning is also called perceptron learning law.
- The training patterns are applied several times, if needed, until the weights do not change appreciably. But there is no guarantee that the weights converge to some stable values. Convergence of the weights depends on whether the problem specified by the input-output pattern pairs is representable or not by a network of this type. For all representable problems the learning law converges.
- During recall, a pattern generated from one of the same sources is given as input. By direct computation of the weighted sum of the input, the network determines the pattern class to which the input belongs. The network thus exhibits accretive behaviour. Even when the input pattern is noisy, the output class may still be correct, provided the noise has not significantly altered the input pattern.

iii. **Pattern mapping** : For a pattern mapping problem input and output patterns are points in the N- and M-dimensional continuous spaces, respectively. The objective is to capture the implied functional relationship or mapping function between the input and output by training a feedforward neural network. This is also called the generalization problem. Once the network generalized by capturing the mapping function through its weights, then during recall from an input pattern the network produces an output which is an interpolated version of the outputs of the training input patterns near the current input pattern. The input patterns are generally naturally occurring patterns as in speech and hand printed characters.

- A multilayer feedforward network with at least two intermediate layers in addition to the input and output layers can perform a pattern mapping task. The number of units in the input and output layers correspond to the dimensions of the input and output patterns, respectively. The additional layers are called hidden layers and the number of units in a hidden layer is determined depending on the problem, usually by trial and error. The network can be trained (i.e. weights at different layers can be adjusted) for a given set of input-output pattern pairs using a generalized delta rule or backpropagation law.
- It is derived using the principle of gradient descent along the error surface in the weight space. The given patterns are applied in some random one by one, and the weights are adjusted using the backpropagation law. The pattern pairs may have to be applied several times till the output error is reduced to an acceptable value.
- Once the network is trained, it can be used to recall the appropriate pattern (in this case some interpolated) for a new input pattern. The computation is straight forward in the sense that the weights and the output functions of the units at different layers are used to compute the activation values and output signals. The signals from the output layer correspond to the output.

3.4.4.2 Pattern Recognition Tasks by Feedback ANN

Auto association (pattern storage)

- Arch : Single layer with feedback, linear processing units
- Learning : Hebb (orthogonal inputs), Delta (linearly independent inputs)
- Recall : Direct

- Limitation : Linear independence of patterns, # of patterns limited to dimensionality
- To overcome : Non-linear processing units, becomes a pattern storage problem.

Pattern storage

- Arch : FBNN, non-linear processing units, states, Hopfield energy analysis
- Learning : Not important
- Recall : Activation dynamics until stable states are reached
- Limitation : False minima, hard problems, limited # patterns
- To overcome : Stochastic update, hidden units.

Pattern environment storage

- Arch : Boltzmann machine, nonlinear processing units, hidden units, stochastic update
- Learning : BM learning law, simulated annealing
- Recall : Activation dynamics, simulated annealing
- Limitation : Slow learning
- To overcome : Different architecture

i. **Auto association** - auto association is the task where in the input and output patterns in each pair are the same i.e., $a_l = b_l$, $l = 1, 2, \ldots, l$. The objective in an auto association task is to design a network that can recall a stored pattern given a corrupted (noisy or partial) version of the pattern. A feedback network with N linear processing units can perform the task of auto association. Such a network can be trained (i.e., the weights can be determined) using either Hebb's law or delta rule. Hebb's learning law leads to a set of optimal weights when the given patterns are orthogonal. Delta rule leads to set of optimal weights when the given patterns are linearly independent. Pattern recall will be exact when the test pattern is same as one of the stored ones, represented by the weights. If the test pattern is a noisy version of the stored pattern, the recalled pattern is also a noisy version of the stored pattern. In fact the network recalls the input pattern itself, as every vector is associated with itself, thus completely eliminating any accretive behavior.

- Thus auto association by a feedback network with linear units is not going to serve any purpose. Moreover, the number of patterns is limited to the dimensionality of the pattern. Although there is no simple learning law, it can be shown that the weights of such a network can be determined to store any L <= N patterns, without any error in recall, where N is the dimension of the input pattern space. The auto association task by a feedback network

with linear units has, any input pattern comes out as itself if it is one of the stored ones and a noise input comes out as a noisy pattern, not as the nearest stored pattern. To overcome this limitation due to the absence of accretive behaviour, the linear units are replaced with units having nonlinear output functions. The resulting feedback network can then perform pattern storage task.

ii. **Pattern storage :** The objective is to store a given set of patterns so that any one of the patterns can be recalled exactly when an approximate (corrupted) version of the pattern is presented to the network. What is needed is the storage of features and their spatial relations in the patterns and the pattern recall should take place even when the features and their relations are slightly modified due to noise and distortion. The approximation of pattern refers to the closeness of the features and their spatial relations in the pattern when compared to the original stored pattern. What is actually stored in practice is the information in the pattern data itself. The approximation is measured in terms of some distance, like Hamming distance (in case of binary patterns.) The distance features is automatically realized through the threshold (binary) features of the output function of a processing unit. The pattern storage is accomplished by a feedback network consisting of nonlinear processing units.

- For the simplest case, the weights on the connecting links between units are assumed to be symmetric, i.e. $w_{ij} = w_{ji}$ and that there is no self feed back, i.e. $w_{ii} = 0$. The output signals of all units at any instant of time define the state of the network at that instant. Each state of the network can be assumed to correspond to some energy the instant. Each state of the network can be assumed to correspond to some energy which is defined in terms of the output state $\{s_i\}$ and weights $[w_{ij}]$ of the network as

$$E = -\frac{1}{2}\sum_i \sum_{j \neq 1} s_i w_{ij} s_j - \sum_i I_i s_i + \sum_i \theta_i s_i$$

where I_i is an external input and θ_i is the threshold of the unit. The energy as a function of the output state can be viewed as something like an energy landscape. The shape of the landscape is dictated by the network units and their interconnection strengths (weights). The feedback and the non-linear processing units of the network create *basins of attraction* in the energy landscape. The basins tend to be region of equilibrium states. If there is a fixed state (point in the output state space) in each of these basins where the energy is minimum, these states corresponds to fixed points of

equilibrium. There could also be periodic (or oscillating) regions or chaotic regions of equilibrium.

- It is the existence of the basins of attraction that is exploited to store the desired patterns and recall them even with approximate inputs as keys. Each pattern is stored at a fixed point of equilibrium of the energy minimum. An erroneous or distorted pattern is more likely to be closer to the corresponding true pattern than to the other stored patterns. Each input pattern results in a state of the network that may be closer to the desired state, in the sense that it may lie near the basin of attraction corresponding to the true state. Since an arbitrary state need not correspond to a stable state, the activation dynamics of the network may eventually lead to a stable state from which the desired pattern may be read or derived.
- Hopfield Net algorithm to store and recall a set of bipolar patterns.

Backpropagation algorithm : Generalized delta rule

Given a set of input-output patterns $a_l, b_l = 1, 2, \ldots, L$

l^{th} input vector $a_l = (a_{l1}, a_{l2}, \ldots, a_{lN})^T$ and output vector $b_l = (b_{l1}, b_{l2}, \ldots, b_{lN})^T$

Assume only one hidden layer and initial setting of weights to be arbitrary

Assume input layer with only linear units. Then output signal = input activation value

η is the learning rate parameter.

Activation of unit i in the input layer $x_{li} = a_{li}$

Activation of unit j in the hidden layer $x_{lj}^h = \sum_{i=l}^{N} w_{ji}^h x_{li} + \theta_j^h$

Output signal from the jth unit in the hidden layer, $s_{lj}^h = f_j^h (x_{lj}^h)$

Activation of unit k in the output layer $x_{lk}^0 = \sum_{j=1}^{H} w_{kj}^0 s_{lj}^h + \theta_k^0$

Output signal from unit k in the output layer $s_{lk}^0 = f_k^0 (x_{lk}^0)$

Error term for the kth output unit $\delta_k^0 = (b_{lk} - s_{lk}^0) f_k^0 (x_{lk}^0)$

Update the weights on the output layer $w_{kj}^0 (t+1) = w_{kj}^0(t) + \eta \delta_{lk}^0 s_{lj}^h$

Error term for the jth hidden unit $\delta_{lj}^h = f_j^{h'} (x_{lj}^h) \sum_{k=1}^{M} \delta_{lk}^0 w_{kj}^0$

Update the weights on the hidden layer $w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta\delta_{lj}^{h} a_{li}$

Calculate the error for the lth pattern $E_l = \frac{1}{2}\sum_{k=1}^{M} \delta_{lk}^{2}$

Total error for all patterns $E = \sum_{l=1}^{L} E$

Apply the given patterns, may be several times, in some random order and update the weighs until the total error reduces to an acceptable value.

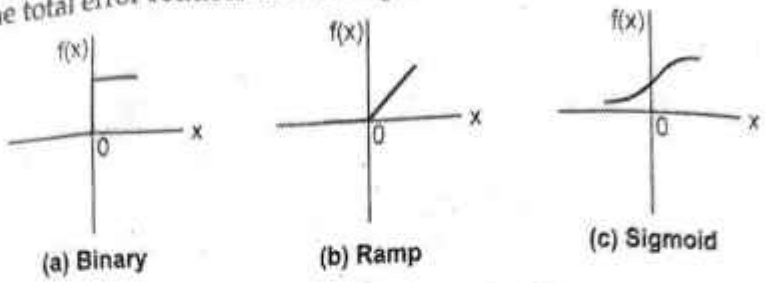
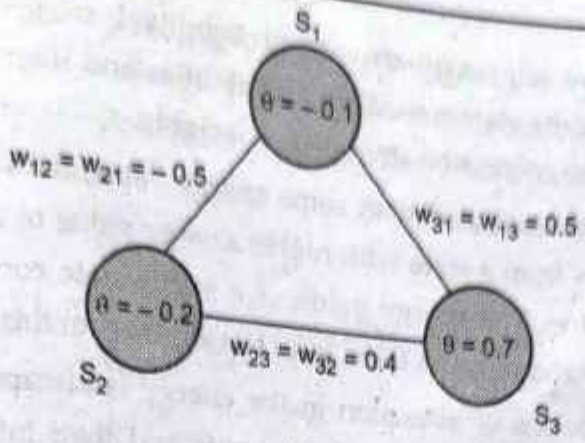


Fig. 3.4.1 Some nonlinear functions

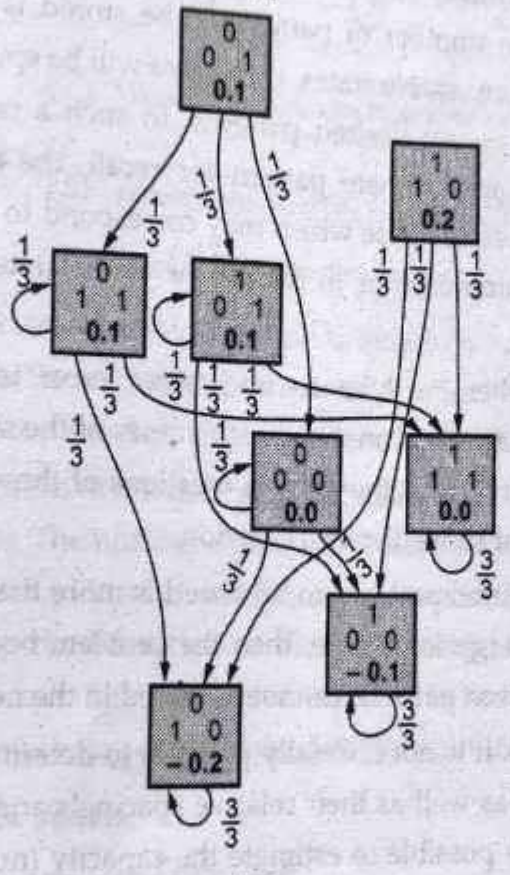
- If the nonlinear output function of each unit is a binary threshold function (curve a in above figure), then the stable stables of the network would lie at the corners of the binary hypercube in the N-dimensional discrete binary space. On the other hand, if the output function is a semilinear function (curve C in above figure), then the points corresponding to these states may move closer to each other within the unit hypercube. If the output function is a horizontal line, then almost all states remain close to each other, and hence there will be only one state for the network.

State transition diagram

- A transition diagram or state transition diagram is a directed graph, **used to represent a finite state machine.** These are used to model objects which have a finite number of possible states and whose interaction with the outside world can be described by its state changes in response to a finite number of events.
- Given a network, it is possible to determine the state transition diagram. Fig. 3.4.2 below shows the state transition diagram for a 3-unit network.
- The diagram illustrates the different states of the network and their transition probabilities.



(a)



(b)

Fig. 3.4.2 (a) A 3-unit feedback network with symmetric weights and binary threshold units. Activation dynamics $x_j = \sum_j w_{ji} s_i - \theta_j$, $s_j = f(s_j)$; Energy $E = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j + \sum_i s_i \theta_i$. (b) State transition diagram for the 3-unit network of Fig. (a). Each block represents a state given by sequence s_1, s_2, s_3. There are eight blocks for eight states. The energy for each state is indicated by the bold numbers with each block. Note that the state diagram has three stable states (111,100 and 010)

- States which have self-transition with probability 1 are stable states. For a given number of units, the state transition probabilities and the number or stable states are dictated by the connection strengths or weights.
- Since each state is associated with some energy value, the state transition diagram shows transitions from a state with higher energy value to a state having lower or equal energy value. The energy value of a stable state corresponds to an energy minimum in the landscape, as there is no transition from this to the other states.
- The number of basins of attraction in the energy landscape depends only on the network, i.e., the number of processing units and their interconnection strengths (weights). When the number of patterns to be stored is less than the number of basins of attraction, i.e., stable states, then there will be spurious stable states, which do not correspond to any desired patterns. In such a case, when the network is presented with an approximate pattern for recall, the activation dynamics may eventually lead to a stable state which may correspond to one of the spurious states or a false energy minimum, or to one of the stable states corresponding to some other pattern.
- In the latter case there will be an undetected error in the recall. The average probability of error depends on the energy values of the stable states corresponding to the desired patterns, and the relative locations of these states in the state space, measured in terms of some distance criterion.
- If the number of desired patterns to be stored is more than the number of basins of attraction in the energy landscape, then the problem becomes a hard problem, in the sense that the given patterns cannot be stored in the network.
- For a given network it is not normally possible to determine exactly the number of basins of attraction as well as their relative spacing's and depths in the state space of the network. It is possible to estimate the capacity (number of patterns that can be stored) of the network and also the average probability of error in recall. The probability of error in recall can be reduced by adjusting the weights in such a way that the resulting energy landscape is matched to the probability distribution of the input patterns. This becomes the problem of storing a pattern environment.

iii. **Pattern environment storage :** A pattern environment is described by the set of desired patterns together with their probability distribution. The objective is to store a pattern environment in a network in such a way that the average probability of error in recall is minimized. This is achieved if the energy landscape is designed in such a way that the desired patterns are stored at the

stable states corresponding to the lowest minima, with the higher probability patterns at lower energy minima points. Boltzmann machine architecture together with the Boltzmann learning law can achieve an optimal storage of pattern environment.

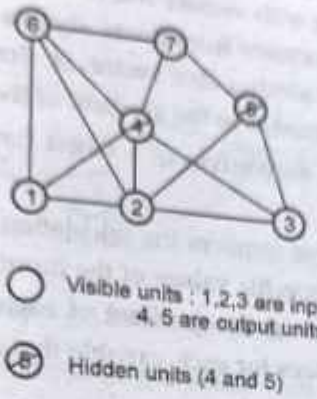


Fig. 3.4.3 Boltzmann architecture

- Each unit is connected to every other unit, although only a few connections are shown in the figure. Some of the visible units can be identified as input units and others as output units if the machine is to be used for pattern mapping.
- The architecture consists of a number of processing units with each unit connecting to all the other units. The number of units is typically larger than the dimension of the input pattern. The additional units are called hidden units. Use of hidden units helps in overcoming the limitation of the hard problems of pattern storage by a fully connected network. The patterns are applied to the so-called visible units, the number of visible units being equal to the dimension of the input patterns.

False Minima and Stochastic Update

Stochastic Update concept and gradient descent

- **Gradient descent** is an optimization algorithm that finds the set of input variables for a target function that results in a minimum value of the target function, called the minimum of the function.
- As its name suggests, gradient descent involves calculating the gradient of the target function.
- From calculus theory and first-order derivative of a function one can calculate the slope or curvature of a function at a given point. Read left to right, a positive derivative suggests the target function is sloping uphill and a negative derivative suggests the target function is sloping downhill.

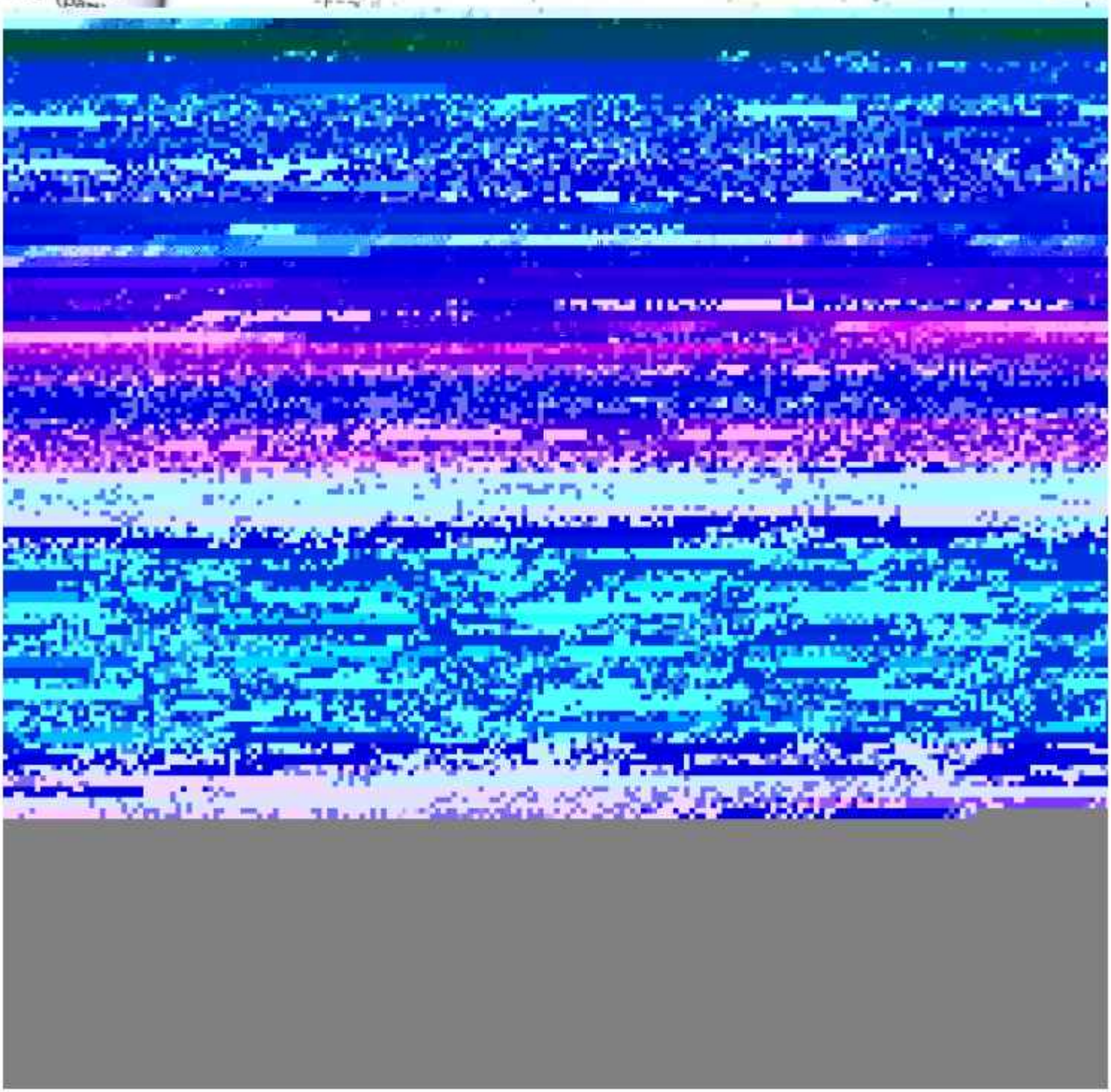
- **Derivative :** Slope or curvature of a target function with respect to specific input values to the function.
- If the target function takes multiple input variables, they can be taken together as a vector of variables. Working with vectors and matrices is referred to linear algebra and doing calculus with structures from linear algebra is called matrix calculus or vector calculus. In vector calculus, the vector of first-order derivatives (partial derivatives) is generally referred to as the gradient of the target function.
- **Gradient :** Vector of partial derivatives of a target function with respect to input variables.
- The gradient descent algorithm requires the calculation of the gradient of the target function with respect to the specific values of the input values. The gradient points uphill, therefore the negative of the gradient of each input variable is followed downhill to result in new values for each variable that results in a lower evaluation of the target function.
- A step size is used to scale the gradient and control how much to change each input variable with respect to the gradient.
- **Step size :** Learning rate or alpha, a hyperparameter used to control how much to change each input variable with respect to the gradient.
- This process is repeated until the minimum of the target function is located, a maximum number of candidate solutions are evaluated or some other stop condition.
- Gradient descent can be adapted to minimize the loss function of a predictive model on a training dataset, such as a classification or regression model. This adaptation is called stochastic gradient descent.
- **Stochastic gradient descent :** Extension of the gradient descent optimization algorithm for minimizing a loss function of a predictive model on a training dataset.
- The target function is taken as the loss or error function on the dataset, such as mean squared error for regression or cross-entropy for classification. The parameters of the model are taken as the input variables for the target function.
- **Loss function :** Target function that is being minimized.
- **Model parameters :** Input parameters to the loss function that are being optimized.

The algorithm is referred to as "stochastic" because the gradients of the target function with respect to the input variables are noisy (e.g. a probabilistic approximation). This means that the evaluation of the gradient may have statistical noise that may obscure the true underlying gradient signal, caused because of the sparseness and noise in the training dataset.

specific input

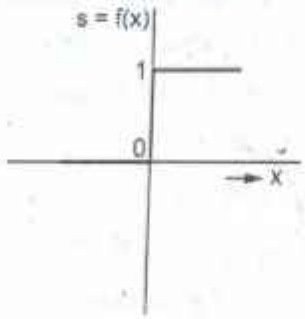
together as a
linear algebra
x calculus or
ives (pa

- Stochastic gradient descent can be used to train (optimize) many different model types, like linear regression and logistic regression, although often more efficient optimization algorithms have been discovered and should probably be used instead.

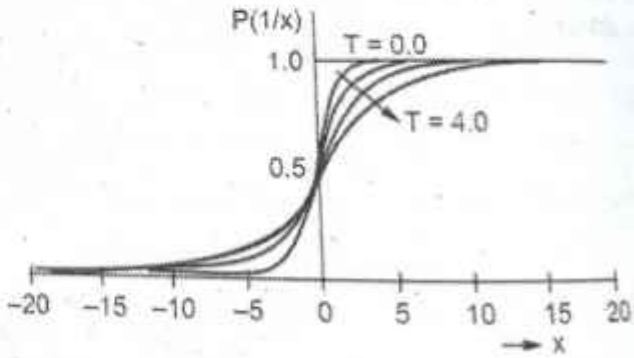


False minima problem and removing false minima

- Error in pattern recall due to false minima can be reduced significantly if initially the desired patterns are stored (by careful training) at the lowest energy minima. The remaining error can be reduced by using suitable activation dynamics. It is assumed that by training one has achieved a set of weights which will enable the desired patterns to be stored at the lowest energy minima. The activation dynamics is modified so that the network can also move to a state of higher energy value initially and then to the nearest deep energy minimum. It is possible to realize this by using a stochastic update in each unit instead of the deterministic update of the output function.
- Stochastic update means that the activation value or the net input to a unit need not decide the next output state of the unit in a deterministic manner as in the case of Hopfield backpropagation algorithm mentioned above. The update is expressed in probabilistic terms, like the probability of firing the unit being greater than 0.5 if the net input exceeds a threshold and less than 0.5 if the net input is less than the threshold for the unit. Note that the output function could still be a threshold logic (hardlimiter), but it is applied in a stochastic manner. With the new activation dynamics, the state transition diagram shows transitions from a lower energy state to a higher energy state as well, the probability of such a transition is dictated by the probability function used in determining the firing of a unit in the stochastic update.
- The probability function (in below figure) can in turn be defined in terms of a parameter, called temperature (T). As the temperature is increased, the uncertainty in the update increases, giving the network a greater chance to go to a higher energy level state.



