CHAPTER 1

INTRODUCTION

NEURAL NETWORKS AND DIGITAL IMAGE PROCESSING

1.1 Introduction

Artificial Neural Networks (ANNs) are computational modeling tools that have found extensive acceptance in many disciplines for modeling complex real-world problems. ANNs may be defined as comprised of densely interconnected adaptive simple elements (called artificial neurons or nodes) that are capable of performing massively parallel computations for data processing and knowledge representation. Although ANNs are drastic abstractions of the biological counterparts, the idea of ANNs is not to replicate the operation of the biological systems but to make use of what is known about the functionality of the biological networks for solving complex problems. The attractiveness of ANNs comes from the remarkable information processing characteristics of the biological system such as nonlinearity, high parallelism, robustness, fault and failure tolerance, learning capability, ability to handle imprecise and fuzzy information, and their ability to generalize. Artificial models possessing such characteristics are desirable because (i) nonlinearity allows better fit to the data, (ii) noise-insensitivity provides accurate prediction in the presence of uncertain data
and measurement errors, (iii) high parallelism implies fast processing and
hardware failure-tolerance, (iv) learning and adaptivity allow the system
to update (modify) its internal structure in response to changing
environment, and (v) generalization enables application of the model to
unlearned data. The main objective of ANN-based computing
(neurocomputing) is to develop mathematical algorithms that will enable
ANNs to learn by mimicking information processing and knowledge
acquisition in the human brain. ANN-based models are empirical in
nature, however they can provide practically accurate solutions for
precisely or imprecisely formulated problems and for phenomena that are
only understood through experimental data and field observations. In
microbiology, ANNs have been utilized in a variety of applications
ranging from modeling, classification, pattern recognition, and
multivariate data analysis (Basheer and Hajmeer, 2000).

One of the recently emerged applications of ANN is digital image
processing. Interest in digital image processing stems from two principal
application areas: improvement of pictorial information for human
interpretation; and processing of image data for storage, transmission, and
representation for autonomous machine perception. An image may be
defined as a two dimensional function, \( f(x,y) \), where \( x \) and \( y \) are spatial
coordinates, and the amplitude of \( f \) at any pair of coordinates \( (x, y) \) is
called the intensity or gray level of the image at that point. When \( (x,y) \) and
the amplitude values of \( f \) are all finite, discrete quantities, it is called as a
digital image. The field of digital image processing refers to processing
digital images by means of a digital computer. A digital image is
composed of a finite number of elements each having a particular location
and value. These elements are referred to as picture elements, image elements, pels and pixels. The areas of application of digital image processing are wide and varied (Gonzalez and Woods, 2002).

1.2 What is a Neural Network?

Work on artificial neural networks has been motivated right from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital computer. The brain is a highly complex, nonlinear and parallel information processing system. It has the capability to organize its structural constituents, known as neurons, so as to perform certain computations many times faster than the fastest digital computer in existence today. At birth, a brain has great structure and the ability to build its own rules through experience. One of the best examples is the acquiring of specific natural language as the mother tongue. Indeed, experience is built up over time, with the most dramatic development of the human brain taking place during the first two years from birth; the development continues well beyond that stage (Haykin, 2003).

A developing neuron is synonymous with a plastic brain: plasticity permits the developing nervous system to adapt to its surrounding environment. Just as plasticity appears to be essential to the functioning of neurons as information-processing units in the human brain, so it is with neural networks made up of artificial neurons. In its most general form, neural network is a machine that is designed to model the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic
components or is simulated in software on a digital computer. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as neurons or processing units. A neural network can be considered as a massively distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network from its environment through a learning process
- Interneuron connection strengths, known as synaptic weights, are used to store acquired knowledge (Haykin, 2003).

It is apparent that a neural network derives its computing power through, (i) its massively parallel distributed structure and (ii) its ability to learn and therefore to generalize. Generalization refers to the neural network producing reasonable outputs for inputs not encountered during training (learning). These two information processing capabilities make it possible for neural networks to solve complex problems that are currently intractable (Haykin, 2003).