(a) Binary output function



(b) Probability function for stochastic update

Fig. 3.4.6 Stochastic update of a unit using probability law.

Probability of firing P(1/x) = 1/[1 + exp (– x/T)]

- Since eventually there is a need of the activation dynamics to lead the network to a stable state corresponding to the pattern closest to the given input pattern, one needs to provide greater mobility for transition to higher states only initially. The mobility is slowly decreased by reducing the temperature, eventually to T = 0. At the lowest temperature the network settles down to a fixed point state corresponding to the desired pattern. At each temperature the network dynamics is allowed to settle to some equilibrium situation, called thermal equilibrium. At thermal equilibrium the average probability of visiting the states of the networks will not change further. The temperature parameter is reduced in a predetermined manner (called annealing schedule), making sure that at each temperature the network is allowed to reach thermal equilibrium before the next change in temperature is made. This process is called simulated
below algorithm. It should

different states of the network does not change any further. This is usually accomplished only approximately.

7. Lower the temperature and repeat step 3 through 7 until a stable state is reached at which point there will not be any further change in the state of the network. The result of recall is the stable output state of the visible units.

- It should be noted that at each temperature the state update dynamics is fixed, as the probability of transition from one state to another depends only on the temperature. The update dynamics is however altered when the temperature is changed and it results in a new state transition diagram.
- The states at thermal equilibrium at T = 0 represent the stable states of the network corresponding to the minima of the energy landscape. The probabilities of these states are also related to the actual minimum energy values of the states. The relation between the probabilities of stable states and energy suggest that the probability of error in the recall of patterns can be further reduced if the probability distribution of the desired patterns, that is, the pattern environment, is known and is used in determining the optimal setting of weights of the network.

Learning in Boltzmann machine

- The objective is to adjust the weights of a Boltzmann machine so as to store a pattern environment described by the set of vectors $\{V_a\}$ and their probabilities of occurrence. These vectors should appear as the outputs of the visible units. Define $\{H_b\}$ as the set of vectors appearing on the hidden units.
- Let $P^+(V_a)$ be the probability that the outputs of the visible units will be clamped (indicated by "+" superscript) to the vector V_a. Then,

$$P^+(V_a) = \sum_b P^+(V_a \wedge H_b),$$

where $P^+(V_a \wedge H_b)$ is the probability of the state of network when the outputs of the visible units are clamped to the vector V_a and the outputs of the hidden units are H_b.

Likewise the probability that V_a will appear on the visible units when none of the visible units are clamped (indicated by "– "superscript) is given by

$$P^-(V_a) = \sum_b P^-(V_a \wedge H_b)$$

Note that $P^+(V_a)$ is given by the pattern environment description and $P^-(V_a)$ depends on the network dynamics and is given by

$$P^-(V_a) = \sum_b \exp/(-E_{ab}/T) / \sum_{mn} \exp/(-E_{mn}/T),$$

where the total energy of the system in the state $V_a \wedge H_b$ is given by

$$E_{ab} = -\frac{1}{2}\sum_{i,j} w_{ij}\, s_i^{ab}\, s_j^{ab}$$

s_i^{ab} refers to the output of the ith unit in the state $V_a \wedge H_b$.

The Boltzmann learning law is derived using the negative gradient descent of the functional

$$G = \sum_a P^+ V_a \ln [P^+(V_a)/P^-(V_a)]$$

It can be shown that

$$-\partial G/\partial w_{ij} = (1/T)(P_{ij}^+ - P_{ij}^-)$$

where

$$P_{ij}^+ = \sum_{a,b} P^+(V_a \wedge H_b)\, s_i^{ab}\, s_j^{ab},$$

$$P_{ij}^- = \sum_{a,b} P^-(V_a \wedge H_b)\, s_i^{ab}\, s_j^{ab}$$

The weight updates are calculated according to

$$\Delta w_{ij} = -\eta(\partial G/\partial w_{ij}) = \eta(1/T)(P_{ij}^+ - P_{ij}^-).$$

- The Boltzmann law is implemented using some annealing schedule for the network during clamped and unclamped phases of the visible units of the network to determine (P_{ij}^+) and P_{ij}^-, respectively.
- The Boltzmann learning law as explained above allows to represent a given environment by the network. The law uses an information theoretic measure to evaluate how well the environment is represented in the network. If a perfect representation is obtained, then there will be as many energy minima as there are desired patterns. But in practice only an approximate representation of the environment is accomplished and hence there will be some spurious stable states which correspond to the false wells in the energy landscape.
- The Boltzmann learning law uses a simulated annealing schedule for implementation, that is, for determining the weight updates at each stage.

- Recall of stored patterns from an approximate input pattern also uses a simulated annealing schedule to overcome the false minima created because of the approximate representation of the environment by the network.
- In general the Boltzmann learning law converges slowly to the desired weights. Moreover, there is no simple way to determine the optimum number of hidden units for a network to solve the given problem of pattern environment storage. The larger the number of hidden units, the greater is the chance for more false minima and hence the greater the probability of error in recalling a stored pattern. The smaller the number of hidden units, the greater the chance that the given problem becomes hard for the network.
- New architectures are needed to overcome some of these limitations of the Boltzmann machine for the problem of pattern environment storage.

3.4.4.3 Pattern Recognition Tasks by Feedforward and Feedback ANN

- There are tasks that can be performed by a network consisting of two layers of processing units : The first layer with linear output units feeds the input pattern to the units in the second layer through a set of feedforward connections with appropriate weights. The outputs of the units in the second layer are feedback to the units in the same layer including feedback to the same unit.
- The self-feedback is usually with a positive weight (excitatory connection) and the feedback to the other units is usually with a negative weight (inhibitory connection). The weights on the feedback connections in the second layer are usually fixed. The first layer of units is called input layer and the second layer is called competitive layer. Different choices of output functions and methods of learning lead to networks for different types of competition tasks. There are three such tasks. Assuming fixed weights in the feedforward connections from the input to the competitive layer and in the feedback connections in the competitive layer, one can study the behaviour of the network for different types of output functions of the units in the competition layer.

i. **Pattern storage (short term memory) -** Consider the output functions to be linear. When an input pattern is applied, the units in the competition layer settle to a steady activation state which will remain there even after the input pattern is removed. The activation pattern will remain as long as the network is not given a different input pattern. Another input pattern will erase the previous activation state. Hence this is called short-term memory. The pattern is stored only temporarily. This pattern storage representation is only of

theoretical interest. There is no application for such a short-term memory function. However, by using a nonlinear output function for the units in the competition layer one could show that the network can perform a pattern clustering task.

ii. **Pattern clustering** - Given a set of patterns, the objective is to design a competition network which groups the patterns into subgroups of patterns based on similarity of features in the patterns. A two layer network with input and competition layers and with nonlinear units in the competition layer can perform the task of pattern clustering or grouping. If a nonlinear output function of the type $f(x) = x^2$ is used for the units in the competitive layer, then it can be shown that the activation dynamics leads to a steady state situation where the network tends to enhance the activity of the unit with the largest activity. When the input pattern is removed, the activities of all units except the largest one will decay to zero. Thus only one of the units in the competitive layer will win.

- The weights leading to the winning unit j are adjusted to respond more to the input pattern a. This weight adjustment is repeated for all the input patterns several times. For input patterns belonging to different groups, different units in the competition layer will win. When the weight vector for each output unit reaches an average position within the cluster, it will stay generally within a small region around that average position. Each unit in the competitive layer refers to a different group or category of patterns.

When an unknown input pattern is given, the activation dynamics leads to a steady state situation where only one unit in the competitive layer is active. That unit gives the category to which the input pattern belongs.

It should be noted that in a competitive network the physical location of the units does not reflect any relation between categories. But there are many situations where the patterns do not fall into fixed categories. There may be a gradual change of features from one pattern to another. This change of features can be captured by a self-organisation network which performs the task of feature mapping.

iii. **Feature map** - Given a set of patterns, the objective is to design a network that would organise the patterns in accordance with similarity of features among them in such a way that by looking at the output of the network one can visually obtain an idea of how different patterns are related. The display of

signals from the output layer (typically in 2-dimension) of units is called a feature map.

- To accomplish the task of feature mapping, a competitive network is modified into one called a self-organising network as shown below.

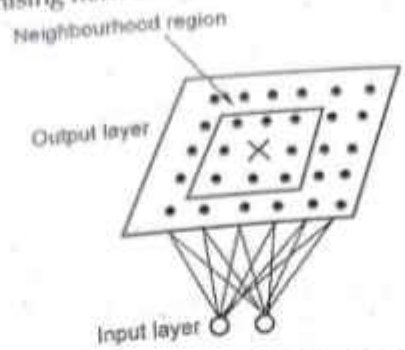


Fig. 3.4.7 A feature mapping architecture. Each input unit is connected to all the units of the output layer

- The modification consists of creating a neighbourhood region around the winning unit in the competitive layer, so that during training all the feedforward weights leading to the units in this region are adjusted to favour the input pattern. The weight adjustment is similar to the case of a competitive network. The neighbourhood region around a winning unit is gradually reduced for each application of the given set of patterns as describe in below algorithm.

Algorithm for self-organizing feature map

1. Initialize the weights from N inputs to the M output units to small random values. Initialize the size of the neighbourhood region R(0).
2. Present a new input **a**
3. Compute the distance d_i between input and the weight on each output unit i :

$$d_i = \sum_{j=1}^{N} [a_j(t) - w_{ij}(t)]^2 \text{ , for } i = 1, 2, \ldots, M$$

where $a_j(t)$ is the input to the jth input unit at time t and $w_{ij}(t)$ is the weight from the jth input unit to the ith output unit.

4. Select the output unit i^* with minimum distance

$$i^* = \text{index of } [\min_i d_i]$$

5. Update weight to node i^* and its neighbours

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(a_i(t) - w_{ij}(t)), \text{ for } i \in R_i^*(t), \text{ and } i = 1, 2, \ldots, N.$$

where $\eta(t)$ is the learning rate parameter $(0 < \eta(t) < 1)$ that decreases with time $R_i^*(t)$ gives the neighbourhood region around the node i^*, at time t.

6. Repeat steps 2 through 5 for all inputs several times.

- For recall, when an unknown input is applied, the activation dynamics determines the winning unit whose location would determine its features relative to the features represented by the other units in its neighbourhood. While a feature map produces a more realistic arrangement of patterns, the output is useful only for visual observation. Since it is difficult to categorize a feature map, it is difficult to use it for applications such as pattern classification. A more complex architecture is needed to exploit the advantages of a feature map for pattern classification purposes.

Review Questions with Answers

1. *Explain associative learning and use of Boltzmann machine in associative learning.* **(Refer section 3.1)**
2. *Discuss use of Hopfield networks in associative learning.* **(Refer section 3.1)**
3. *What is simulated annealing ? How it is useful in ANN.* **(Refer section 3.2)**
4. *What are advantages and disadvantages of simulated annealing ?* **(Refer section 3.2)**
5. *Write a note Boltzmann algorithm.* **(Refer section 3.3)**
6. *List and explain various types of Boltzmann machines.* **(Refer section 3.3)**
7. *List various uses of Boltzmann machine.* **(Refer section 3.3)**
8. *What are various pattern recognition tasks ?* **(Refer section 3.4)**
9. *Explain tasks of feedforward ANN in pattern recognition.* **(Refer section 3.4)**
10. *Explain tasks of feedback ANN in pattern recognition.* **(Refer section 3.4)**

❑❑❑

Unit IV

4 Competitive Learning Neural Network

Syllabus

Components of CL network,Pattern clustering and feature mapping network, ART networks, Features of ART models, character recognition using ART network, Self-Organization Maps (SOM): Two Basic Feature Mapping Models, Self-Organization Map, SOM Algorithm, Properties of Feature Map, Computer Simulations, Learning Vector Quantization, Adaptive Pattern Classification.

Contents

4.1 Competitive Learning and Adaptive Resonance Theory (ART)

4.2 Self-Organization Maps (SOM) - Two Basic Feature Mapping Models, Self-Organization Map, SOM Algorithm, Properties of Feature Map, Computer Simulations, Learning Vector Quantization, Adaptive Pattern Classification.

4.1 Competitive Learning and Adaptive Resonance Theory (ART)

4.1.1 Competitive Learning

4.1.1.1 Introduction to Competitive Learning

- Competitive learning is a type of unsupervised learning model used in machine learning and artificial intelligence systems.
- In this unsupervised training model output units are said to be in competition for input patterns. During training, the output unit that provides the highest activation to a given input pattern is declared the weights of the winner and is moved closer to the input pattern, whereas the rest of the neurons are left unchanged.
- This strategy is also called winner-take-all since only the winning neuron is updated. Output units may have lateral inhibitory connections so that a winner neuron can inhibit others by an amount proportional to its activation level.
- The winner-take-all networks are the networks that are based on the competitive learning rule and will use the strategy where it chooses the neuron with the greatest total inputs as a winner. The connections between the output neurons show the competition between them and one of them would be 'ON' which means it would be the winner and others would be 'OFF'.

4.1.1.2 Components of CL network

- CL network is similar to a single layer feed-forward network having feedback connection between the outputs. The connections between the outputs are inhibitory type, which means the competitors never support themselves.

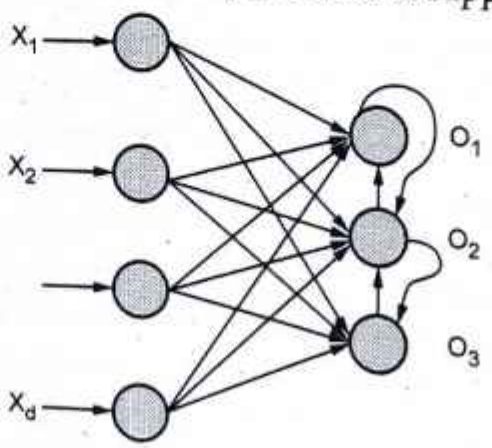


Fig. 4.1.1 Single layer network

- A simple competitive network is basically composed of two networks namely, each of them specializes in a different function.

1) Hamming net - The Hamming net measures how much the input vector resembles the weight vector of each perceptron.

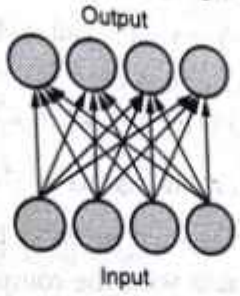


Fig. 4.1.2 Hamming net

2) Maxnet (Mexican Hat Output) - The Maxnet finds the perceptron with the maximum value.

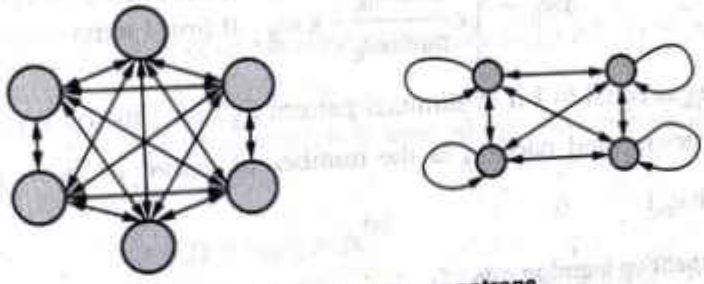


Fig. 4.1.3 Maxnet perceptrons

Here the computations are done as follows,

- Close neighbors : (w > 0)
- Not so close neighbors : (w < 0)
- Distance neighbors : (w = 0)

Properties of competitive learning

- The units in a given layer are broken into several sets of non-overlapping clusters. Each unit within a cluster inhibits every other unit within a cluster. Within each cluster, the unit receiving the largest input achieves its maximum value while all other units in the cluster are pushed to their minimum value. One can arbitrarily set the maximum value to 1 and the minimum value to 0.
- Every unit in every cluster receives inputs from all members of the same set of input units.

- A unit learns if and only if it wins the competition with other units in its cluster.
- A stimulus pattern S_j consists of a binary pattern in which each element of the pattern is either active or inactive. An active element is assigned the value 1 and an inactive element is assigned the value 0.
- Each unit has a fixed amount of weight (all weights are positive) that is distributed among its input lines. The weight on the line connecting to unit i on the upper layer from unit j on the lower layer is designated w_{ij}. The fixed total amount of weight for unit j is designated $\sum_j w_{ij} = 1$. A unit learns by shifting weight from its inactive to its active input lines. If a unit does not respond to a particular pattern, no learning takes place in that unit. If a unit wins the competition, then each of its input lines gives up some portion ϵ of its weight and that weight is then distributed equally among the active input lines. Mathematically, this learning rule can be stated as,

$$\Delta w_{ij} = \begin{cases} 0 & \text{if unit i loses on stimulus k} \\ \epsilon \dfrac{active_{jk}}{nactive_k} - \epsilon w_{ij} & \text{if unit i wins on stimulus k} \end{cases}$$

where $active_{jk}$ is equal to 1 if in stimulus pattern S_k unit j in the lower layer is active and is zero otherwise and $nactive_k$ is the number of active units in pattern S_k (thus $nactive_k = \sum_j active_{jk}$)

Building the competitive learning rule

Following are the important factors of competitive learning rule,

i. Condition to be a winner

Suppose if a neuron y_k wants to be the winner, then there would be the following condition

$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all j, } j \neq k \\ 0 & \text{otherwise} \end{cases}$$

This indicates that any neuron say y_k wants to win, then its induced local field (output_of_summation_units) say v_k should be the largest among all the other neurons in the network.

ii. Condition of sum total of weight

Next constraint over the competitive learning rule is the sum total of weights to a particular output neuron is going to be 1. For example, consider a neuron k then

$$\sum_k w_{kj} = 1 \text{ for all } k$$

iii. Change of weight for the winner.

When a neuron does not respond to the input pattern, then no learning takes place in that neuron. Also a particular neuron wins, then the corresponding weights are adjusted as follows,

$$\Delta w_{kj} = \begin{cases} -\alpha (x_j - w_{kj}), & \text{if neuron k wins} \\ 0 & \text{if neuron k losses} \end{cases}$$

Where α is a learning rule here.

Below is the summary of above constraints,

- With normalized vectors, the activation function of the ith unit can be computed as the inner product of the unit's weight vector w_1 and a particular input pattern $x^{(n)}$

$$g_i(x^{(n)}) = w_i^T x^{(n)}$$

- Note : the inner product of two normal vectors is the cosine of the angle between them
- The neuron with largest activation is then adapted to be more like the input that caused the excitation

$$w_i(t+1) = w_i(t) + \eta x^{(n)}$$

- Following update, the weight vector is renormalized ($\|w\| = 1$)

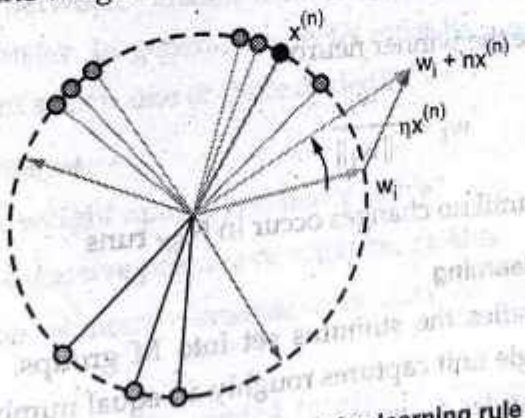


Fig. 4.1.4 Building competitive learning rule

If weights and input patters are un-normalized, the activation function becomes the Euclidean distance

$$g_i(x^{(n)}) = \sqrt{\sum_i (w_i - x_i^{(n)})^2}$$

- The learning rule then become

$$w_i(t+1) = w_i(t) + \eta(x^{(n)} - w_i(t))$$

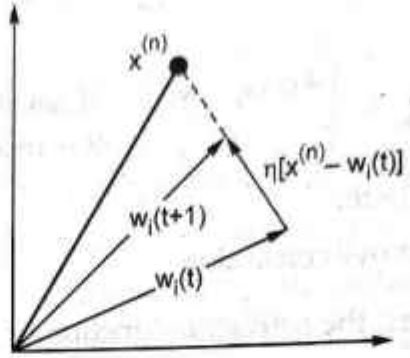


Fig. 4.1.5 Building competitive learning rule

Comptetitive learning algorithm

1. Normalize all input patterns
2. Randomly select a pattern $x^{(n)}$

   2a. Find the winner neuron

   $$i = [w_i^{(n)}]$$

   2b. Update the winner neuron

   $$w_i = w_i + \eta x^{(n)}$$

   2c. Normalize the winner neuron

   $$w_i = \frac{w_i}{\|w_i\|}$$

3. Go to step 2 until no changes occur in N_{EX} runs

Features of competitive learning

- Each cluster classifies the stimulus set into M groups, one for each unit in the cluster. Every single unit captures roughly an equal number of stimulus patterns. It is possible to consider a cluster as forming an M-valued feature in which every stimulus pattern is classified as having exactly one of the M possible values of this feature. Thus, a cluster containing two units acts as a binary feature detector. One element of the cluster responds when a particular feature is present in the stimulus pattern, otherwise the other element responds.

- If there is structure in the stimulus patterns, the units will break up the patterns along structurally relevant lines. Roughly speaking, this means that the system will find clusters if they are there.
- If the stimuli are highly structured, the classifications are highly stable. If the stimuli are less well structured, the classifications are more variable and a given stimulus pattern will be responded to first by one and then by another member of the cluster. These configurations can be more or less stable. For example, if the stimulus points do not actually fall into good clusters, then the configurations will be relatively unstable and the presentation of each stimulus will modify the pattern of responding so that the system will undergo continual evolution. On the other hand, if the stimulus patterns fall in the best way into clusters, then the system will become very stable in the sense that the same units will always respond to the same stimuli.
- The particular grouping done by a particular cluster depends on the starting value of the weights and the sequence of stimulus patterns actually presented. A large number of clusters, each receiving inputs from the same input lines can, in general, classify the inputs into a large number of different groupings or, alternatively, discover a variety of independent features present in the stimulus population. This can provide a kind of distributed representation of the stimulus patterns.
- To a first approximation, the system develops clusters that minimize within - cluster distance, maximize between - cluster distance and balance the number of patterns captured by each cluster. In general, tradeoffs must be made among these various forces and the system selects one of these tradeoffs.

Competitive learning advantages

1. Similar to Instar weight update and hence stable.
2. Won't result in excessive positive or negative weights.
3. Ideal when one wants to analyze raw data of which there is no prior knowledge.
4. Idea of completion can be applied to different types of networks (Hebbian, Kohonen networks, etc.).
5. It can be used for Vector quantization.
6. It can be used for clustering – competitive rule allows a single layer network to group data that lies in a neighborhood of the input space

Competitive learning limitations

1. If a PE weight vector is far away from any of the data clusters, it may never win competition.
2. One is not sure about what it has learnt.
3. If used for clustering, number of clusters have to be decided in advance.
4. Inner product takes account of both direction and magnitude of vector. If vectors are not normalized, may select wrong weight vector after competition.

Applications of competitive networks

1. Vector quantization
2. Analyze raw data
3. Bibliographic classification
4. Image browsing systems
5. Medical diagnosis (visualization of data and
6. Speech recognition
7. Data compression
8. Separating sound sources
9. Environmental modeling
10. Feature extraction
11. Dimensionality reduction
12. Optimization

4.1.2 ART Networks - Working, Pattern Clustering, Feature Mapping, Advantages, Applications

4.1.2.1 Introduction to ART Networks

- The Adaptive Resonance Theory (ART) was incorporated as a hypothesis for human cognitive data handling. The hypothesis has prompted neural models for pattern recognition and unsupervised learning. ART system has been utilized to clarify different types of cognitive and brain data.
- This network was developed by Stephen Grossberg and Gail Carpenter in 1987. It is based on competition and uses unsupervised learning model. Adaptive Resonance Theory ART networks, is always open to new learning adaptive without losing the old patterns *resonance*. Fundamentally, ART network is a vector classifier which accepts an input vector and classifies it into one of the categories depending upon which of the stored pattern it resembles the most.

- The Adaptive Resonance Theory addresses the **stability-plasticity** (stability can be defined as the nature of memorizing the learning and plasticity refers to the fact that they are flexible to gain new information) dilemma of a system that asks how learning can proceed in response to huge input patterns and simultaneously not to lose the stability for irrelevant patterns.
- Other than **stability-plasticity**, the stability-elasticity dilemma is concerned about how a system can adapt new data while keeping what was learned before. For such a task, a feedback mechanism is included among the ART neural network layers. In this neural network, the data in the form of processing elements output reflects back and ahead among layers. If an appropriate pattern is build-up, the resonance is reached, then adaption can occur during this period.
- This formal analysis of how to overcome the learning instability accomplished by a competitive learning model, result into an expended hypothesis, called **Adaptive Resonance Theory** (ART).

4.1.2.2 Operational Principal and Features of ART Models, ART 1

The main operation of ART classification can be divided into the following phases,

- **Recognition phase -** The input vector is compared with the classification presented at every node in the output layer. The output of the neuron becomes "1" if it best matches with the classification applied, otherwise it becomes "0".
- **Comparison phase -** In this phase, a comparison of the input vector to the comparison layer vector is done. The condition for reset is that the degree of similarity would be less than vigilance parameter.
- **Search phase -** In this phase, the network will search for reset as well as the match done in the above phases. Hence, if there would be no reset and the match is quite good, then the classification is over. Otherwise, the process would be repeated and the other stored pattern must be sent to find the correct match.

Working of ART and ART 1

1. ART1 is an unsupervised learning model primarily designed for recognizing binary patterns.
2. It comprises an attentional subsystem, an orienting subsystem, a vigilance parameter and a reset module.

3. The vigilance parameter has a huge effect on the system. High vigilance produces higher detailed memories.
4. The ART1 comprises of two competitive networks, comparison field layer L_1 and the recognition field layer F_2, two control gains, Gain1 and Gain2 and two Short-Term Memory (STM) stages S_1 and S_2.
5. Long Term Memory (LTM) follows somewhere in the range of S_1 and S_2 multiply the signal in these pathways. It gains control and empowers L_1 and L_2 to recognize the current stages of the running cycle.
6. STM reset wave prevents active L_2 cells when mismatches between bottom-up and top-down signals happen at L_1. The comparison layer gets the binary external input passing it to the recognition layer liable for coordinating it to a classification category. This outcome is given back to the comparison layer to find out when the category coordinates the input vector. If there is a match, then a new input vector is read and the cycle begins once again. If there is a mismatch, then the orienting system is in charge of preventing the previous category from getting a new category match in the recognition layer. The given two gains control the activity of the recognition and the comparison layer, respectively. The reset wave specifically and enduringly prevents active L_2 cell until the current is stopped. The offset of the input pattern ends its processing L_1 and triggers the offset of Gain2. Gain2 offset causes consistent decay of STM at L_2 and thereby prepares L_2 to encode the next input pattern without bais.
7. Processing of ART 1
   a. ART1 is a self-organizing neural network having input and output neurons mutually couple using bottom-up and top-down adaptive weights that perform recognition.
   b. The system is first trained as per the adaptive resonance theory by inputting reference pattern data under the type of 5 * 5 matrix into the neurons for clustering within the output neurons.
   c. Further, the maximum number of nodes in L_2 is defined following by the vigilance parameter. The inputted pattern enrolled itself as short term memory activity over a field of nodes L_1.

d. Combining and separating pathways from L_1 to coding field L_2, each weighted by an adaptive long - term memory track, transform into a net signal vector T. Internal competitive dynamics at L_2 further transform T, creating a compressed code or content addressable memory.

e. With strong competition, activation is concentrated at the L_2 node that gets the maximal $L_1 \rightarrow L_2$ signal.

f. Thus the overall processing is divided into four phases comparision, recognition, search and learning.

ART model is unsupervised in nature and consists of,

1. Input unit layer - L_1 layer or the comparison field where in the inputs are processed.

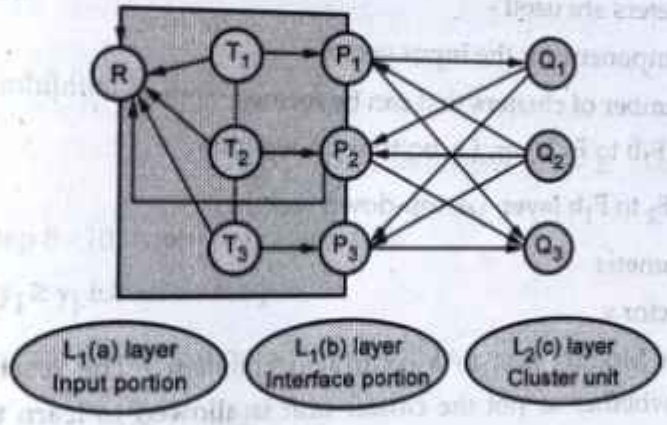


Fig. 4.1.6 ART1 - Layer 1

L_1a layer Input portion – In ART1, there would be no processing in this portion rather than having the input vectors only. It is connected to F_1b layer *interface portion*.

L_1b layer Interface portion – This portion combines the signal from the input portion with that of F_2 layer. F_1b layer is connected to F_2 layer through bottom up weights $\mathbf{b_{ij}}$ and F_2 layer is connected to F_1b layer through top down weights t_{ji}.

2. Layer 2 - F_2 layer or the recognition field which consists of the clustering units. This is a competitive layer. The unit having the largest net input is selected to learn the input pattern. The activation of all other cluster unit are set to 0.

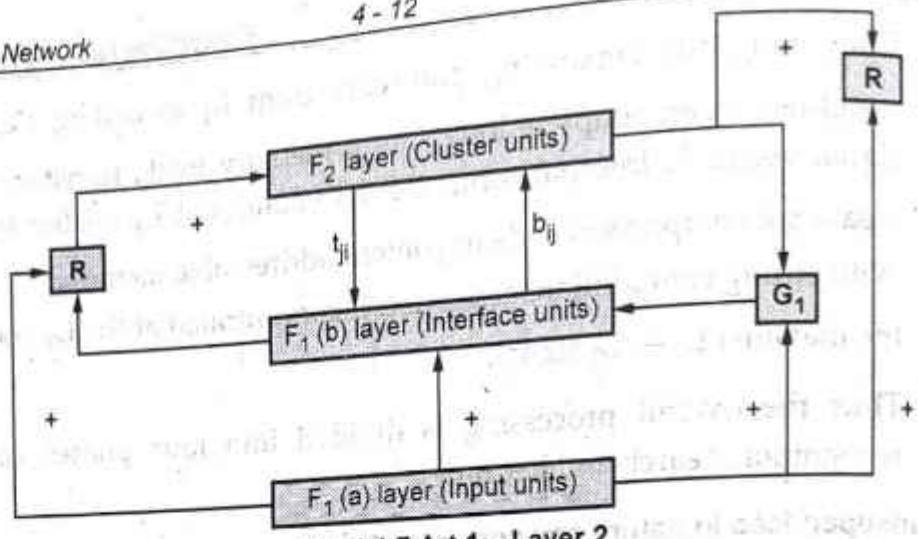


Fig. 4.1.7 Art 1 – Layer 2

Parameters used

Following parameters are used -

n - Number of components in the input vector

m - Maximum number of clusters that can be formed

b_{ij} - Weight from F_1b to F_2 layer, i.e. bottom-up weights

t_{ji} - Weight from F_2 to F_1b layer, i.e. top-down weights

p - Vigilance parameter

$||x||$ - Norm of vector x

3. The Reset Module acts as a control mechanism - The **reset unit** makes the decision whether or not the cluster unit is allowed to learn the input pattern depending on how similar its top-down weight vector is to the input vector and to the decision. This is called the vigilance test.

   The **vigilance parameter** helps to incorporate new memories or new information. Higher vigilance produces more detailed memories, lower vigilance produces more general memories.

   Supplement unit - The major issue with Reset mechanism is that the layer F_2 must have to be inhibited under certain conditions and must also be available when some learning happens. That is why two supplemental units namely, G_1 and G_2 is added along with reset unit, **R**. They are called **gain control units**. These units receive and send signals to the other units present in the network. '+' indicates an excitatory signal, while '–' indicates an inhibitory signal.

ART algorithm

Step 1 - Initialize the learning rate, the vigilance parameter and the weights as follows -

$$\alpha > 1 \text{ and } 0 < \rho \leq 1$$

$$0 < b_{ij}(0) < \frac{\alpha}{\alpha - 1 + n} \text{ and } t_{ij}(a) = 1$$

Step 2 - Continue step 3 - 9, when the stopping condition is not true.

Step 3 - Continue step 4 - 6 for every training input.

Step 4 - Set activations of all F_1a and F_1 units are follows

$$F_2 = 0 \quad \text{and} \quad F_1\alpha = \text{Input vectors}$$

Step 5 - Input signal from F_1 a to F_1b layer must be sent like

$$s_i = x_i$$

Step 6 - For every inhibited F_2 node

$$y_j = \sum_i b_{ij}x_i \quad \text{The condition is } y_j \neq -1$$

Step 7 - Perform step 8 - 10, when the reset is true

Step 8 - Find J for $y_J \geq y_j$ for all nodes j

Step 9 - Again calculate the activation on F_1b as follows

$$x_i = \text{sit } J_i$$

Step 10 - Now, after calculating the norm of vector x and vector s, we need to check the reset condition as follows -

If $\|x\|/\|s\|$ < vigilance parameter ρ, then in hibit node J and go to step 7

Else If $\|x\|/\|s\| \geq$ vigilance parameter p, then proceed further.

Step 11 - Weight updating for node J can be done as follows -

$$b_{ij}(\text{new}) = \frac{ax_i}{\alpha - 1 + \|x\|}$$

$$t_{ij}(\text{new}) = x_i$$

Step 12 - The stopping condition for algorithm must be checked and it may be as follows -

- Do not have any change in weight.
- Reset is not performed for units.
- Maximum number of epochs reached.

- Generally **two types of learning** exists, slow learning and fast learning. In fast learning, weight update during resonance occurs rapidly. It is used in ART1. In slow learning, the weight change occurs slowly relative to the duration of the learning trial. It is used in ART2.

Types of Adaptive Resonance Theory (ART)

The ARTs can be classified as follows :

- **ART1** - It is the simplest and the basic ART architecture. It is capable of clustering binary input values.
- **ART2** - It is extension of ART1 that is capable of clustering continuous - valued input data.
- **Fuzzy ART** - It is the augmentation of fuzzy logic and ART.
- **ARTMAP** - It is a supervised form of ART learning where one ART learns based on the previous ART module. It is also known as predictive ART.
- **FARTMAP** - This is a supervised ART architecture with Fuzzy logic included.

Advantages of Adaptive Learning Theory (ART)

1. It can be utilized with different techniques to give more precise outcomes.
2. It is stable and is not disturbed by a wide variety of inputs provided to its network.
3. It doesn't ensure any stability in forming clusters.
4. It can be used in variety fields such as face recognition, embedded system and robotics, target recognition, medical diagnosis, signature verification, etc.
5. It has benefit over competitive learning that the competitive learning can not include new clusters when considered necessary but ART can.

Advantages and Limitations of Adaptive Resonance Theory

All the ART networks are not consistent (like the Fuzzy ART and ART1) as they depend upon the order in which training data, or upon the learning rate.

4.1.2.3 ART Applications Character Recognition using ART Network

There are varied ART applications including character and target recognition, face recognition, medical diagnosis, signature verification, mobile control robot.

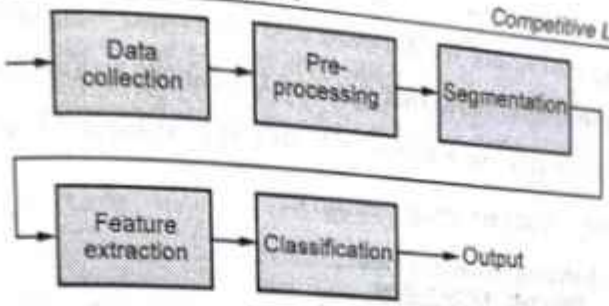


Fig. 4.1.8 Character recognition process

- A typical character recognition system consists of stages given in above figure. The recognition process starts with data collection stage. Data is collected in the form of images of different machine printed alphabets.
- Next the data is converted into binary form. The images are noise free so no need to do extra preprocessing for removing noise.
- Segmentation is the most important part of character recognition systems. This step involves localization of the limits of each character and to isolate them properly. In this the identification of the boundaries of the character and separating them for further processing.
- After segmentation a set of features are required for each character. In feature extraction stage each character is represented as a feature vector, which becomes its identity. This vector is used to distinguish the character from other characters. The features extracted from the images will be the inputs given Back Propagation algorithm for classification.
- There are two main methods for extracting features in character recognition
  1. In the first method, the algorithm for feature detection defines a character by evaluating its lines and strokes.
  2. In the second method, pattern recognition works by identifying the entire character.
- In the ART algorithm, a backward network is adopted for vigilance test in addition to the conventional forward networks between the input neurons and output neurons, two widely used versions of ART algorithms are : ART-1 for binary-valued patterns and ART-2 for continuous - valued patterns.
- It can adaptively create a new neuron for an incoming input pattern if it is determined (by a vigilance test) to be sufficiently different from the existing clusters. Such a vigilance test is incorporated into the adaptive backward network.

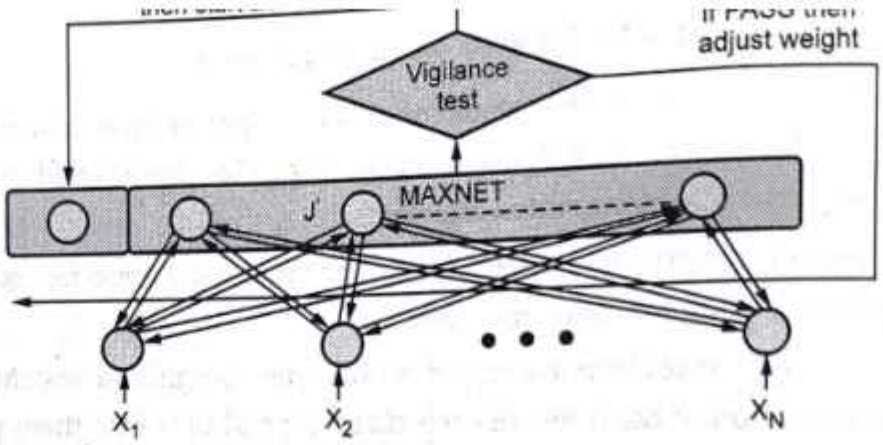


Fig. 4.1.9 ART structure

4.2 Self-Organization Maps (SOM) - Two Basic Feature Mapping Models, Self-Organization Map, SOM Algorithm, Properties of Feature Map, Computer Simulations, Learning Vector Quantization, Adaptive Pattern Classification

4.2.1 Basics of SOM

- A **Self-Organizing Map (SOM)** is a type of Artificial Neural Network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two - dimensional), discretized representation of the input space of the training samples, called a **map** and is therefore a method to do dimensionality reduction.
- **SOM** was introduced by Finnish professor Teuvo Kohonen in the 1980s is sometimes called a **Kohonen map**.
- Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error - correction learning (such as backpropagation with gradient descent) and in the sense that they use a neighborhood function to preserve the topological properties of the input space.
- A Self-organizing Map is a data visualization technique. SOMs map multidimensional data onto lower dimensional subspaces where geometric relationships between points indicate their similarity. The reduction in

dimensionality that SOMs provide allows people to visualize and interpret what would otherwise be, for all intents and purposes, indecipherable data.

- SOMs generate subspaces with an unsupervised learning neural network trained with a competitive learning algorithm. Neuron weights are adjusted based on their proximity to "winning" neurons (i.e. neurons that most closely resemble a sample input). Training over several iterations of input data sets results in similar neurons grouping together and vice versa.

4.2.2 Working of SOM

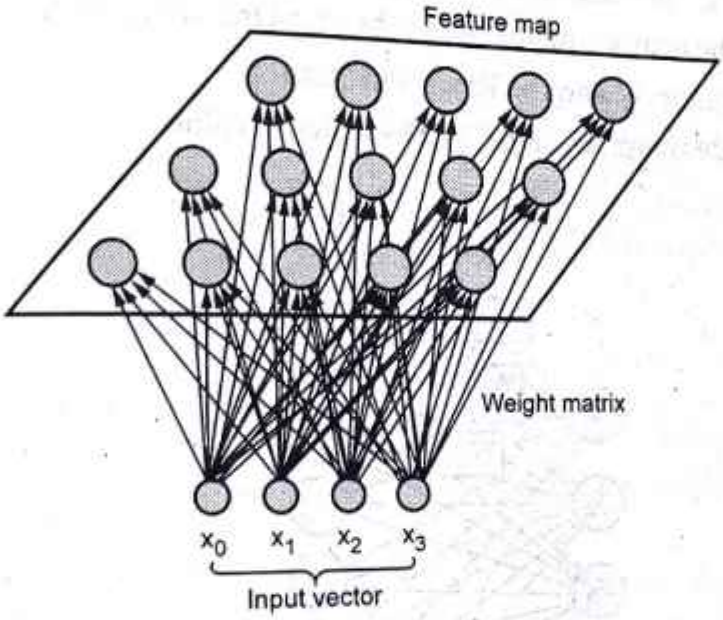


Fig. 4.2.1 Self organizing map

1. Self - organizing maps consist of two basic layers, the first one is the input layer, and the second one is the output layer, which is also known as a feature map. Each data point in the dataset recognizes itself by competing for a representation.
2. The Self-Organizing Maps' mapping steps start from initializing the weight to vectors. After this, a random vector as the sample is selected and the mapped vectors are searched to find which weight best represents the chosen sample.
3. Each weighted vector has neighboring weights present that are close to it. The chosen weight is then rewarded by being able to become a random sample vector. This helps the map to grow and form different shapes.
4. Most generally, they form square or hexagonal shapes in a 2D feature space. This whole process is repeatedly performed a large number of times and more than 1000 times.

5. Self-Organizing - an unsupervised ANN uses competitive learning to update its weights that is competition, cooperation and adaptation. Each neuron of the output layer is present with a vector with dimension n. The distance between each neuron present at the output layer and the input data is computed. The neuron with the lowest distance is termed as the most suitable fit. Updating the vector of the suitable neuron in the final process is known as adaptation, along with its neighbour in cooperation. After selecting the suitable neuron and its

vector, by running through all weight vectors. The weight with the shortest distance is the winner.

To determine the best matching unit, one method is to iterate through all the nodes and calculate the Euclidean distance between each node's weight vector and the current input vector. The node with a weight vector closest to the input vector is tagged as the BMU.

The **Euclidean distance is given as :**

$$\text{Distance} = \sqrt{\sum_{i=0}^{i=n} (X_i - W_i)^2}$$

iv. The suitable weight is further rewarded with transitioning into more like the sample vector. The neighbours transition like the sample vector chosen. The closer a node is to the Best Matching Unit, the more its weights get altered and the farther away the neighbour is from the node, the less it learns.

v. Repeat the second step for N iterations.

4.2.3 Feature Map

- **Feature Map** is also called as Activation map. Once the filters are extracted from the Image. And these filters are small sections of the image which will be having different features.
- Number of filters used on the input should be created the same number amount of feature maps. So an input image with 6 filters will have 6 feature maps. Central goal of SOM is to generate and learn the feature map.
- The aim is to learn a **feature map** from the spatially **continuous input space**, in which our input vectors live, to the low dimensional spatially **discrete output space**, which is formed by arranging the computational neurons into a grid.
- The stages of the SOM algorithm that achieves this can be summarised as follows :

1. **Initialization** - Choose random values for the initial weight vectors w_j.
2. **Sampling** - Draw a sample training input vector x from the input space.
3. **Matching** - Find the winning neuron I(x) that has weight vector closest to the input vector, i.e. the minimum value of $d_j(x) = \sum_{i=1}^{D} (x_i - w_{ji})^2$.
4. **Updating** - Apply the weight update equation $\Delta w_{ji} = \eta(t)\ T_{j,I(x)}(t)\ (x_i - w_{ji})$ where $T_{j,I(x)}(t)$ is a Gaussian neighbourhood and $\eta(t)$ is the learning rate.
5. **Cotinuation** - Keep returning to step 2 until the feature map stops changing.

Properties of the feature map

- Once the SOM algorithm has converged, the feature map displays important statistical characteristics of the input space. Given an input vector x, the feature map Φ provides a winning neuron I(x) in the output space and the weight vector $w_{I(x)}$ provides the coordinates of the image of that neuron in the input space.

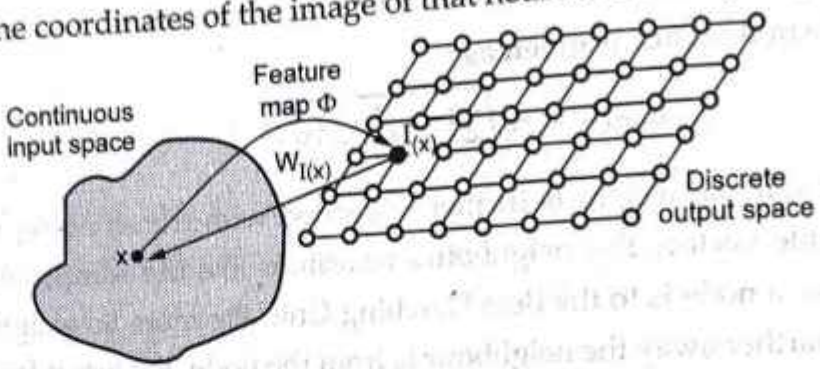


Fig. 4.2.3

Properties of feature map

1. Approximation of the input space - The feature map is represented by the set of weight vectors $\{w_i\}$ in the output space, provides a good approximation to the input space. The aim of the SOM as storing a large set of input vectors {x} by finding a smaller set of prototypes $\{w_i\}$ so as to provide a good approximation to the original input space. The theoretical basis of this idea is rooted in vector quantization theory, the motivation of which is dimensionality reduction or data compression.

   In effect, the goodness of the approximation is given by the total square distance,

$$D = \sum_x \|x - w_{I(x)}\|^2$$

   which to minimize. Gradient descent method is followed in SOM in weight update algorithm, which confirms that it is generating a good approximation of the input space.

2. Topological ordering - The feature map computed by the SOM algorithm is topologically ordered that is the spatial location of a neuron in the output lattice/grid corresponds to a particular domain or feature of the input pattern. The topological ordering property is a direct consequence of the weight update equation that forces the weight vector $w_{I(x)}$ of the winning neuron I(x) to move toward the input vector x. The crucial factor is that the weight updates are

move the weight vectors w_j of the closest neighbouring neurons j along with the winning neuron I(x). Together these weight changes cause the whole output space to become appropriately ordered.

The feature map can be visualized as an elastic or virtual net with a grid like topology. Each output node can be represented in the input space at coordinates given by their weights. Then if the neighbouring nodes in output space have their corresponding points in input space connected together, the resulting image of the output grid reveals directly the topological ordering at each stage of the network training.

3. Density matching - The feature map reflects variations in the statistics of the input distribution - regions in the input space from which the sample training vectors x are drawn with high probability of occurrence are mapped onto larger domains of the output space and therefore with better resolution than regions of input space from which training vectors are drawn with low probability.

There is a need to relate the input vector probability distribution p(x) to the magnification factor m(x) of the feature map. Generally, for two dimensional feature maps the relation cannot be expressed as a simple function, but in one dimension one can sh

4.2.4 Advantages and Disadvantages of Self - organizing Maps

Advantages

- Data can be easily interpreted and understood with the help of techniques like reduction of dimensionality and grid clustering.
- Self-organizing maps are capable of handling several types of classification problems while providing a useful and intelligent summary from the data at the same time.

Disadvantages

- It does not create a generative model for the data and therefore the model does not understand how data is being created.
- Self-organizing maps are not good choice to perform well while working with categorical data and even worse for mixed types of data.
- The model preparation time is comparatively very slow and hard to train against the slowly evolving data.

4.2.5 Learning Vector Quantization

- The Learning Vector Quantization Algorithm (or LVQ for short) is an artificial neural network algorithm that allows to choose the number of training instances to suspend and know exactly what these examples should look like.
- The representation of LVQ is a collection of codebook vectors. These are randomly selected at the beginning and are suitable for optimally summarizing the training data set in multiple iterations of the learning algorithm. After learning, one can use the codebook vector to K-nearest neighbors (A similar forecast).
- The most similar neighbor (best matching codebook vector) is found by calculating the distance between each codebook vector and the new data instance. Then return the class value of the best matching unit or (actual value in the case of regression) as a prediction. The best results are obtained if the data is rescaled to the same range (for example between 0 and 1).
- A Kohonen SOM is a clustering technique, which can be used to provide insight into the nature of data. One can transform this unsupervised neural network into a supervised LVQ neural network.
- The network architecture is just like a SOM, but without a topological structure.
- Each output neuron represents a known category (e.g. apple, pear, orange).

Input vecto $\underline{x} = (x_1, x_2, x_3, \ldots, x_n)$

Weight vector for the jth output neuron $\underline{w}_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{nj})$

C_j = Category represented by the jth neuron.

This is pre-assigned.

T = Correct category for input $\underline{x}$

Define **Euclidean** distance between the input vector and the weight vector of the jth neuron as :

$$D(j) = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$$

- Training algorithm

STEP	
0	Initialize weight vectors to the first m training vectors, where m is the number of different categories and set $\alpha(0)$. This weight initialization techniques is presented here is only one of many different methods.
1	While stopping condition false, do steps 2 to 6.
2	For each training input vector $\underline{x}$, do steps 3 to 4.
3	Find J so that D(J) is a minimum

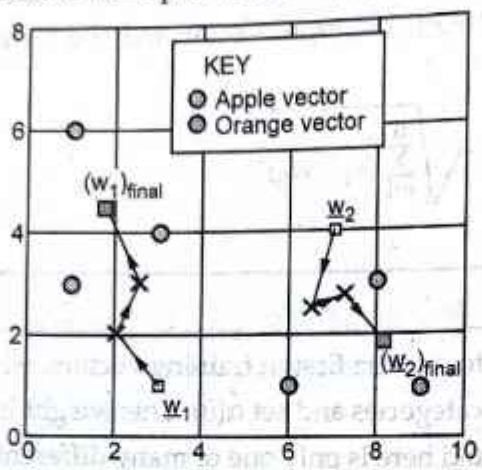
4	Update the weights of the J neuron as follows :
	IF $T = C_J$ THEN
	$\underline{w}_J(new) = \underline{w}_J(old) + \alpha(\underline{x} - \underline{w}_J(old))$
	(i.e. move the weight vector $\underline{w}$ toward the input vector $\underline{x}$)
	IF $T \neq C_J$ THEN
	$\underline{w}_J(new) = \underline{w}_J(old) - \alpha(\underline{x} - \underline{w}_J(old))$
	(i.e. move $\underline{w}$ away from $\underline{x}$)

5	Reduce learning rate is α
6	Test stopping condition : This may be a fixed number of iterations or the learning rate reaching a sufficiently small value.

- Example - Measure the weight and height of three apples and three oranges.
- The input vector $\underline{x} = (x_1, x_2, x_3, \ldots, x_n)$ in this case would be $\underline{x}$ = (height, weight).

These input vectors are shown in the graph below.

- Using only two neurons in the output layer, let the initial weight vectors of the first and second neuron be $\underline{w}_1 = (3,1)$ and $w_2 = (7,4)$ respectively.
- Using a learning rate of 0.5, the graph below shows how the weight vectors of the two neurons change as the LVQ network is presented with input vectors. The order of presentation of the input vector is (1,3), (3,4), (6,1), ..., (1,6).



$$x = \begin{bmatrix} x_1 & x_2 & T \\ 1 & 3 & 1 \\ 3 & 4 & 1 \\ 6 & 1 & -1 \\ 8 & 3 & -1 \\ 9 & 1 & -1 \\ 1 & 6 & 1 \end{bmatrix}$$

$\alpha = 0.5$

Fig. 4.2.4

Review Questions with Answers

1. *Write a note on competitive learning.* **(Refer section 4.1.1)**
2. *Explain components of competitive learning.* **(Refer section 4.1.2)**
3. *Explain adaptive resonance learning.* **(Refer section 4.1.2)**
4. *List various applications of ART.* **(Refer section 4.1.2)**
5. *What are features of ART models.* **(Refer section 4.1.2)**
6. *Brief about SOM.* **(Refer section 4.2)**
7. *How SOM works ?* **(Refer section 4.2)**
8. *What is feature map ?* **(Refer section 4.2)**
9. *List advantages and disadvantages of SOM.* **(Refer section 4.2)**
10. *Explain learning vector quantization.* **(Refer section 4.2)**

Unit V

5 Convolution Neural Network

Syllabus

Building blocks of CNNs, Architectures, convolution / pooling layers, Padding, Strided convolutions, Convolutions over volumes, SoftMax regression, Deep Learning frameworks, Training and testing on different distributions, Bias and Variance with mismatched data distributions, Transfer learning, multi-task learning, end-to-end deep learning, Introduction to CNN models: LeNet - 5, AlexNet, VGG - 16, Residual Networks.

Contents

5.1 Introduction to Convolution Neural Network

5.2 The Basic Structure of CNN

5.3 Convolution Operation

5.4 Pooling

5.5 Fully Connected Layers

5.6 Convolutions Over Volume

5.7 SoftMax Regression

5.8 Deep Learning Frameworks

5.9 Bias and Variance with Mismatched Data Distributions

5.10 Learning Representations from Data

5.11 Multi-Task Learning

5.12 End-to-End Deep Learning

5.13 Introduction to CNN Models

5.1 Introduction to Convolution Neural Network

- Convolutional Neural Network (CNN) is a deep learning neural network designed for processing structured arrays of data such as images. A CNN is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer.
- Convolutional neural network is also called ConvNet.
- In CNN, 'convolution' is referred to as the mathematical function. It's a type of linear operation in which you can multiply two functions to create a third function that expresses how one function's shape can be changed by the other.
- In simple terms, two images that are represented in the form of two matrices, are multiplied to provide an output that is used to extract information from the image.
- CNN represents the input data in the form of multidimensional arrays. It works well for a large number of labeled data. CNN extract the each and every portion of input image, which is known as receptive field. It assigns weights for each neuron based on the significant role of the receptive field.
- Instead of preprocessing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute.
- The goal of CNN is to reduce the images so that it would be easier to process without losing features that are valuable for accurate prediction.
- A convolutional neural network is made up of numerous layers, such as convolution layers, pooling layers, and fully connected layers, and it uses a back-propagation algorithm to learn spatial hierarchies of data automatically and adaptively.
- To understand the Concept of Convolutional Neural Networks (CNNs), let us take an example of the images our brain can interpret.
- As soon as we see an image, our brain starts categorizing it based on the color, shape and sometimes also the message that image is conveying. Similar thing can be done through machines even after a rigorous training. But the difficulty is there is a huge difference in what humans interpret and what machine does. For a machine, the image is merely an array of pixels. There is a unique pattern included the image is merely an array of pixels. There is a unique pattern included in each object present in the image and the computer tries to find out these patterns to get the information about the image.

- Machines can be trained giving tons of images to increase its ability to recognize the objects included in a given input image.
- Most of the digital companies have opted for CNNs for image recognition, some of these include Google, Amazon, Instagram, Interest, Facebook, etc.
- Hence, we define a convolutional neural network as : "A neural network consisting of multiple convolutional layers which are used mainly for image processing, classification, segmentation and other correlated data".

5.1.1 Advantages and Disadvantages of CNN

1. Advantages :

- CNN automatically detects the important features without any human supervision.
- CNN is also computationally efficient.
- Higher accuracy.
- Weight sharing is another major advantage of CNNs.
- Convolutional neural networks also minimize computation in comparison with a regular neural network.
- CNNs make use of the same knowledge across all image locations.

2. Disadvantages :

- Adversarial attacks are cases of feeding the network 'bad' examples to cause misclassification.
- CNN requires lot of training data.
- CNNs tend to be much slower because of operations like maxpool.

5.1.2 Application of CNN

- CNN is mostly used for image classification, for example to determine the satellite images containing mountains and valleys or recognition of handwriting, etc. image segmentation, signal processing, etc. are the areas where CNN are used.
- **Object detection :** Self-driving cars, AI-powered surveillance systems and smart homes often use CNN to be able to identify and mark objects. CNN can identify objects on the photos and in real-time, classify, and label them.
- **Voice synthesis :** Google Assistant's voice synthesizer uses Deepmind's WaveNet ConvNet model.
- **Astrophysics :** They are used to make sense of radio telescope data and predict the probable visual image to represent that data.

5.2 The Basic Structure of CNN

- Fig. 5.2.1 shows basic architecture of CNN.

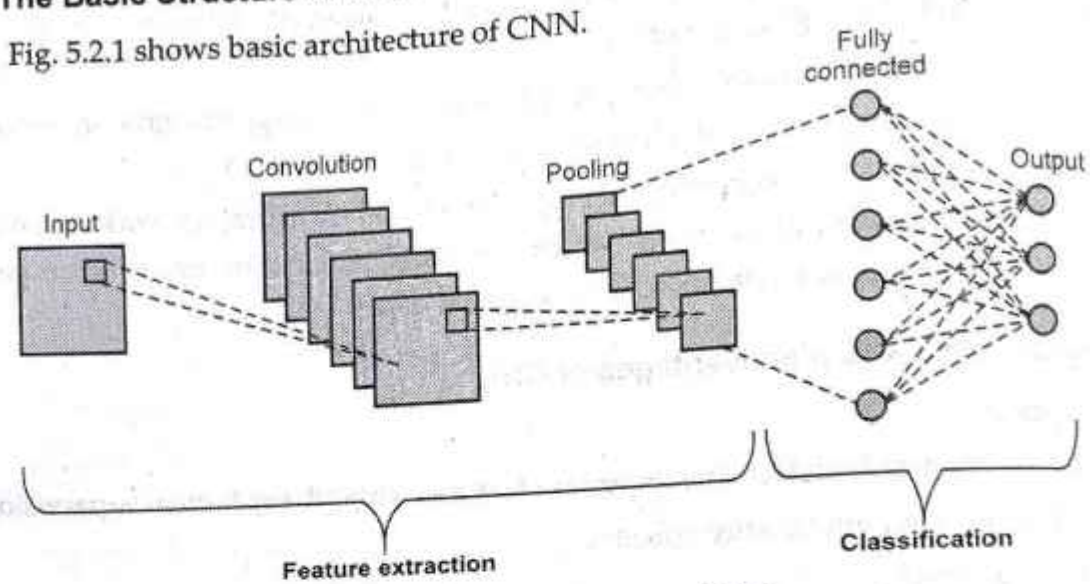


Fig. 5.2.1 Basic architecture of CNN

- A convolutional neural network, as discussed above, has the following layers that are useful for various deep learning algorithms. Let us see the working of these layers taking an example of the image having dimension of $12 \times 12 \times 4$. These are :
  1. **Input layer :** This layer will accept the image of width 12, height 12 and depth 4.
  2. **Convolution layer :** It computes the volume of the image by getting the dot product between the image filters possible and the image patch. For example, there are 10 filters possible, then the volume will be computed as $12 \times 12 \times 10$.
  3. **Activation function layer :** This layer applies activation function to each element in the output of the convolutional layer. Some of the well accepted activation functions are ReLu, Sigmoid, Tanh, Leaky ReLu, etc. These functions will not change the volume obtained at the convolutional layer and hence it will remain equal to $12 \times 12 \times 10$.
  4. **Pool layer :** This function mainly reduces the volume of the intermediate outputs, which enables fast computation of the network model, thus preventing it from overfitting.

5.3 Convolution Operation

- Convolution operation focuses on extracting/preserving important features from the input. Convolution operation allows the network to detect horizontal and vertical edges of an image and then based on those edges build high-level features in the following layers of neural network.

- In general form, convolution is an operation on two functions of a real valued argument. To motivate the definition of convolution, we start with examples of two functions we might use.
- Suppose we are tracking the location of a spaceship with a laser sensor. Laser sensor provides a single output x(t), the position of the spaceship at time t. Both "x" and "t" are real-valued, i.e., we can get a different reading from the laser sensor at any instant in time.
- Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements.
- We can do this with a weighting function w(a), where "a" is the age of a measurement. If we apply such a weighted average operation at every moment, we obtain a new function providing a smoothed estimate of the position "s" of the spaceship.
- Convolution operation uses three parameters : Input image, Feature detector and Feature map.
- Convolution operation involves an input matrix and a filter, also known as the kernel. Input matrix can be pixel values of a grayscale image whereas a filter is a relatively small matrix that detects edges by darkening areas of input image where there are transitions from brighter to darker areas. There can be different types of filters depending upon what type of features we want to detect, e.g. vertical, horizontal, or diagonal, etc.
- Input image is converted into binary 1 and 0. The convolution operation, shown in Fig. 5.3.1 is known as the feature detector of a CNN. The input to a convolution can be raw data or a feature map output from another convolution. It is often interpreted as a filter in which the kernel filters input data for certain kinds of information.

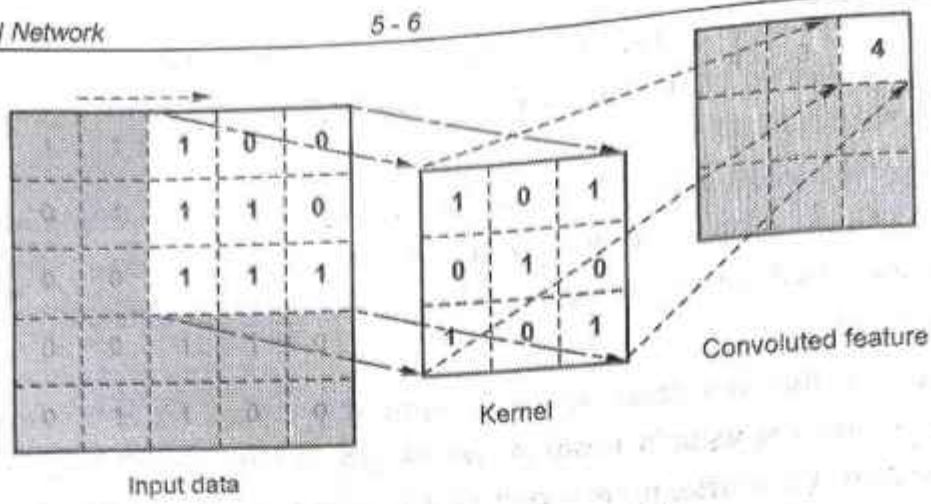


Fig. 5.3.1 Convolution operation

- Sometimes a 5×5 or a 7×7 matrix is used as a feature detector. The feature detector is often referred to as a "kernel" or a "filter,". At each step, the kernel is multiplied by the input data values within its bounds, creating a single entry in the output feature map.
- Generally, an image can be considered as a matrix whose elements are numbers between 0 and 255. The size of image matrix is : *image height*image width*number of image channels*
- A grayscale image has 1 channel, where a colour image has 3 channels.
- Kernel : A kernel is a small matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers produce different results under convolution. The size of a kernel is arbitrary but 3×3 is often used. Fig. 5.3.2 shows example of kernel.

0	1	0
1	1	1
0	1	0

Fig. 5.3.2 Example of kernel

- Convolutional layers perform transformations on the input data volume that are a function of the activations in the input volume and the parameters.
- In reality, convolutional neural networks develop multiple feature detectors and use them to develop several feature maps which are referred to as convolutional layers and it is shown in Fig. 5.3.3.

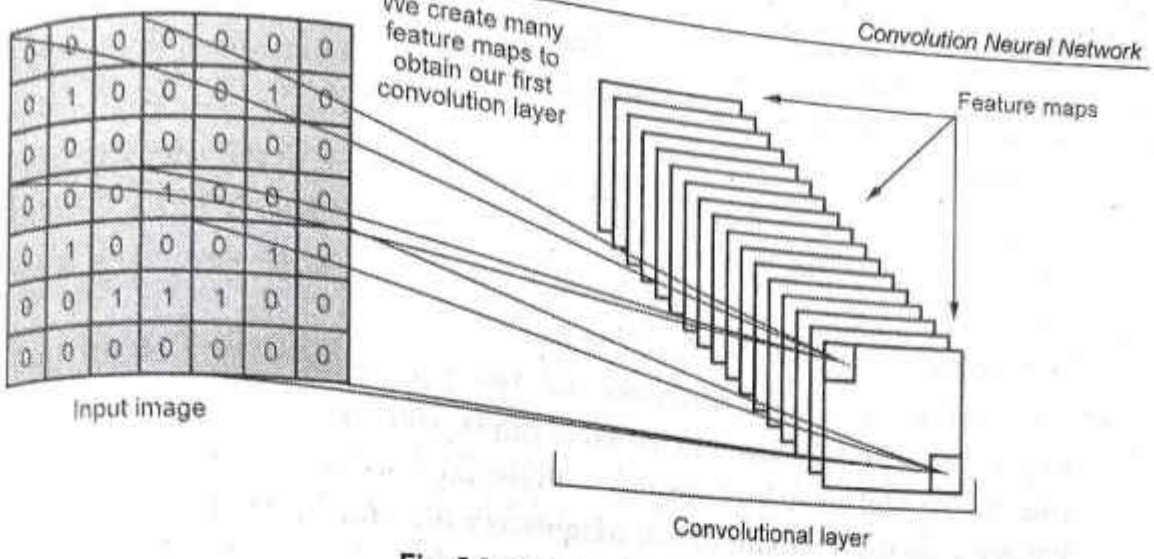


Fig. 5.3.3 Feature detectors

- Through training, the network determines what features it finds important in order for it to be able to scan images and categorize them more accurately.
- Convolutional layers have parameters for the layer and additional hyper-parameters. Gradient descent is used to train the parameters in this layer such that the class scores are consistent with the labels in the training set.
- Components of convolutional layers are as follows :
  a) Filters
  b) Activation maps
  c) Parameter sharing
  d) Layer-specific hyper-parameters
- Filters are a function that has a width and height smaller than the width and height of the input volume. The filters are tensors, and they are used to convolve the input tensor when the tensor is passed to the layer instance. The random values inside the filter tensors are the weights of the convolutional layer.
- Sliding each filter across the spatial dimensions (width, height) of the input volume during the forward pass of information through the CNN. This produces a two dimensional output called an activation map for that specific filter.

5.3.1 Parameter Sharing

- Parameter sharing is used in CNN to control the total parameter count. Convolutional layers reduce the parameter count further by using a technique called parameter sharing.

- The user can reduce the number of parameters by making an assumption that if one feature can compute at some spatial position (x_1), then it is useful to compute a different place (x_2, y_2).
- In other words, denoting a single 2D slice of depth as a depth slice. For example, during back-propagation, every neuron in the network will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.
- If all neurons in a single depth slice are using the same weight vector, then the forward pass of the convolutional layer can be computed in each depth slice as a convolution of the neuron's weights with the input volume. This is the reason why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.
- Fig. 5.3.4 shows convolution shares the same parameters across all spatial locations.

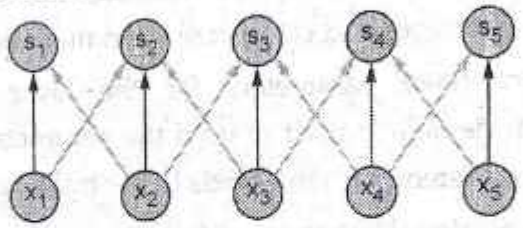


Fig. 5.3.4 Convolution shares the same parameters across all spatial locations

5.3.2 Equivariant Representation

- Convolution function is equivariant to translation. This means that shifting the input and applying convolution is equivalent to applying convolution to the input and shifting it.
- If we move the object in the input, its representation will move the same amount in the output.
- **General definition :** If representation(transform(x)) = Transform(representation(x)) then representation is equivariant to the transform
- Convolution is equivariant to translation. This is a direct consequence of parameter sharing.
- It is useful when detecting structures that are common in the input. For example, edges in an image.
- Equivariance in early layers is good. We are able to achieve translation-invariance (via max-pooling) due to this property.
- Convolution is not equivariant to other operations such as change in scale or rotation.

- **Example of equivariance :** With 2D images convolution creates a map where certain features appear in the input. If we move the object in the input, the representation will move the same amount in the output. It is useful to detect edges in first layer of convolutional network. Same edges appear everywhere in image, so it is practical to share parameters across entire image.

5.3.3 Padding

- **Padding** is the process of adding one or more pixels of zeros all around the boundaries of an image, in order to increase its effective size. Zero padding helps to make output dimensions and kernel size independent.
- One observation is that the convolution operation reduces the size of the $(q + 1)^{th}$ layer in comparison with the size of the qth layer. This type of reduction in size is not desirable in general, because it tends to lose some information along the borders of the image. This problem can be resolved by using padding.
- 3 common zero padding strategies are :

  a) **Valid convolution :** Extreme case in which no zero-padding is used whatsoever, and the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the input. For a kernel of size k in any dimension, the input shape of m in the direction will become $m - k + 1$ in the output. This shrinkage restricts architecture depth.

  b) **Same convolution :** Just enough zero-padding is added to keep the size of the output equal to the size of the input. Essentially, for a dimension where kernel size is k, the input is padded by $k - 1$ zeros in that dimension.

  c) **Full convolution :** Another extreme case where enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m + k - 1$.

  d) The 1D block is composed by a configurable number of filters, where the filter has a set size; a convolution operation is performed between the vector and the filter, producing as output a new vector with as many channels as the number of filters. Every value in the tensor is then fed through an activation function to introduce nonlinearity.
- When padding is not used, the resulting "padding" is also referred to as a valid padding. Valid padding generally does not work well from an experimental point of view. In the case of valid padding, the contributions of the pixels on the borders of the layer will be under-represented compared to the central pixels in the next hidden layer, which is undesirable.

5.3.4 Stride

- Convolution functions used in practice differ slightly compared to convolution operations as it is usually understood in the mathematical literature.
- In general a convolution layer consists of application of several different kernels to the input. This allows the extraction of several different features at all locations in the input. This means that in each layer, a single kernel is not applied. Multiple kernels are used as different feature detectors.
- The input is generally not real-valued but instead vector valued. Multi-channel convolutions are commutative only if the number of output and input channels is the same.
- In order to allow for calculation of features at a coarser level, strided convolutions can be used. The effect of strided convolution is the same as that of a convolution followed by a down sampling stage. This can be used to reduce the representation size.
- The **stride** indicates the pace by which the filter moves horizontally and vertically over the pixels of the input image during convolution. Fig. 5.3.5 shows stride during convolution.

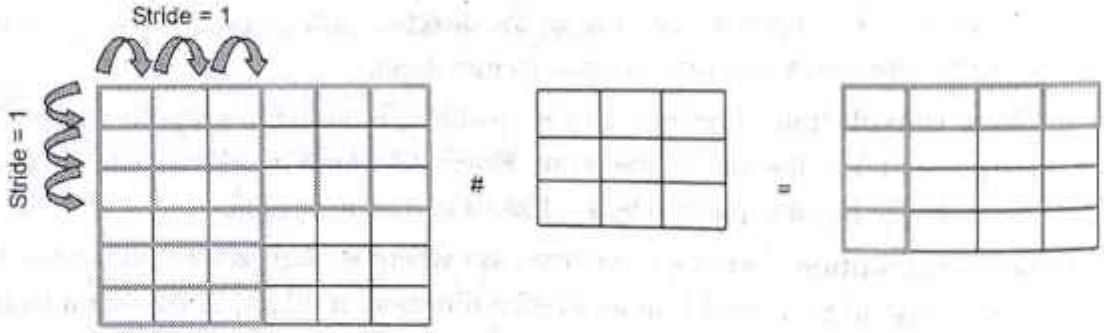


Fig. 5.3.5 Stride during convolution

- Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. Stride is a component for the compression of images and video data. For example, if a neural network's stride is set to 1, the filter will move one pixel or unit, at a time. If stride = 1, the filter will move one pixel.
- Stride depends on what we expect in our output image. We prefer a smaller stride size if we expect several fine-grained features to reflect in our output. On the other hand, if we are only interested in the macro-level of features, we choose a larger stride size.

5.3.5 Typical Setting

- For square images, we use stride sizes of 1 in most settings. Even when strides are used, small strides of size 2 are used. In cases where the input images are not square, preprocessing is used to enforce this property.
- For example, one can extract square patches of the image to create the training data. The number of filters in each layer is often a power of 2, because this often results in more efficient processing. Such an approach also leads to hidden layer depths that are powers of 2.

5.3.6 ReLU Layer

- In this layer we remove every negative value from the filtered image and replace it with zero. This function only activates when the node input is above a certain quantity. So, when the input is below zero the output is zero.
- However, when the input rises above a certain threshold it has a linear relationship with the dependent variable. This means that it is able to accelerate the speed of a training data set in a deep neural network that is faster than other activation functions.
- In traditional neural networks, the activation function is combined with a linear transformation with a matrix of weights to create the next layer of activations.
- The reason why the rectifier function is typically used as the activation function in a convolutional neural network is to increase the nonlinearity of the data set. By removing negative values from the neurons' input signals, the rectifier function is effectively removing black pixels from the image and replacing them with gray pixels.

5.4 Pooling

- Pooling helps the representation become slightly invariant to small translations of the input. A pooling function takes the output of the previous layer at a certain location L and computes a "summary" of the neighborhood around L.
- The pooling layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input.
- The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network and hence to also control overfitting. No learning takes place on the pooling layers.

- The addition of a pooling layer after the convolutional layer is a common pattern used for ordering layers within a convolutional neural network that may be repeated one or more times in a given model.
- The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps. Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps.
- The size of the pooling operation or filter is smaller than the size of the feature map. This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g. each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size.
- For example, a pooling layer applied to a feature map of 6×6 (36 pixels) will result in an output pooled feature map of 3×3 (9 pixels). The pooling operation is specified, rather than learned.
- The pooling operation, also called subsampling, is used to reduce the dimensionality of feature maps from the convolution operation. Max pooling and average pooling are the most common pooling operations used in the CNN.
- Pooling units are obtained using functions like max-pooling, average pooling and even L2-norm pooling. At the pooling layer, forward propagation results in an $N \times N$ pooling block being reduced to a single value - value of the "winning unit". Back-propagation of the pooling layer then computes the error which is acquired by this single value "winning unit".
- Pooling layers, also known as down sampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling :
- Max pooling : As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- Average pooling : As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.
- Invariance to local translation can be useful if we care more about whether a certain feature is present rather than exactly where it is.

5.5 Fully Connected Layers

- Fully connected layers have the normal parameters for the layer and hyperparameters. This layer performs transformations on the input data volume that are a function of the activations in the input volume and the parameters.
- Neural networks are a set of dependent nonlinear functions. Each individual function consists of a neuron (or a perceptron).
- In fully connected layers, the neuron applies a linear transformation to the input vector through a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function f.

$$y_{jk}(x) = f\left(\sum_{i=1}^{n_H} w_{jk}x_i + w_{j0}\right)$$

- Here, we are taking the dot product between the weights matrix W and the input vector x. The bias term (W0) can be added inside the nonlinear function. I will ignore it for the rest of the article as it doesn't affect the output sizes or decision-making and is just another weight.
- The activation function "f" wraps the dot product between the input of the layer and the weights matrix of that layer.

5.6 Convolutions Over Volume

- Suppose, instead of a 2-D image, we have a 3-D input image of shape $6 \times 6 \times 3$. How will we apply convolution on this image ? We will use a $3 \times 3 \times 3$ filter instead of a 3×3 filter. Let's look at an example:
- Input : $6 \times 6 \times 3$
- Filter : $3 \times 3 \times 3$
- The dimensions above represent the height, width and channels in the input and filter. Keep in mind that the number of channels in the input and filter should be same. This will result in an output of 4×4. Fig. 5.6.1 shows convolution image.

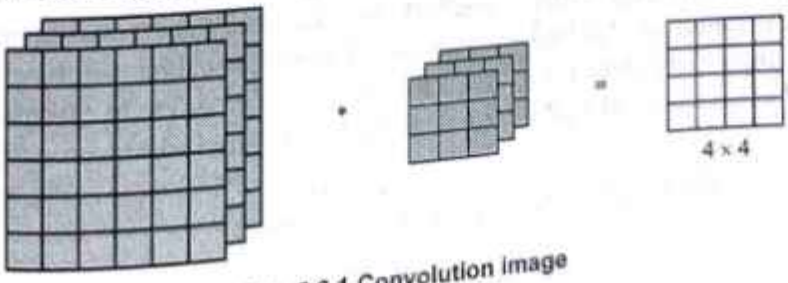


Fig. 5.6.1 Convolution image

- Since there are three channels in the input, the filter will consequently also have three channels. After convolution, the output shape is a 4×4 matrix. So, the first element of the output is the sum of the element-wise product of the first 27 values from the input (9 values from each channel) and the 27 values from the filter. After that we convolve over the entire image.
- Instead of using just a single filter, we can use multiple filters as well. How do we do that ? Let's say the first filter will detect vertical edges and the second filter will detect horizontal edges from the image. If we use multiple filters, the output dimension will change. So, instead of having a 4×4 output as in the above example, we would have a $4 \times 4 \times 2$ output (if we have used 2 filters) :

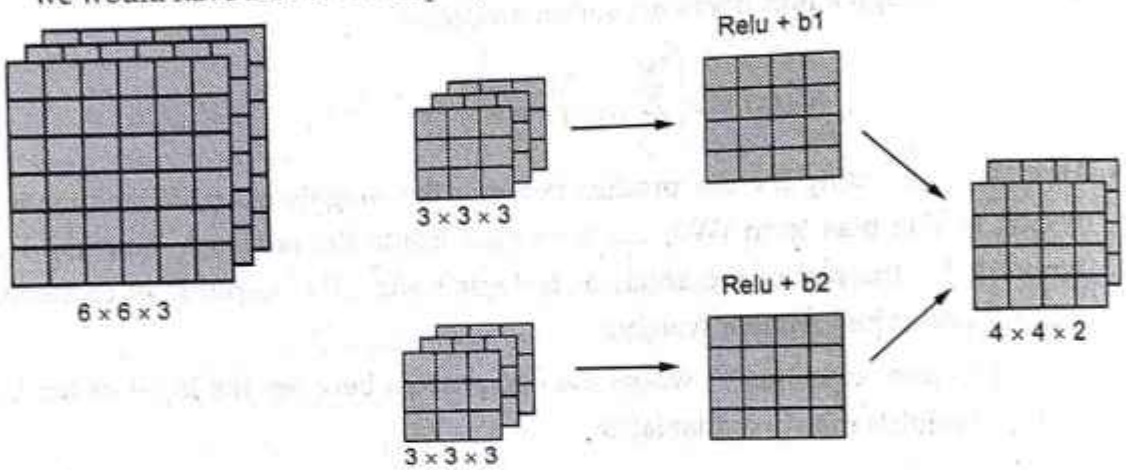


Fig. 5.6.2 Example of convolution over volume

- After a series of 3D convolutional layers, we need to 'flatten' the 3D tensor to a 1D tensor and add one or several dense layers to connect the output to the response variable.

5.7 SoftMax Regression

- The softmax function is nothing but a generalization of the sigmoid. The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. The output values are between the range [0, 1].
- When we have to deal with a classification with more than 2 possible levels, we use a generalization of the logistic regression function called softmax regression; a logistic regression class for multi-class classification tasks. In Softmax regression, we replace the sigmoid logistic function by the so-called softmax function.

$$P(y = j \mid z^{(i)}) = \phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^{k} e^{z_k^{(i)}}}$$

where we define the net input z as

$$z = w_1x_1 + \ldots + w_mx_m + b = \Sigma_{i=1}^{m} w_1x_1 + b = w^Tx + b$$

- The w is the weight vector, x is the feature vector of 1 training sample and b is the bias unit. A bias unit is an extra neuron added to each pre-output layer that stores the value of 1. Bias units aren't connected to any previous layer and in this sense don't represent a true activity. It is used in the case the sum of the weights is equal to zero. Now, this softmax function computes the probability that this training sample x(i) belongs to class j given the weight and net input z(i).
- Softmax is the only activation function recommended to use with the categorical cross-entropy loss function.

5.8 Deep Learning Frameworks

What is deep learning ?

- The term "deep" usually refers to the number of hidden layers in the neural network.
- Deep learning is a subset of machine learning, which is predicated on the idea of learning from example. In machine learning, instead of teaching a computer a massive list of rules to solve the problem, we give it a model with which it can evaluate examples and a small set of instructions to modify the model when it makes a mistake.
- The basic idea of deep learning is that repeated composition of functions can often reduce the requirements on the number of base functions (computational units) by a factor that is exponentially related to the number of layers in the network.
- Deep learning eliminates some of data pre-processing that is typically involved with machine learning.
- Fig. 5.8.1 shows relation between AI, ML and Deep learning.

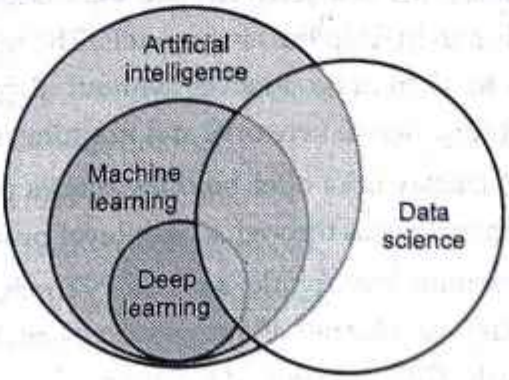


Fig. 5.8.1 Relation between AI, ML and Deep learning

- For example, let's say that we had a set of photos of different pets and we wanted to categorize by "cat" and "dog". Deep learning algorithms can determine which features (e.g. ears) are most important to distinguish each animal from another. In machine learning, this hierarchy of features is established manually by a human expert.
- In deep learning, a computer model learns to perform classification tasks directly from images, text or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.
- Deep learning classifies information through layers of neural networks, which have a set of inputs that receive raw data. For example, if a neural network is trained with images of birds, it can be used to recognize images of birds. More layers enable more precise results, such as distinguishing a crow from a raven as compared to distinguishing a crow from a chicken.
- Deep learning consists of the following methods and their variations :
  a) Unsupervised learning systems such as Boltzmann machines for preliminary training, auto-encoders, generative adversarial network.
  b) Supervised learning such as convolutional neural networks which brought technology of pattern recognition to a new level.
  c) Recurrent neural networks, allowing to train on processes in time.
  d) Recursive neural networks, allowing to include feedback between circuit elements and chains.

Deep Learning (DL) Frameworks

- A deep learning framework is a software package used by researchers and data scientists to design and train deep learning models. The idea with these frameworks is to allow people to train their models without digging into the algorithms underlying deep learning, neural networks and machine learning.
- **Deep Learning (DL) frameworks** offer building blocks for designing, training and validating deep neural networks through a high-level programming interface.
- Widely used deep learning frameworks such as PyTorch, TensorFlow, MXNet and others can also use GPU-accelerated libraries such as cuDNN and NCCL to deliver high-performance multi-GPU accelerated training.

Why Use a Deep Learning Framework ?

- They supply readily available libraries for defining layers, network types (CNNs, RNNs) and common model architectures.
- They can support computer vision applications; image, speech and natural language processing.
- They have familiar interfaces via popular programming languages such as Python, C, C++ and Scala.
- Many deep learning frameworks are accelerated by NVIDIA deep learning libraries such as cuDNN, NCCl and cuBLAS for GPU accelerated deep learning training.

5.8.1 TensorFlow

- TensorFlow is an end-to-end open source platform for deep learning. It was developed by the Google brain team and released in November 2015.
- It uses Python or JavaScript to provide a convenient front-end API for building applications, while executing those applications in high-performance C++.
- TensorFlow is popular for its TensorFlow core programs where it has two main actions.
  1. Building the computational graph in the construction phase.
  2. Running the computational graph in the execution phase.
- A tensor is a mathematical object and a generalization of scalars, vectors and matrices. A tensor can be represented as a multidimensional array. Let's understand how TensorFlow works.
  1. Its programs are usually structured into a construction phase and an execution phase.
  2. The construction phase assembles a graph that has nodes (ops/operations) and edges (tensors).
  3. The execution phase uses a session to execute ops (operations) in the graph.
- Features of Tensorflow are as follows :
  1. Faster debugging with Python tools.
  2. Dynamic models with Python control flow.
  3. Support for custom and higher-order gradients.
  4. TensorFlow offers multiple levels of abstraction, which helps to build and train models.

5. TensorFlow provides the flexibility and control with features like the Keras Functional API and model.
6. Well-documented so easy to understand.
7. Probably the most popular easy to use with Python.

- TensorFlow programs use a tensor data structure to represent all data, only tensors are passed between operations in the computation graph. TensorFlow allows developers to create dataflow graphs, structures that describe how data moves through a graph or a series of processing nodes.
- Each node in the graph represents a mathematical operation and each connection or edge between nodes is a multidimensional data array or tensor.
- TensorFlow applications can be run on most any target that's convenient : A local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.
- TensorFlow supports three main types of data types : Constants, variables and Placeholders.
- Constant in TensorFlow : tf.constant() function is used to create constant in TensorFlow. Here, it will always create a host/CPU tensor. Syntax is as follows :

  tf.constant (value, dtype=None, shape=Name, name='Constant')

  where

  a) **Value :** A constant value or list. This value is the constant value that is passed to the function.

  b) **Dtype :** It is the data type of the element. dtype is passed to the specify the datatype of the value which is passed as constant. If dtype is not mentioned, then it concludes the type from value, by default.

  c) **Shape :** It is optional dimensions given to the value.

  d) **Name :** Another optional attribute which assigns a name for the tensor, which contains the constant value.

  e) **Verify_shape :** It is a boolean value that helps enabling of verification of the shape of values.

5.8.2 Keras

- Keras is a high-level neural networks API, capable of running on top of Tensorflow, Theano and CNTK. It enables fast experimentation through a high level, userfriendly, modular and extensible API. Keras can also be run on both CPU and GPU.
- Keras is written in Python and supports multiple back-end neural network computation engines.
- Keras provides seven different datasets, which can be loaded in using Keras directly. These include image datasets as well as a house price and a movie review datasets.
- Salient features of Keras :
  1. Keras is a high-level interface and uses Theano or Tensorflow for its backend.
  2. It runs smoothly on both CPU and GPU.
  3. Keras supports almost all the models of a neural network - fully connected, convolutional, pooling, recurrent, embedding, etc. Furthermore, these models can be combined to build more complex models.
  4. Keras, being modular in nature, is incredibly expressive, flexible and apt for innovative research.
  5. Keras is a completely Python-based framework, which makes it easy to debug and explore.
- The model is the core Keras data structure. There are two main types of models available in Keras : The sequential model and the model class used with the functional API.
- The easiest way of creating a model in Keras is by using the sequential API, which lets us stack one layer after the other. The problem with the sequential API is that it doesn't allow models to have multiple inputs or outputs, which are needed for some problems.
- The functional API allows users to create the same models but offers more flexibility at the cost of simplicity and readability. It can be used with multiple input and output layers and shared layers, which enables to build complex network structures. When using the functional API, we always need to pass the previous layer to the current layer. It also requires the use of an input layer. The functional API is also often used for transfer learning.

- Keras provides seven different datasets, which can be loaded using Keras directly. These include image datasets, a house price and a movie review dataset. Keras is a high-level neural networks API running on top of Tensorflow.
- Demerits of Keras :
  1. Restricted to use only on smaller datasets.
  2. Slower performance since it is completely written in Python.
  3. The project repository is not as huge compared to TensorFlow.

5.8.2.1 Difference Between Keras and Tensorflow

Keras	Tensorflow
Keras is a high-level API which is running on top of TensorFlow.	TensorFlow is a framework that offers both high and low-level APIs.
Keras is used for small datasets as it is slower.	TensorFlow is used for high-performance models and massive datasets that require execution fast.
It can be used for low-performance models.	It is used for high-performance models.
Working with beginner-friendly projects with small datasets.	Rendering heavy projects with ease.
Readable and concise architecture.	Comparatively complex architecture.
Community support is minimal but growing.	Being one of the first deep learning frameworks that were developed, it has an extensive community and a rich reserve of readable material.

5.8.3 PyTorch

- PyTorch is an open source machine learning library for Python and is completely based on torch. It is primarily used for applications such as natural language processing. It was developed by Facebook's AI research lab.
- PyTorch uses dynamic computation, which allows greater flexibility in building complex architectures. PyTorch uses core Python concepts like classes, structures and conditional loops.
- PyTorch has a very simple interface like Python. It provides an easy way to use API.
- It allows developers to train a neural network model in a distributed manner. It provides optimized performance in both research and production with the help of native support for peer to peer communication and asynchronous execution of collective operation from Python and C++.

- The PyTorch core is used to implement tensor data structure, CPU and GPU operators, basic parallel primitives and automatic differentiation calculations.
- PyTorch is supported by many major cloud platforms such as AWS. With the help of prebuilt images, large scale training on GPU's and ability to run models in a production scale environment etc.; it provides frictionless development and easy scaling.

5.8.4 Caffe

- Convolutional Architecture for Fast Feature Embedding (Caffe) is a deep learning framework made with expression, speed and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors.
- Caffe is primarily a C++ library and exposes a modular development interface. Caffe offers interfaces for daily use by way of the command line, Python and MATLAB.
- Caffe processes data in the form of Blobs which are N-dimensional arrays stored in a C-contiguous fashion. Data is stored both as data we pass along the model and as diff, which is a gradient computed by the network.
- Data layers handle how the data is processed in and out of the caffe model. Pre-processing and transformation like random cropping, mirroring, scaling and mean subtraction can be done by configuring the data layer. Furthermore, pre-fetching and multiple-input configurations are also possible.

5.8.5 Shogun

- Shogun is an open source machine learning software library built in C++. Shogun was developed in 1999 by Soeren Sonnenburg and Gunnar Raetsch.
- It offers numerous algorithms and data structures for machine learning problems. The focus of Shogun is on kernel machines such as support vector machines for regression and classification problems. Shogun also offers a full implementation of hidden Markov models.
- Shogun is a community learning platform in every sense, anyone can use the toolbox to learn about ML and apply it to solve problems.
- Shogun is community-based and non-commercial. It is currently released under the GPLv3. It is also moving towards BSD compatibility with optional GPL (General Public License) parts. This means companies can use the code without having to turn their code-base to GPL.

5.9 Bias and Variance with Mismatched Data Distributions

- In the experimental practice we observe an important phenomenon called the bias variance dilemma.
- In supervised learning, the class value assigned by the learning model built based on the training data may differ from the actual class value. This error in learning can be of two types, errors due to 'bias' and error due to 'variance'.
- Fig. 5.9.1 shows bias-variance trade off.

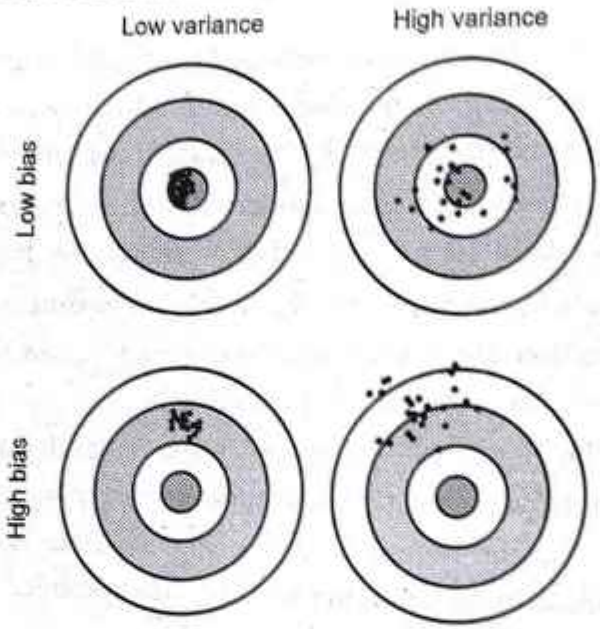


Fig. 5.9.1 Bias-variance trade off

- Give two classes of hypothesis (e.g. linear models and k-NNs) to fit to some training data set, we observe that the more flexible hypothesis class has a low bias term but a higher variance term. If we have parametric family of hypothesis, then we can increases the flexibility of the hypothesis but we still observe the increase of variance.
- The bias-variance-dilemma is the problem of simultaneously minimizing two sources of error that prevent supervised learning algorithm from generalizing beyond their training set :
  1. The bias is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs.
  2. The variance is error from sensitivity to small fluctuations in the training set. High variance can cause overfitting : Modeling the random noise in the training data, rather than the intended outputs.

- In order to reduce the model error, the designer can aim at reducing either the bias or the variance, as the noise components is irreducible.
- As the model increases in complexity, its bias is likely to diminish. However, as the number of training examples is kept fixed, the parametric identification of the model may strongly vary from one DN to another. This will increase the variance term.
- At one stage, the decrease in bias will be inferior to the increase in variance, warning that the model should not be too complex. Conversely, to decrease the variance term, the designer has to simplify its model so that it is less sensitive to a specific training set. This simplification will lead to a higher bias.

Example 5.9.1 *Explain the above Fig. 5.9.2 (a), (b) and (c).* **SPPU : Oct.-19, In Sem, Marks 5**

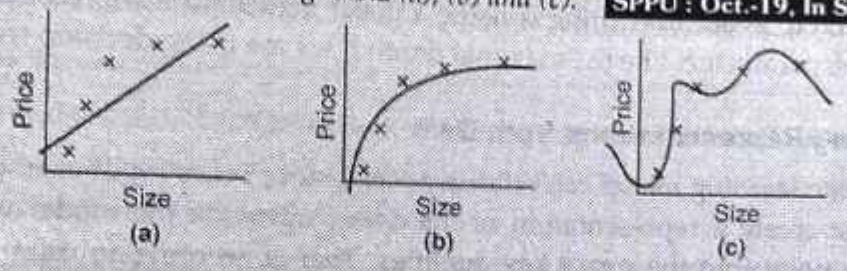
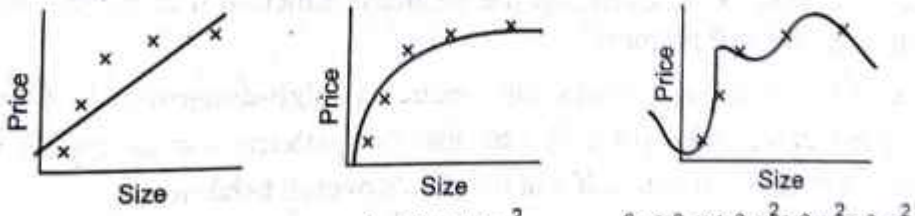


Fig. 5.9.2

Solution : Given Fig. 5.9.2 is related to overfitting and underfitting.

Underfitting (High bias and low variance) :

- A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data.
- It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data.



$\theta_0 + \theta_1 x$ **High bais (underfit)**

$\theta_0 + \theta_1 x + \theta_2 x^2$ **High bais (underfit)**

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_2 x^2 + \theta_2 x^2$ **High variance (overfit)**

Fig. 5.9.3

- In such cases the rules of the machine learning model are too easy and flexible to be applied on such minimal data and therefore the model will probably make a lot of wrong predictions.

- Underfitting can be avoided by using more data and also reducing the features by feature selection.

Overfitting (High variance and low bias) :

- A statistical model is said to be overfitted, when we train it with a lot of data.
- When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set.
- Then the model does not categorize the data correctly, because of too many details and noise.
- The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models.
- A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

5.10 Learning Representations from Data

- A machine learning model can't directly see, hear or sense input examples. Instead, we must create a representation of the data to provide the model with a useful vantage point into the data's key qualities. That is, in order to train a model, we must choose the set of features that best represent the data.
- Representation learning is a class of machine learning approaches that allow a system to discover the representations required for feature detection or classification from raw data. The requirement for manual feature engineering is reduced by allowing a machine to learn the features and apply them to a given activity.
- In representation learning, data is sent into the machine and it learns the representation on its own. It is a way of determining a data representation of the features, the distance function and the similarity function that determines how the predictive model will perform.
- Representation learning works by reducing high-dimensional data to low-dimensional data, making it easier to discover patterns and anomalies while also providing a better understanding of the data's overall behavior.
- Representation learning is concerned with training machine learning algorithms to learn useful representations.
- Deep neural networks can be considered representation learning models that typically encode information which is projected into a different subspace. These representations are then usually passed on to a linear classifier to, for instance, train a classifier.

- Representation learning can be divided into :
  a) **Supervised representation learning :** Learning representations on task A using annotated data and used to solve task B.
  b) **Unsupervised representation learning :** Learning representations on a task in an unsupervised way. These are then used to address downstream tasks and reduce the need for annotated data when learning news tasks.

5.10.1 Greedy Layer - Wise Unsupervised Pretraining

- Greedy algorithms break a problem into many components, then solve for the optimal version of each component in isolation. Unfortunately, combining the individually optimal components is not guaranteed to yield an optimal complete solution.
- Greedy layer-wise unsupervised pretraining relies on single-layer representation learning algorithm. Each layer is pretrained using unsupervised learning, taking the output of previous layer and producing as output a new representation of the data, whose distribution is hopefully simpler.

Why it is called Greedy layer-wise pretraining ?

- Greedy because it is a greedy algorithm that optimizes each piece of the solution independently
- Layer-wise because independent pieces are the layers of the network and training proceeds one layer at a time.
- Pretraining because it is only a first step before applying a joint training algorithm is applied to fine-tune all layers together.
- Unsupervised feature learning algorithm L, which takes a training set of examples and returns an encoder or feature function f. The X is raw input data.

$f \leftarrow$ Identity function

$\bar{X} = X$

for $f = 1 \ldots m$ do

$f^{(k)} = L(X)$

$f \leftarrow f^{(k)} \circ f$

$X \leftarrow f^{(k)}(X)$

end for

if fine-tuning then

$f \leftarrow \tau(f \cdot X \cdot Y)$

end if

Return f

- Greedy : Optimize each piece of the solution independently, on piece at a time.
- **Layer-wise :** The independent pieces are the layer of the network. Training proceeds once layer at a time, training the kth layer while keeping the previous ones fixed.
- **Unsupervised :** Each layer is trained with an unsupervised representation learning algorithm.

When and why does unsupervised pretraining ideas work ?

1. **Idea1 :** Choice of initial parameters for a deep NN can have a significant regularizing effect on the model.

- It remains possible that pretraining initializes the model in a location that would otherwise be inaccessible. For example : A region surrounded by areas where the cost function varies so much from one example to another that mini-batches give only a very noisy estimate of the gradient a region surrounded by areas where the Hessian matrix is so poorly conditioned that gradient descent methods must use very small steps.
- We cannot characterize the exactly what aspects of the pretrained parameters are retained during the supervised training stages.

2. **Idea2 :** Learning about the input distribution can help with learning about the mapping from input to output.

- Some features that are useful for the unsupervised task may also be useful for supervised tasks. This is not yet understood at a mathematical, theoretical level. Many aspects of this approach are highly dependent on the specific models used.
- For example, if we wish to add a linear classifier on top of pretrained features, the features must make the underlying classes linearly separable. This is another reason that simultaneous supervised and unsupervised learning can be preferable.

Disadvantage of Unsupervised Pretraining :

1. Unsupervised pretraining does not offer a clear way to adjust the strength of the regularization arising from the unsupervised stage. When we perform unsupervised and supervised learning simultaneously, instead of using the pretraining strategy, there is a single hyperparameter, usually a coefficient attached to the unsupervised cost, that determine shows strongly supervised objective will regularize the supervised model.
2. Two separate training phases has its own hyperparameters. The performance of the second phase cannot be predicted during the first phase, so there is a long delay between proposing hyperparameters for the first phase and being able to

update them using feedback from the second phase. Most principled approach : Use validation set error in the supervised phase to select hyperparameters of the pretraining phase

5.10.2 Transfer Learning and Domain Adaptation

- Transfer learning and domain adaptation refer to situations where what has been learned in one setting is exploited to improve generalization in another setting.
- Transfer learning, multitask learning and domain adaptation can be achieved via representation learning when there exist features that are useful for different settings or tasks, corresponding to underlying factors that appear in more than one setting.
- Two extreme form of transfer learning :
  1. **One-shot learning :** Only one example of transfer task is given for one-shot learning. It is possible because the representation learns cleanly separate the underlying classes during first stage. During the transfer learning stage, only one labeled example is needed to infer the label of many possible test examples that all cluster around the same point in representation space.
  2. **Zero-shot learning :** No labeled examples are given at all. It is only possible because additional information has been exploited during training.

11 Multi-Task Learning

- Multi-task learning is a sub-field of machine learning that aims to solve multiple different tasks at the same time, by taking advantage of the similarities between different tasks.
- Multitask learning is a way to improve generalization by pooling the examples arising out of several tasks.
- Different supervised tasks shared the same input and some intermediate-level representation.
- We can view multi-task learning as a form of inductive transfer. Inductive transfer can help improve a model by introducing an inductive bias, which causes a model to prefer some hypotheses over others.
- For instance, a common form of inductive bias is ℓ_1 regularization, which leads to a preference for sparse solutions.

5.11.1 Types of Multi-Task Learning

- Multi-tasking learning are of two types : hard or soft parameter sharing of hidden layers.
- **Hard parameter :** It is generally applied by sharing the hidden layers between all tasks, while keeping several task-specific output layers. Hard parameter sharing greatly reduces the risk of overfitting.
- Fig. 5.11.1 shows hard sharing of hidden layers.

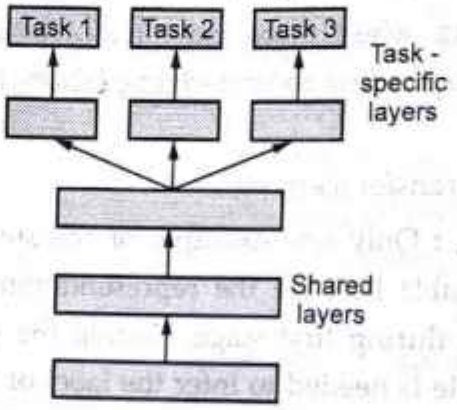


Fig. 5.11.1 Hard sharing of hidden layers

- **Soft parameter** sharing adds a constraint to achieve similarity among related parameters rather than sharing the same value. Furthermore, we penalize the difference in parameters across the models that we train for each task. By loosely connecting the shared space representations, this method, in contrast to rigid sharing, allows activities greater flexibility.
- Fig. 5.11.2 shows soft parameter sharing multi-tasking.

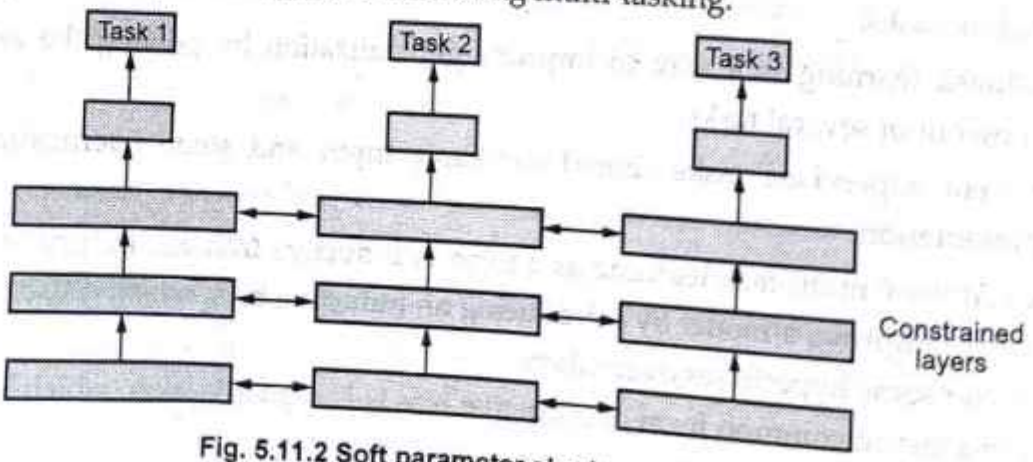


Fig. 5.11.2 Soft parameter sharing multi-tasking

- In soft parameter sharing on the other hand, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar.

5.11.2 Advantages of Multi-task Learning

- Reduce overfitting.
- Implicit data augmentation.
- Regularization : MTL acts as a regularizer by introducing an inductive bias.
- Focus on relevant features : If a task is very noisy or data is limited and high-dimensional, it can be difficult for a model to differentiate between relevant and irrelevant features. MTL can help the model focus its attention on those features that actually matter as other tasks will provide additional evidence for the relevance or irrelevance of those features.

5.12 End-to-End Deep Learning

- End-to-end (E2E) learning refers to training a possibly complex learning system represented by a single model that represents the complete target system, bypassing the intermediate layers usually present in traditional pipeline designs.
- End-to-end deep learning is the simplification of a processing or learning system into one neural network. Example is speech recognition model. It requires small data set.

Fig. 5.12.1 Traditional way

- End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data.
- It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc. The End-to-End deep learning way - large data set.
- Fig. 5.12.2 shows end-to-end deep learning.

Fig. 5.12.2 End-to-end deep learning

- Limitation :
  1. A huge amount of data is necessary.
  2. Highly efficient available modules cannot be used.
  3. Difficult to improve or modify the system

5.13 Introduction to CNN Models

5.13.1 LeNet - 5

- LeNet-5 is a convolutional neural network architecture that was created by Yan LeCunn in 1998. It includes 7 layers, excluding the input layer, which contains th trainable parameters called weights.
- **LeNet-5** consists of two parts : i) A convolutional encoder consisting of tw convolutional layers; and ii) A dense block consisting of three fully connecte layers.
- Fig. 5.13.1 shows architecture of LaNet-5.

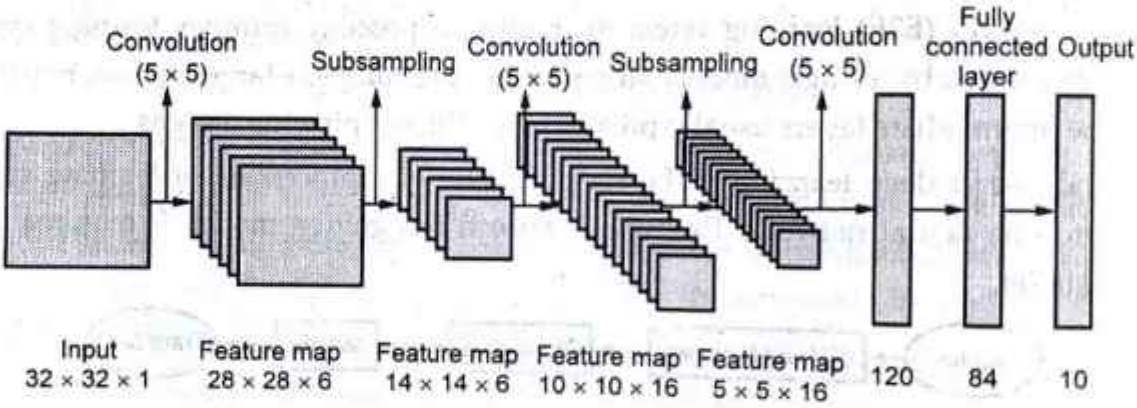


Fig. 5.13.1 LaNet-5 architecture

- Layer C1 is a convolutional layer with six feature maps where the size of the featur maps is 28 × 28;
- Layer S2 is a sub-sampling layer with six feature maps where the size of the featur maps is 14 × 14;
- Layer C3 is a convolutional layer with sixteen feature maps where the size of the feature maps is 10 × 10;
- Layer S4 is s sub-sampling layer with sixteen feature maps where the size of feature maps is 5 × 5;
- Layer C5 is a convolutional layer with 120 feature maps where the size of the feature maps is 1 × 1;

- Max pooling is applied between every two conv layers. After the tensors are flattened, two fully-connected (dense) layers are used. The output layer is a softmax layer to compute the softmax loss function for learning.
- AlexNet uses Rectified Linear Units (ReLU) instead of the tanh function, which was standard at the time. ReLU's advantage is in training time; a CNN using ReLU was able to reach a 25 % error on the CIFAR-10 dataset six times faster than a CNN using tanh.
- Use of multiple GPUs and parallel computing is highlighted in the training of AlexNet. The use of ReLU as the activation function for image classification by CNN.
- Multiple GPUs : GPUs were still rolling around with 3 gigabytes of memory. This was especially bad because the training set had 1.2 million images. AlexNet allows for multi-GPU training by putting half of the model's neurons on one GPU and the other half on another GPU. Not only does this mean that a bigger model can be trained, but it also cuts down on the training time.
- Overlapping Pooling : CNNs traditionally "pool" outputs of neighboring groups of neurons with no overlapping.
- AlexNet had 60 million parameters, a major issue in terms of overfitting. Two methods were employed to reduce overfitting : Data Augmentation and dropout.
- The results of AlexNet show that a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning.

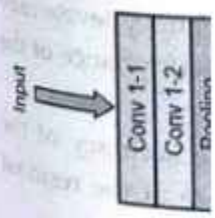
Advantages and Disadvantages of AlexNet

1. **Advantages of AlexNet**
   - AlexNet is considered as the milestone of CNN for image classification.
   - Many methods such as the conv and pooling design, dropout, GPU, parallel computing, ReLU is still the industrial standard for computer vision.
   - The unique advantage of AlexNet is the directly image input to the classification model.
   - The convolution layers can automatically extract the edges of the images, and fully connected layers learning these features.
   - Theoretically the complexity of visual patterns can be effective extracted by adding more conv layers.

2. **Disadvant**
   - AlexNe
     GoogLE
   - The use
   - Use nor
     effectiv
     method
   - The pe
     and Re

5.13.3 VGG - 16

- It is a Convol
  and Andrew
  Visual Geom
- VGG16 is a v
  of 16 convol
  Architecture.
  can be traine
- It is currentl
  from images.
  been used in
- VGG16 is an
  1000 images
  popular algo
- VGG-16 mai
- Fig. 5.13.3 sh



- Convolution
  The most im

2. **Disadvantages of AlexNet**
   - AlexNet is NOT deep enough compared to the later model such as VGG Net, GoogLENet, and ResNet.
   - The use of large convolution filters (5 * 5) is not encouraged.
   - Use normal distribution to initiate the weights in the neural networks cannot effective solve the problem of gradient vanishing, replaced by the Xavier method later.
   - The performance is surpassed by more complex models such as GoogLENet and ResNet.

5.13.3 VGG - 16

- It is a Convolutional Neural Network (CNN) model proposed by Karen Simonyan and Andrew Zisserman at the University of Oxford. The VGG model stands for the Visual Geometry Group from Oxford.
- VGG16 is a variant of VGG model with 16 convolution layers. VGGNet-16 consists of 16 convolutional layers and is very appealing because of its very uniform Architecture. Similar to AlexNet, it has only 3 × 3 convolutions, but lots of filters. It can be trained on 4 GPUs for 2-3 weeks.
- It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor.
- VGG16 is an object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7 % accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.
- VGG-16 mainly has three parts : Convolution, pooling and fully connected layers.
- Fig. 5.13.3 shows VGG-16 architecture.

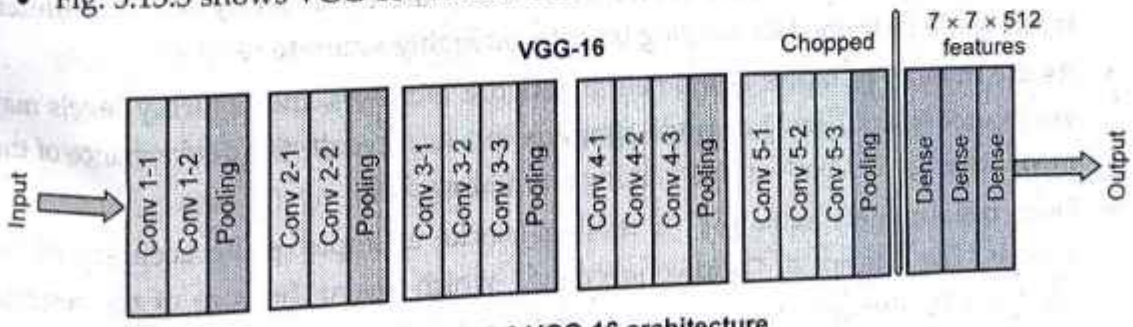


Fig. 5.13.3 VGG-16 architecture

- Convolution layer - In this layer, filters are applied to extract features from images. The most important parameters are the size of the kernel and stride.

- Pooling layer - Its function is to reduce the spatial size to reduce the number of parameters and computation in a network.
- Fully connected - These are fully connected connections to the previous layers as in a simple neural network.
- The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five max pooling layers and three dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer.
- VGG16 takes input tensor size as 224, 244 with 3 RGB channels. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3×3 filter with stride 1 and always used the same padding and max pool layer of 2×2 filter of stride 2.
- The convolution and max pool layers are consistently arranged throughout the whole architecture Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters.
- Three Fully-Connected (FC) layers follow a stack of convolutional layers : The first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels. The final layer is the soft-max layer.

5.13.4 Residual Networks

- Residual network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. ResNet has many variants that run on the same concept but have different numbers of layers. Resnet50 is used to denote the variant that can work with 50 neural network layers.
- When working with deep convolutional neural networks to solve a problem related to computer vision, machine learning experts engage in stacking more layers. These additional layers help solve complex problems more efficiently as the different layers could be trained for varying tasks to get highly accurate results.
- As the number of layers of the neural network increases, the accuracy levels may get saturated and slowly degrade after a point. As a result, the performance of the model deteriorates both on the training and testing data.
- Deep residual nets make use of residual blocks to improve the accuracy of the models. The concept of "skip connections," which lies at the core of the residual blocks, is the strength of this type of neural network.

- The skip connections in ResNet solve the problem of vanishing gradient in deep neural networks by allowing this alternate shortcut path for the gradient to flow through.
- **Residual Block :** Resnets are made up of smaller sections called residual blocks. Each residual block consists of 2 or 3 layers of neural network where we have an incoming activation to the first residual block. This activation is taken by the first layer and is also passed on to the next layer or layer further down in the resnet.
- Fig. 5.13.4 shows residual block.

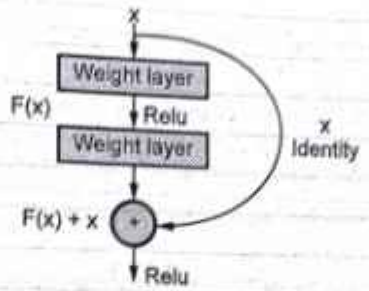


Fig. 5.13.4 Residual block

- In a residual network, each layer feeds to its next layer and directly to the 2-3 layers below it. The residual block consists of two 3×3 convolution layers and an identity mapping also called a shortcut connection. Each convolution layer is followed by a batch normalization layer and a ReLU activation function. An element-wise addition is performed between the identity mapping with the output of the last batch normalization.
- It provides an alternative path for the better flow of gradients during the backpropagation. It helps the earlier layers to learn better, which helps to improve the performance of the network.
- The identity mapping learns an identity function that ensures that the residual block performs as good as the lower layer.
- The residual network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connections are added and thus forming a residual network with 34-layers.

❑❑❑

Unit VI

6 Applications of ANN

Syllabus

Pattern classification - Recognition of Olympic games symbols, Recognition of printed Characters. Neocognitron - Recognition of handwritten characters. NET Talk : to convert English text to speech. Recognition of consonant vowel (CV) segments, texture classification and segmentation.

Contents

6.1 Applications Artificial Neural Network I - Pattern Classification

6.2 Applications Artificial Neural Network II - Recognition of Printed Characters

6.3 Applications Artificial Neural Network III - Neocognitron - Recognition of Handwritten Characters

6.4 Applications Artificial Neural Network IV - NETtalk - Convert English Text to Speech Application

6.5 Applications Artificial Neural Network V - Recognition of Consonant Vowel (CV) Segments, Texture Classification and Segmentation

6.1 Applications Artificial Neural Network I - Pattern Classification

- Neural networks is a best choice for pattern classification due to its ability of a multilayer feedforward neural network to form complex decision regions in the pattern space for classification.
- Many pattern recognition problems, especially character or other symbol recognition and vowel recognition, have been implemented using a multilayer neural network. It should be well noted that multilayer neural networks are not directly applicable for situations where the patterns are deformed or modified due to transformations such as translation, rotation and scale change, although some of them may work well even with large additive uncorrelated noise in the data.
- When the data is directly presentable to the classification network, the direct applications are highly successful.
- When the data is clean and ready to input then it is easy to process for multilayer neural network. Limits of classification performance will be reached if the symbols are degraded due to 'deformations', in the case of Olympic symbols or if the input corresponds to casually 'handwritten'.
- Characters in the case of character recognition, or if the 'knowledge' of the bidding sequence and the 'reasoning' power of human players have to be used in the bridge bidding problem.

Recognition of Olympic games symbols :

The problem - Recognition of Olympic games symbols.

The data -

Fig. 6.1.1 Olympic games symbols used for input to pattern classification network

- All the symbols are represented as black and white pixels on a 128 × 128 points grid. Although the symbols appear complex in terms of detail, each symbol represents a rigid-object-like behaviour. This behaviour ensures that the relative pixel positions of a symbol do not change even under severe degradation, such as translation, rotation and scaling.
- From above 128 × 128 points images set, a 16 × 16 element array is reconstructed as shown below.

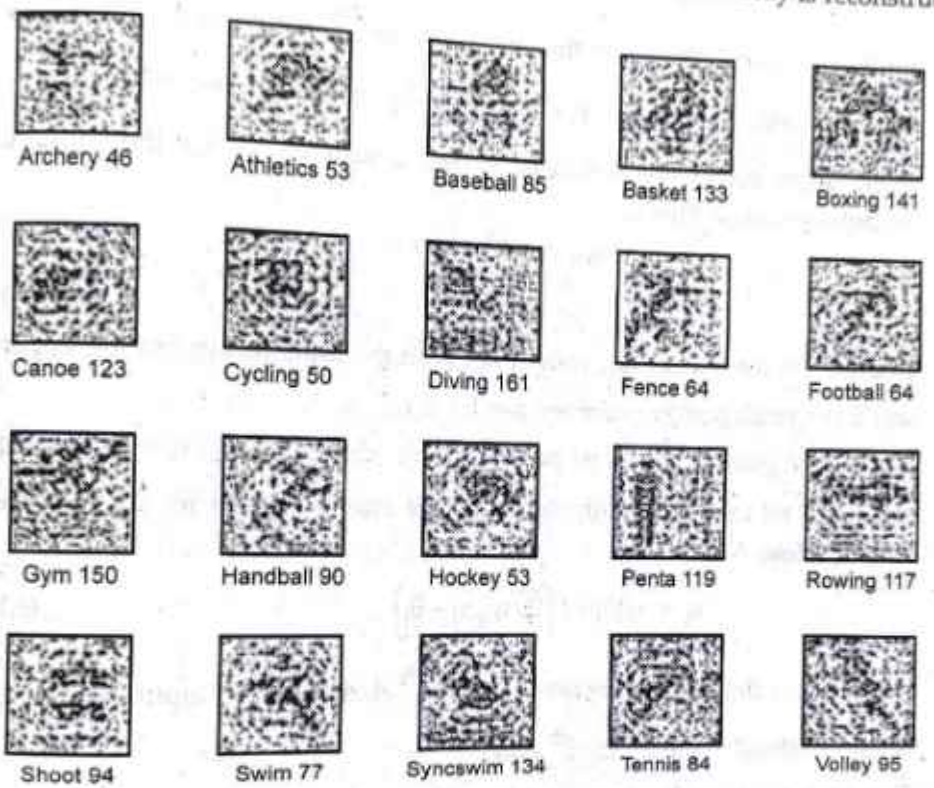


Fig. 6.1.2 A reconstructed images from 16 × 16 element sensor array. The class decision of the network is given along with the activation value of the winning pattern.

Pattern Classification Process

1. Here, Hamming network which performs template matching using neural principles is suitable for classification. The Hamming network is a maximum likelihood classifier for binary inputs and it performs correlation matching between the input and the stored templates.
2. It consists of two subnets as shown in below Fig. 6.1.2. The lower subnet consists of M(128 × 128) input units, each corresponding to a pixel in the given pattern and N output units corresponding to the N pattern classes, which in this case is 20.

3. In the lower subnet the connection weights between the input and output units are fixed in such a way that the network calculates the distance from the input pattern to each of the N stored pattern classes. The weights are given by below equation,

$$w_{ij} = a_{ij}/2,\ \theta_i = M/2,\ 1 < j \le M,\ i \le I \le N \qquad \ldots(6.1.1)$$

where w_{ij} is the connection weight from the input unit j to the output unit i in the lower subnet, θ_i is the threshold for the ith output unit and a_{ij}, is the element j of the pattern for the ith symbol. The values of a_{ij} are – 1 or 1.

4. In the upper subnet, the weights are fixed in such a way that the output units inhibit each other. That is,

$$v_{kl} = 1, \quad \text{for } k = 1$$
$$= -E, \text{ for } k \ne 1 \qquad \ldots(6.1.2)$$

where v_{kl} is the connection weight between the units K and *l* in the upper net and E is a small positive number, say E = 0.1.

5. When a bipolar pattern is presented for classification, the lower subnet calculates its matching score (si) with the stored pattern for the ith class as shown below,

$$s_i = s_i(0) = f\left(\sum_j w_{ij} x_j - \theta_i\right) \qquad \ldots(6.1.3)$$

where f (.) is the output function, x_j is the jth element of the input pattern and θ_i is the threshold value for the ith unit.

6. The output of the lower subnet is presented to the upper subnet, where a competitive interaction takes place among the units. The dynamics of the upper subnet is given by,

$$s_i(t+1) = f\left(s_i(t) - \varepsilon \sum_{k \ne i} s_k(t)\right),\ i = 1, 2, \ldots, N \qquad \ldots(6.1.4)$$

7. The competition continues until the output of only one unit remains positive and the outputs of all other units become negative. The positive unit corresponds to the class of the input pattern.

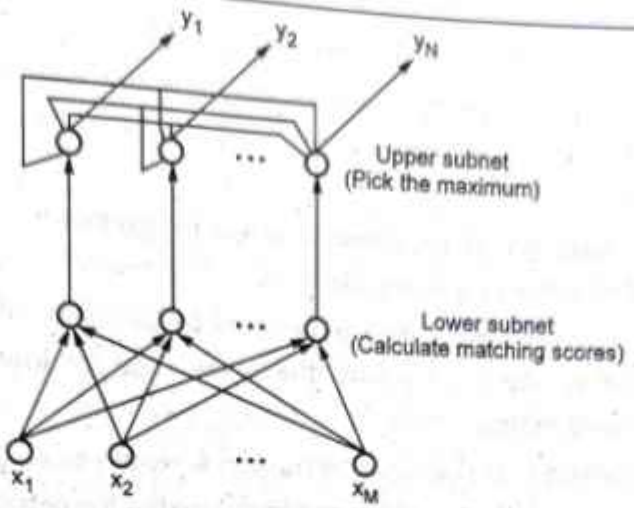


Fig. 6.1.3 The Hamming network. The input and output units are represented by x and y vectors, respectively.

8. For the set of degraded symbols given in Fig. 6.1.2, the correct classification performance is 100 % which is highly impressive, since it is difficult even for human to identify visually the discriminating features in many of these images. When the degradation is increased by reducing the resolution using an array of 8×8 elements, the recognition performance is only 13 out of 20.
9. The below Fig. 6.1.4 shows the number of patterns correctly classified out of 20.

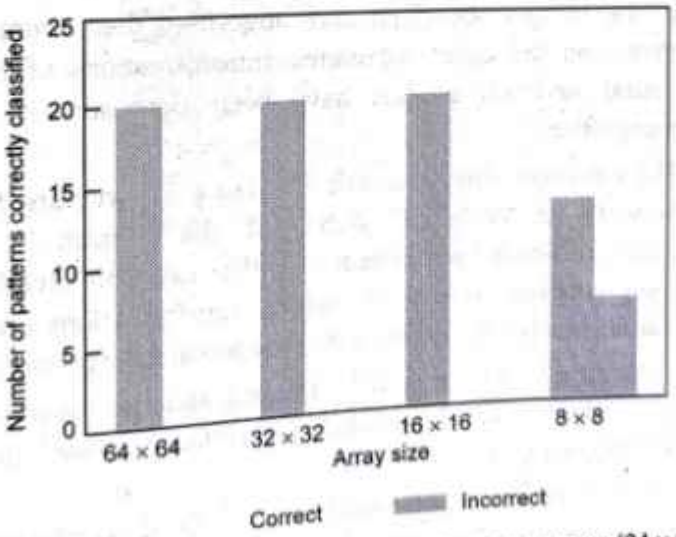


Fig. 6.1.4 Recognition performance summary with different sparse arrays (64 × 64, 32 × 32, 16 × 16 and 8 × 8 sensors). Graph shows the number of patterns correctly classified out of twenty patterns in each case.

From the above experiment and the summarized graph below are the conclusions,

(a) Many images for the 16 × 16 sensor array size case seem to have very few visual clues (Fig. 6.1.2) for human to recognize, but were recognized correctly by the network.

(b) The performance of the classifier degrades gradually with increasing image degradation due to sparsity and noise.

(c) The activation values of the winner units are indicative of the level of the image degradation, that is the greater the degradation the lower is the activation of the winning units.

(d) The matching scores obtained at the first layer are measures of similarity of the input pattern with each of the patterns stored in the network. But the activation level of each unit in the second layer is affected by the activation values of all other units.

Hence when an output unit becomes positive, its activation level not only reflects how close the input image is to the identified pattern, but also gives an idea of the degree of confidence given to this decision relative to other patterns stored in the network. Thus the activation value also reflects the complexity of the symbol set in terms of how close in shape the symbols are.

10. **Avoiding effects of object transformations -**

(i) If the images are clean and noise-free, then there exist methods to overcome the effects of metric transformations of the objects. Several neural network models have been proposed for invariant pattern recognition.

(ii) The networks which achieve invariance by structure, the structure of the network is designed such that the output is invariant to the transformations of interest. In the case of invariance by training, representative samples of various transformations are presented during training so that the network learns equivalent transformations.

Invariance by structure or by training assumes the existence of a fixed set of weights which provide invariance over the continuum of transformations.

(iii) It also assumes that a network can be trained to estimate this set of weights from examples. But invariances cannot be built as static functions in the structure. They have to be dynamically estimated from the data. Alternatively, one can first address the transformation invariance by feature extraction and then use these features as input to a classifier.

6.2 Applications Artificial Neural Network II - Recognition of Printed Characters

The problem - Recognition of printed characters.

The data - images (128 × 128 pts) of ten characters of alphabet.

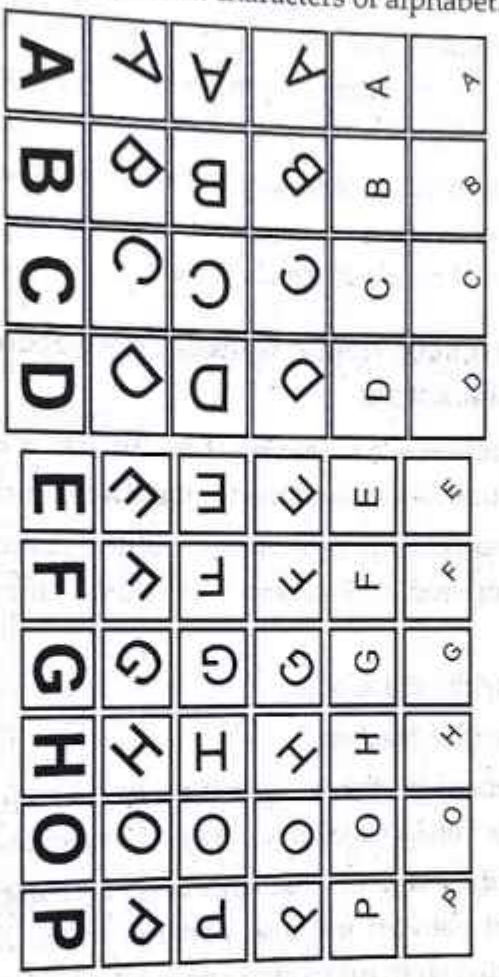


Fig. 6.2.1 Rotated, scaled and translated images of characters used for translation invariant recognition

Pattern recognition process

1. The ten characters and some transformed versions of these characters used in the current pattern recognition are shown in Fig. 6.2.1. In this case 100 % classification accuracies could be obtained for all the test data upto a scale reduction of 1/12 of the linear dimension of the original, that is, the reduced image is about 10 × 10 points.
2. Here the feature approach gives better transformation invariant recognition than in the case of the Olympic games symbols, since the objects are simpler in detail in the case of the printed characters of the alphabet.

3. The pattern classifications can be accomplished using neural network models for objects whose images are severely degraded by transformations and noise. But in all these cases the objects considered are assumed to be rigid, in the sense that there was no relative displacement in different parts of the object.
4. It should be noted that the above illustrations differ from the handwritten characters in which different parts of a character are deformed differently in each sample.
5. Thus the classification methods based on correlation matching or training a feedforward network using moment features are not useful for problems such as recognition of handwritten characters.

6.3 Applications Artificial Neural Network III - Neocognitron - Recognition of Handwritten Characters

1. The neocognitron is a **hierarchical multilayered neural network** proposed by Professor Kunihiko Fukushima for **handwritten character recognition**.
2. There are various different versions of the neocognitron. Two original basic versions proposed by Professor Fukushima differ in used **learning** principle namely,
   - Learning without a teacher
   - Learning with a teacher.
3. The first version of the neocognitron was based on the **learning without a teacher**. This version is often called **self-organized** neocognitron.
4. The **main advantage** of neocognitron is its ability to recognize correctly not only learned patterns but also patterns which are produced from them by using of partial shift, rotation or another type of distortion.
5. **Feature extraction in neocognitron - Hierarchical feature extraction** is the basic principle of the neocognitron. Hierarchical feature extraction consists in **distribution of extracted features to several stages**. The simplest features (usually only rotated lines) are extracted in the first stage and in each of the following stages the more complex features are extracted. In this process it is important fact that only informations obtained in the previous stage are used for feature extraction in the certain stage.

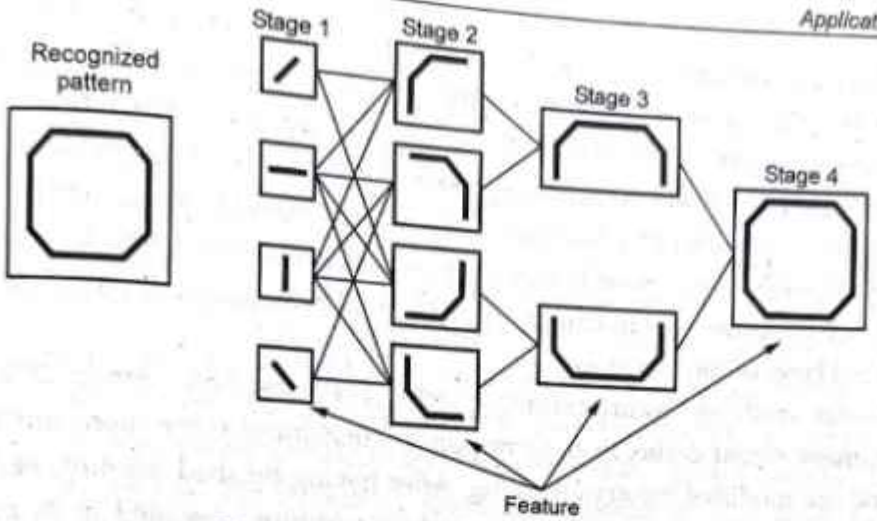


Fig. 6.3.1 Hierarchical Feature Extraction

6. Neocognitron architecture

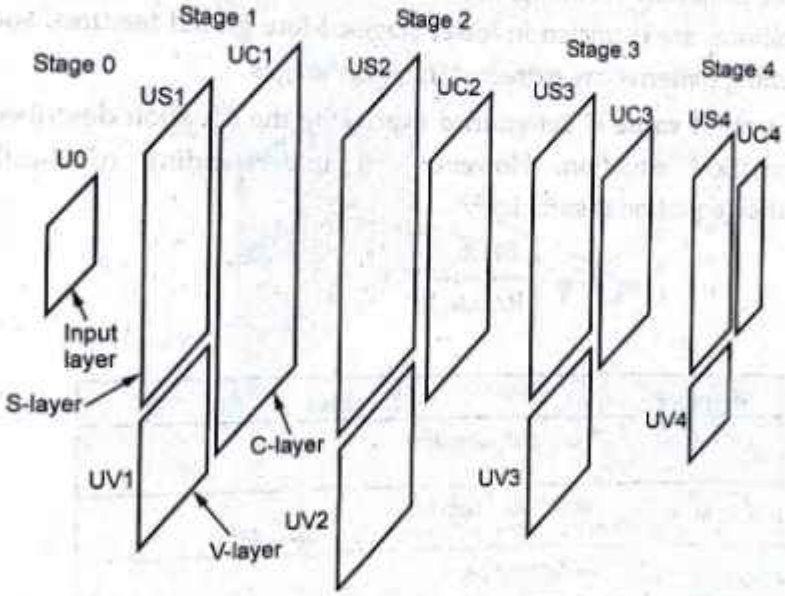


Fig. 6.3.2 Neocognitron network structure

Symbol	Denotes
U0	Input layer
USl	S-layer in the l-th stage of the network
UVl	V-layer in the l-th stage of the network
UCl	C-layer in the l-th stage of the network

i) A typical architecture of the network has the lowest stage which is the input layer consisting of two-dimensional array of cells, which correspond to photoreceptors of the retina. There are retinotopically ordered connections between cells of adjoining layers. Each cell receives input connections that lead from cells situated in a limited area on the preceding layer. Layers of "S-cells" and "C-cells" are arranged alternately in the hierarchical network. In the above figure the contrast-extracting layer is inserted between the input layer and the S-cell layer of the first stage.

ii) **S-cells** work as feature-extracting cells. They resemble simple cells of the primary visual cortex in their response. Their input connections are variable and are modified through learning. After having finished learning, each S-cell come to respond selectively to a particular feature presented in its receptive field. The features extracted by S-cells are determined during the learning process. Generally speaking, local features, such as edges or lines in particular orientations, are extracted in lower stages. More **global** features, such as parts of learning patterns, are extracted in higher stages.

S-cell output value is determined exactly by the equation described in below mathematical equation. However, for understanding of S-cell function simplified equation is sufficient,

$$u_s = \varphi^* \left[\frac{E(a,u_c)}{I(r,b,u_v)} \right]$$

Where,

Symbol	Denotes
u_s	S-cell output value
φ^*	Non-linear function
E	Excitatory part
a	a-weights
u_c	Output values of C-cells from connection areas
I	Inhibitory part
r	Selectivity
b	b-weight
u_v	V-cell output value

The S-cells ability to extract not only learned features but also deformed representations of these features is influenced by the choice of parameter denoted as **selectivity** to a great extent.

iii) **C-cells**, which resembles complex cells in the visual cortex, are inserted in the network to allow for positional errors in the features of the stimulus. The input connections of C-cells, which come from S-cells of the preceding layer, are fixed and invariable. Each C-cell receives excitatory input connections from a group of S-cells that extract the same feature, but from slightly different positions. The C-cell responds if at least one of these S-cells yield an output. Even if the stimulus feature shifts in position and another S-cell comes to respond instead of the first one, the same C-cell keeps responding. Thus, the C-cell's response is less sensitive to shift in position of the input pattern. One can also express that C-cells make a blurring operation, because the response of a layer of S-cells is spatially blurred in the response of the succeeding layer of C-cells.

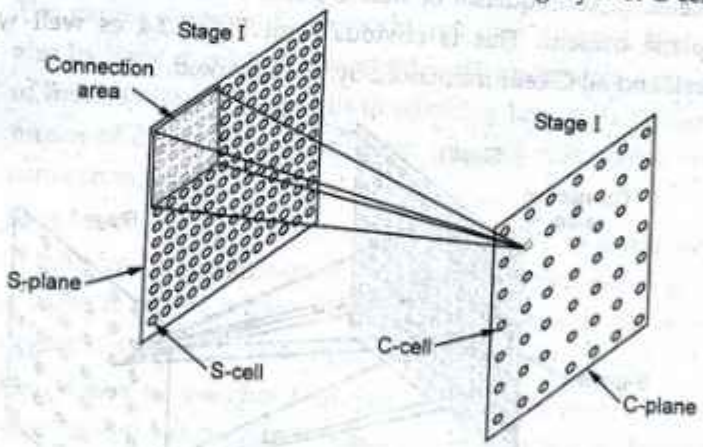


Fig. 6.3.3 Connection area of the C cell

C-cell output value depends on activity of **S-cells** from **connection area**. The greater number of active S-cells is or the greater their activities are the greater C-cell output value is. C-cell function is exactly described in below equation.

$$u_{cl}(n, k) = \Psi\left[\sum_{k=1}^{k_{sl}} J_l(K, k) \cdot \sum_{v \in Dl} d_l(v) \cdot u_{Sl}(n + v, K)\right]$$

l Serial number of the stage

n **Cell** position

k Serial number of the cell plane

K_{Sl} Number of cell planes in the previous **S-layer**

J_l Connection factor

v Position in the **connection area**

D_t Connection area of the **C-cell**

d_t d-weights (> = 0)

$$\Psi[x] = \frac{\varphi[x]}{1 + \varphi[x]}$$

$$\varphi[x] = \begin{cases} x \\ 0 \end{cases}$$

For C-cell to be active it is sufficient that at least one active S-cell is present in its connection area. With regard to overlapping of neighbouring C-cell connection areas activity of one S-cell affects activity of greater number of C-cells. In consequence of that C-plane contains a **blurred** representation of S-plane content. This is obvious from Fig. 6.3.4 as well where one active S-cell and all C-cells influenced by it are marked.

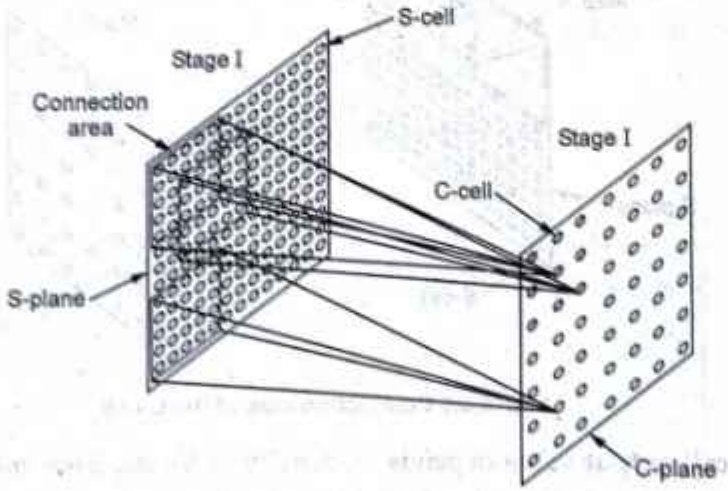


Fig. 6.3.4 C - cell function

Ability of C-cell to **compress** content of connection area in the certain way is the next consequence of C-cell function. Hence one can decrease the density of cells in C-layer to the half of density of cells in previous S-layer in some cases.

iv) Each layer of S-cells or C-cells is divided into sub-layers, called "**cell-planes**", according to the features to which the cells responds. The cells in each cell-plane are arranged in a two-dimensional array. A cell-plane is a group of cells

that are arranged retinotopically and share the same set of input connections. That is the connections to a cell-plane have a translational symmetry. As a result, all the cells in a cell-plane have receptive fields of an identical characteristic, but the locations of the receptive fields differ from cell to cell. The modification of variable connections during the learning progresses also under the restriction of shared connections.

v) V-cell function - Each **V-cell** in the neocognitron **evaluates outputs of C-cells** (or receptor cells) from the certain connection areas from previous C-layer (or input layer). **Size of connection areas** is the same for all V-cells and S-cells in one stage of the network and it is determined at construction of the network. **V-cell output value** represents average activity of cells from connection areas and it is used for inhibition of corresponding S-cell activity. Exact specification of V-cell function is described in mathematical description of its behaviour.

7. Weights and connections

i) The neocognitron is characteristic not only by large number of **cells** but also by large number of **connections**. These connections serve for **transfer of information** between cells in adjoining layers. Particular cell obtains by means of connections information from all cells which are located in its connection areas.

For each connection there is a **weight** by means of it one can affect amount of transferred information. If a connection is thought of as a pipeline with a valve it can be compare with weight assigned to the connection to a degree of opening of this valve.

ii) **Four types of weights** (a-weights, b-weights, c-weights and d-weights) exist in the neocognitron. Each of these types of weights is used for connections between two layers of different types. It is shown schematically in below figure.

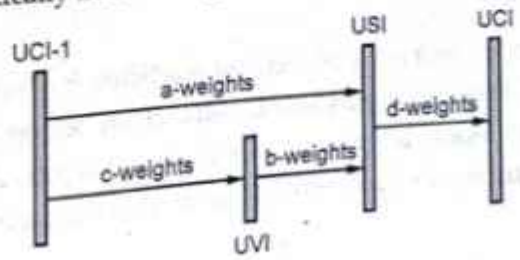


Fig. 6.3.5 Weights in the neocognitron

iii) **Weight sharing** is the next term being connected with weights. By this term one can designate the fact that all cells in one cell plane use the same weights for connections leading from cells in their connection areas. By the means of weight sharing it is guaranteed that all cells from one cell plane always extract the same feature.

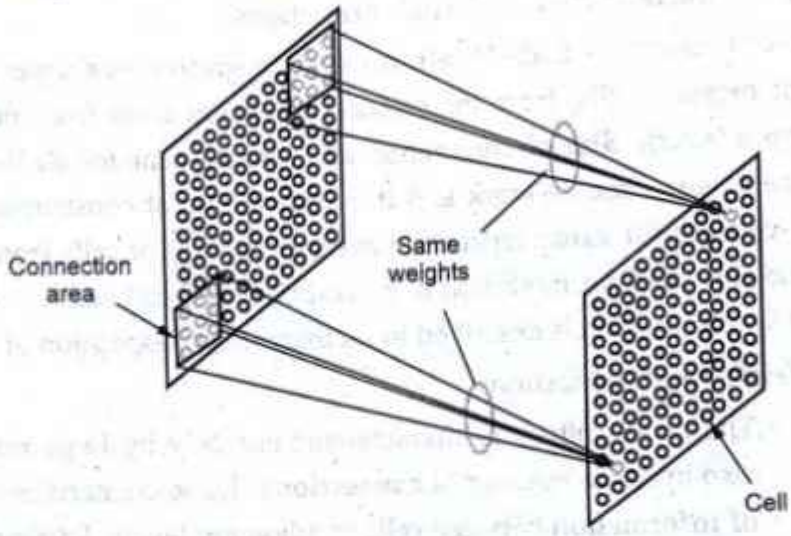


Fig. 6.3.6 Weight sharing

One can split the weights as shown in above **figure** according to the way which they are adjusted,

- Weights modified by learning
  - **a-weights**
  - **b-weights**
- Fixed weights
  - **c-weights**
  - **d-weights**

a-weights

The a-weights are the first type of weights **modified by learning**. These weights are used for connections **between S-cells and C-cells** which belong to their **connection areas**. **Features** extracted by S-cells are encoded in these a-weights. Adjusting of a-weights is performed during **learning** of the network according to the presented **training patterns**.

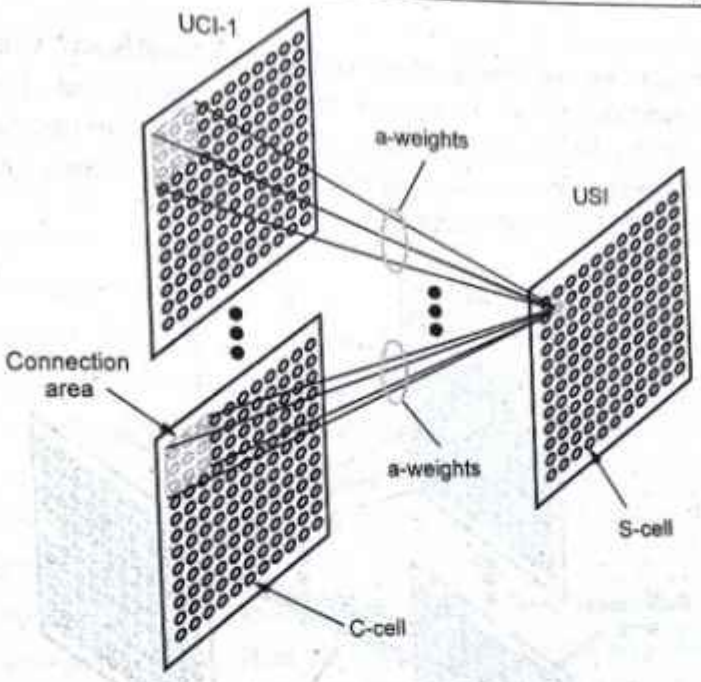


Fig. 6.3.7 a-weights

b-weights

- The b-weights are the second type of weights **modified by learning**. These weights are used for connections between **S-cells and corresponding V-cells**. Adjusting of b-weights is performed during **learning** of the network according to the presented **training patterns** as well.

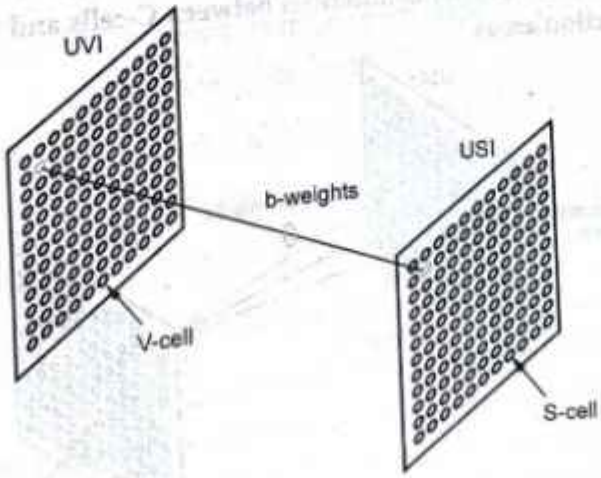


Fig. 6.3.8 b-weights

c-weights

- **Fixed** c-weights are used for connections **between V-cells and C-cells** which belong to their **connection areas**. Values of c-weights are determined at construction of the network. These weights are most often set up in such a way that they mostly reduce transfer of information from the periphery of connection area and towards to the center of area the degree of reduction decreases.

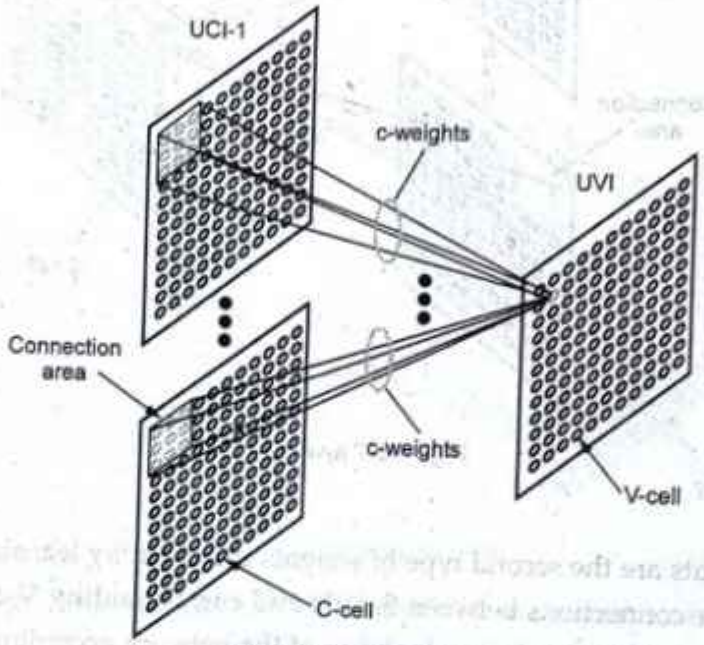


Fig. 6.3.9 c-weights

d-weights

- **Fixed** d-weights are used for connections **between C-cells and S-cells** which belong to their **connection areas**.

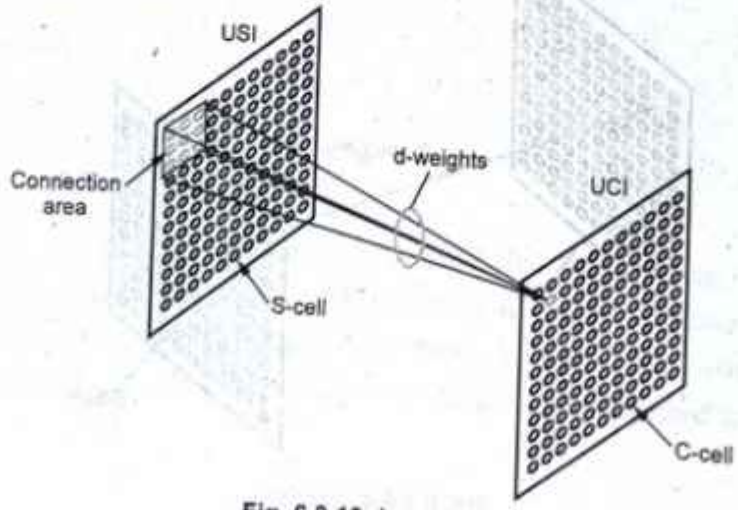


Fig. 6.3.10 d-weights

- As well as c-weights also d-weights are determined at construction of the network and again in such a way so as to reduce transfer of information from periphery of connection areas mostly.

8. Learning in neocognitron - Learning in this version of the network is controlled by a **teacher**. His task is to determine what features shall be extracted in particular stages of the network and to prepare corresponding training patterns before beginning of learning.

   Learning of the neocognitron proceeds stage by stage from the lowest stage of the network and it inheres in **adjusting of modifiable weights** (i.e. a-weights and b-weights) according to the response of already learned parts of the network to presented training patterns. For each S-plane in the network one training pattern is usually used and this pattern is usually necessary to present to the network only once.

   On the beginning of learning teacher have to set all **a-weights** and **b-weights** in the network to zero. Then he selects S-plane from layer US1 and in this cell plane he selects one of cells, so-called **seed cell**. Presentation of training pattern given for this S-plane into the **input layer** U0 is the next step. Finally teacher adjusts weights of the seed cell according to the equations mentioned in mathematical description of learning as shown below.

$$\Delta a_l(v,K,\hat{k}) = q_l \cdot c_l(v) \cdot u_{cl_1}(\hat{n} + v,K)$$

$$\Delta a_l(\hat{k}) = q_l \cdot u_{vl}(\hat{n})$$

l Serial number of the stage

a_l a-weights (>=0)

b_l b-weights (>=0)

c_l c-weights (>=0)

q_l Learning coefficient (>>0)

$\hat{n}$ **Seed cell** position

$\hat{k}$ Serial number of the cell plane with seed cell

Since **weight sharing** is used in the neocognitron adjusting of weights of all the other S-cells in the cell plane occurs simultaneously. If more training patterns for the selected S-plane exist then they are presented subsequently and process repeats. In opposite case it is moved to learning of the next S-plane.

9. **Recall** in the neocognitron inheres in evaluation of output values of all **cells** stage by stage. The result of this process is a decision to which of **learned** categories presented pattern belongs.

   The process of recall begins with presentation of pattern intended for recognition to the **input layer** U0. Then output values of **V-cells** in the layer UV1 are evaluated. **S-cells** from the layer US1 can **extract** the simplest **features** and **C-cells** from layer UC1 ensure decreasing of effect of extracted **features** shifts. The whole process repeats analogically for all the following layers of the network. After completion of recall output values of C-cells from the highest layer of the network correspond to measures of belonging presented pattern to categories which the particular C-cells represent.

10. Types of neocognitron - Various modifications of the neocognitron have been proposed to improve the recognition rate or to make biologically more natural. If backward (i.e., top-down) connections are added to the conventional neocognitron, for example, the network come to have an ability to recognize occluded patterns correctly and can restore the occluded parts of the patterns. Even if many patterns are presented simultaneously, it focuses **attention** to individual patterns one by one and recognizes them correctly.

6.4 Applications Artificial Neural Network IV - NETtalk - Convert English Text to Speech Application

1. Speech Signal - Speech signal is the output of a time - varying vocal tract system excited by a time-varying excitation signal. The vocal tract system, including the coupling of the nasal tract, can be accurately described in terms of the positions of the articulators such as tongue, lips, jaw, velum, etc.
2. Usually, the vocal tract system is approximately described in terms of the acoustic features such as the frequency response or the resonances (formants) and anti - resonances (anti-formants) of the system. These features are easier to extract from the signal than the articulatory parameters.
3. The excitation of the vocal tract system consists of broadly three categories,
   i) Voiced source (the quasiperiodic excitation due to the vibrating vocal folds).
   ii) Unvoiced source (the turbulent flow of air at a narrow constriction created in the vocal tract during production)

iii) Plosive source (the abrupt release of the pressure built up behind a closure in the vocal tract system). The voiced source is characterized by the periodicity (pitch period) and the change of the pitch period with time (intonation).

4. The short -time characteristics of the speech signal are represented by the short-time (10-20 ms) spectral features of the vocal tract system as well as the nature of excitation in the short-time segment. These are called **segmental features**. Supra- segmental features of speech are represented by the variation of the pitch period (intonation), the durations of different sound units, and the coarticulation reflecting the dependence of characteristics of one sound unit on the neighbouring sound units during speech production.

5. Speech is a sequence of sound units corresponding to a linguistic message. Below are important applications in speech area,

   a) **Speech recognition -** The objective is to determine the sequence of sound units from the speech signal so that the linguistic message in the form of text can be decoded from the speech signal.

   b) **Speech synthesis -** The objective is to determine the sequence of sound units corresponding to a text so that the given text message can be encoded into a speech signal.

   c) **Speaker identification -** The objective is to determine the identity of the speaker from the speech signal.

6. Problems in speech recognition problems -

   i) The main challenge and problem in the speech applications is processing of the speech signal in a manner similar to human auditory processing mechanism, so that features relevant to a particular task can be extracted.

   ii) The speech problem is further complicated by the fact that the message is conveyed not only through the segmental features but also by the suprasegmental features. It is the lack of understanding of these segmental and suprasegmental features and their extraction mechanism that makes the speech tasks extremely difficult for implementation by machines.

7. The NETtalk working

   i) It is a multilayer feedforward neural network (Fig. 6.4.1) developed to generate pronunciation units or phoneme code from an input text. The phoneme code is presented as input to a speech synthesizer to produce speech corresponding to the text.

ii) The network consists of an input layer of 7 * 29 binary units, corresponding to 7 input text characters including punctuation, one hidden layer with 80 units and one output layer with 26 units corresponding to 26 different phonetic units.

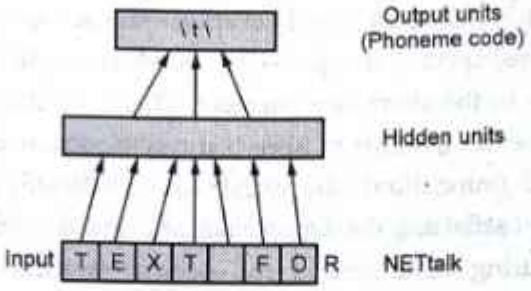


Fig. 6.4.1 NETtalk - feedforward network to convert English text to speech

iii) The network takes as input 7 consecutive characters at a time and produces the phoneme corresponding to the pronunciation of the letter at the centre of the input string. Each character of the input is represented as a 29-bit string. The network was trained on 1024 words from a set of English phoneme exemplars giving the pronunciation of words, m d was tested using different text sentences. The system produces a string of phonemes for a given input text. The speech produced by a synthesizer using these strings of phonemes as input was intelligible, although the quality was poor.

iv) The network was capable of distinguishing between vowels and consonants. On a new text the network achieved a generalization accuracy of 78 % in phonetic transcription. The system merely demonstrated the ability of a neural network to capture the letter to sound rules from input-output data. The quality will be obviously poor, as it cannot capture the significant supresegmental features which are essential for producing natural sounding synthetic speech.

6.5 Applications Artificial Neural Network V - Recognition of Consonant Vowel (CV) Segments, Texture Classification and Segmentation

1. Consonant-Vowel (CV) utterance typically forms a production unit (syllable) in speech and hence several attempts have been reported for recognition of CV utterances. Since these are dynamic sounds, the spectral patterns change with time. Each utterance of a CV unit is represented as a temporal sequence of

spectral vectors. Each spectral vector corresponding to a fixed 10 ms segment may be represented using 16 log spectral coefficients on a mel-frequency scale or using the corresponding mel-scale cepstral coefficients.

2. The number of spectral vectors per CV utterance generally varies. But usually a fixed duration (50-200 ma) segment of CV enclosing the vowel onset, the transition to vowel and some steady part of the vowel, is used to represent a CV unit. The CV units are thus temporal sequence patterns and hence static pattern recognition networks like Multilayer Feedforward Neural Network (MLFFNN) are not suitable for recognition of these units.

3. Moreover, discriminability among these CV units is low due to domination of the vowel context. Straight forward method to perform sequence recognition is to view the temporal sequence of the spectral vectors as a two-dimensional spatial input pattern for a MLFTNN. The conventional backpropagation learning can then be used to train the network. A better approach for CV recognition is through Time Delay Neural Networks (TDNN). TDNN is a MLFFNN with its input consisting of time-delayed input frames of data. The input to the intermediate hidden layers also consists of time-delayed outputs of the preceding layer. Below figure illustrates the idea of a time-delay,

(See Fig. 6.5.1 on next page)

In the Fig. 6.5.1 the neural network is applied for classification of three CV units lbl, Id and Igl. The time sequence is reflected in, the computation of block of three frames of data at a time, with two frames overlapping for successive blocks. The Fig. 6.5.1 shows multiple copies of the TDNN aligned with adjacent input spectral vectors. The first TDNN is shown in the boxes marked by thick lines. In this case each utterance has 15 frames of data, each frame consisting of 16 spectral components. For this, 13 different copies of TDNN are created. These include 13 copies of the input layer, nine copies of the first hidden layer, and only one second hidden layer and one output layer. The output layer has three units corresponding to the three classes Ibl, ldl, and Igl. For each TDNN, each unit in a layer is connected to all the units in the layer below it. For each TDNN there are 16×3 units in the input layer, 8×5 units in the first hidden layer and 9×3 units in the second hidden layer and 3 units in the output layer. Multiple copies of the TDNN as shown in the Fig. 6.5.1 enable the entire history of the network activity to be present at the same time. This allows the use of the backpropagation learning algorithm to train the network.

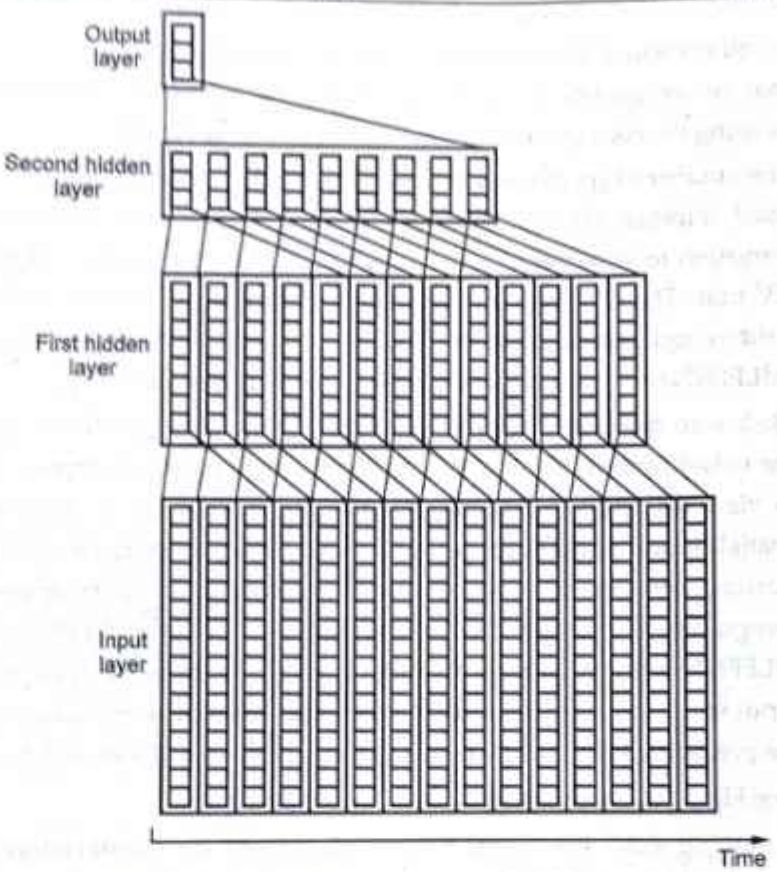


Fig. 6.5.1 Time delay neural network architecture

4. The 16 log spectral coefficients for each 10 ms frame are normalized using the average of each coefficient for an the 15 frames in the utterance and the coefficients are mapped into the range [– 1, 1]. The normalized values are given as input to the TDNN network. The speech data for the three classes was excised from continuous speech in Japanese and a database of about 800 CV units was collected for a given speaker. The TDNN was able to discriminate the three classes with an accuracy of 98.5 %. Considering the fact that the data for each class has significant variation due to contextual effects, this result is impressive. Extending this network model for large number of CV classes requires modular approach, where it is necessary to distinguish the groups of CV classes first before the individual classes can be identified. Additionally, because of the large size of the network, the training of the network will be slow. It will also be difficult to collect sufficient data to obtain good generalization performance from such a type of large networks.

Review Questions with Answers

1. *What is the central concept of pattern recognition ? How ANN plays role in pattern recognition ?* **(Refer section 6.1)**
2. *How printed characters are recognized using ANN ?* **(Refer section 6.2)**
3. *Write a note on Neocognitron.* **(Refer section 6.3)**
4. *Explain NETtalk application and use ANN in NETtalk application.* **(Refer section 6.4)**
5. *Brief on vowel and consonants segment recognition, texture classification.* **(Refer section 6.5)**

❑❑❑

SOLVED MODEL QUESTION PAPER (In Sem)

Artificial Neural Network

T.E. (AI&DS) Semester - VI (As Per 2019 Pattern)

Time : 1 Hour] **[Maximum Marks : 30**

N. B. :

i) Attempt Q.1 or Q.2, Q.3 or Q.4.

ii) Neat diagrams must be drawn wherever necessary.

iii) Figures to the right side indicate full marks.

iv) Assume suitable data, if necessary.

Q.1 a) *Draw and explain architecture of single layer feed forward network.* **(Refer section 1.5.1)** **[5]**

b) *What is activation functions ? Explain any one activation functions.* **(Refer section 1.6)** **[5]**

c) *Explain structure and working of biological neural network.* **(Refer section 1.3)** **[5]**

OR

Q.2 a) *Discuss history of neural network.* **(Refer section 1.2)** **[5]**

b) *Explain multilayer perceptron.* **(Refer section 1.8.2)** **[5]**

c) *What is recurrent neural network ? Explain.* **(Refer section 1.5.3)** **[5]**

Q.3 a) *What is gradient decent rules ? Explain vanishing gradient problem.* **(Refer section 2.5)** **[6]**

b) *What is auto-associative memory ? Explain difference between Auto-associative memory and hetero-associative memory.* **(Refer section 2.1)** **[9]**

OR

Q.4 a) *What is backpropagation ? Explain advantages and disadvantages of backpropagation.* **(Refer section 2.7)** **[6]**

b) *Define supervised learning. Explain advantages and disadvantages of supervised learning. Compare supervised and unsupervised learning.* **(Refer section 2.6)** **[9]**

SOLVED MODEL QUESTION PAPER (End Sem)

Artificial Neural Network

T.E. (AI&DS) Semester - VI (As Per 2019 Pattern)

Time : 2 $\frac{1}{2}$ Hours] **[Maximum Marks : 70**

N. B. :

i) ***Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.***

ii) ***Neat diagrams must be drawn wherever necessary.***

iii) ***Figures to the right side indicate full marks.***

iv) ***Assume suitable data, if necessary.***

Q.1 a) *Discuss use of hopfield networks in associative learning.* **(Refer section 3.1)** **[10]**

b) *What are advantages and disadvantages of simulated annealing ?* **(Refer section 3.2)** **[8]**

OR

Q.2 a) *Explain tasks of feedforward ANN in pattern recognition.* **(Refer section 3.4)** **[8]**

b) *Explain adaptive resonance learning with its applications.* **(Refer section 4.1.2)** **[10]**

Q.3 a) *Write a note on competitive learning.* **(Refer section 4.1.1)** **[5]**

b) *How SOM works ?* **(Refer section 4.2)** **[12]**

OR

Q.4 a) *What are features of ART models.* **(Refer section 4.1.2)** **[5]**

b) *List advantages and disadvantages of SOM.* **(Refer section 4.2)** **[4]**

c) *Explain learning vector quantization.* **(Refer section 4.2)** **[8]**

Q.5 a) *Draw and explain basic structure of CNN.* **(Refer section 5.2)** **[6]**

b) *Write short note on SoftMax regression.* **(Refer section 5.7)** **[6]**

c) *Explain any one CNN model.* **(Refer section 5.13)** **[6]**

OR

Q.6 a) *What is multi-task learning ? Explain types of multi-task learning.* **(Refer section 5.10)** **[6]**

b) *What is deep learning ? Explain difference between keras and tensorflow.* **(Refer section 5.8)** [6]

c) *What is convolution operation ? Explain in detail.* **(Refer section 5.2)** [6]

Q.7 a) *What is the central concept of pattern recognition ? How ANN plays role in pattern recognition ?* **(Refer section 6.1)** [6]

b) *Write a note on Neocognitron.* **(Refer section 6.3)** [6]

c) *How printed characters are recognized using ANN ?* **(Refer section 6.2)** [5]

OR

Q.8 a) *List and discuss any two pattern recognition applications where ANN plays key role.* **(Refer section 6.1)** [6]

b) *Explain NETtalk application and use ANN in NETtalk application.* **(Refer section 6.4)** [6]

c) *Brief on vowel and consonants segment recognition, texture classification.* **(Refer section 6.5)** [5]

❑❑❑