The use of neural networks offers the following properties and capabilities (Hagan et. al., 2002). An artificial neuron can be linear or nonlinear. A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. Another capability of the neural network is its input-output mapping property (Haykin, 2003). The neural network learns from the examples by constructing an input-output mapping for the problem. Neural networks have a built in capability to adapt their synaptic weights to change in the surrounding environment. In particular, a neural
network trained to operate in a specific environment can easily be retrained to deal with minor changes in the operating environmental conditions. Another property of neural network is its 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. This latter information may be used to reject ambiguous patterns and thereby improve the classification performance of the network (Haykin, 2003).

Knowledge is represented by the very structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextual information is dealt with naturally by a neural network. As indicated earlier, a neural network, implemented in hardware, has the potential to be inherently fault tolerant, or capable of robust computation, in the sense that its performance degrades gracefully under adverse operating conditions. For example, if a neuron or its connecting links are damaged, recall of a stored pattern is impaired in quality. However, due to the distributed nature of information stored in the network, the damage has to be extensive before the overall response of the network is degraded seriously. Thus in principle, a neural network exhibits a graceful degradation in performance rather than catastrophic failure. The massively parallel nature of a neural network makes it potentially fast for the computation of certain tasks. This feature makes a neural network well suited for implementation using very-large-scale-integrated (VLSI) technology (Haykin, 2003). An important property of neural network is its uniformity of analysis and design. The same notation
is used in all domains involving the application of neural networks. This feature manifests itself in different ways.

- Neuron in one form or another, represent an ingredient common to all neural networks
- This commonality makes it possible to share theories and learning algorithms in different applications of neural network.
- Modular networks can be built through a seamless integration of modules.

The design of a neural network is motivated by analogy with the brain, which is the living proof that fault to learnt parallel processing is not only physically possible but also sufficiently fast and powerful (Haykin, 2003).

1.2.1 Human Brain

The human nervous system may be viewed as a three-stage system as shown in Fig.1.1. Central to the system is the brain, represented by the neural net, which continually receives information, perceives it and make appropriate decisions. Two sets of arrows are shown in the figure. Those pointing from the left to right indicate the forward transmission of information-bearing signals through the system.

![Block diagram representation of nervous system.](image)

*Fig. 1.1 Block diagram representation of nervous system.*
The arrows pointing from right to left signify the presence of feedback in the system. The receptors convert stimuli from the human body or the external environment into electrical impulse that convey information to the neural net. The effectors convert electrical impulse generated by the neural net into discernible responses as system outputs.

The human nervous system consists of billions of neurons of various types and lengths relevant to their location in the body (Schalkoff, 1997). The struggle to understand the brain has made easier because of the pioneering work of Ramon y Cajal, who introduced the idea of neurons as structural constituents of brain (Haykin, 2003). Typically, neurons are five to six orders of magnitude slower than silicon logic gates. However, the brain makes up for the relatively slow rate of operation of a neuron by having a truly staggering number of neurons with massive interconnections between them. It is estimated that there are approximately 10 billion neurons in the human cortex, and 60 trillion synapses or connections. The net result is that the brain is an enormously efficient structure. A neuron has three principal components: the dendrite, the cell body and the axon. The dendrites are tree-like receptive networks of nerve fibres that carry electrical signals into the cell body as in Fig.1.2. The cell body has a nucleus that contains information about heredity traits, and a plasma that holds the molecular equipment used for producing the material needed by the neuron (Jain et. al., 1996). The dendrites receive signals from other neurons and pass them over to the cell body. The total receiving area of the dendrites of a typical neuron is approximately 0.25 mm² (Zupan and Gasteiger, 1993). The cell body effectively sums and thresholds these incoming signals. The axon is a single long fibre that carries the signal
from the cell body out to other neurons. The point of contact between an axon of one cell and a dendrite of another cell is called a synapse. It is the arrangement of neurons and the strengths of the individual synapses, determined by a complex chemical process that establishes the function of the neural network (Haykin, 2003).

Fig. 1.2 The Pyramidal Cell
Synapses are elementary structural and functional units that mediate the interaction between neurons. The most common kind of synapse is a chemical synapse, which operates as follows: A presynaptic process liberates a transmitter substance that diffuses across the synaptic junction between neurons and then acts on a post synaptic process. Thus a synapse converts a presynaptic electrical signal into a chemical signal and then back into a post synaptic electrical signal. In traditional descriptions of neural organization, it is assumed that a synapse is a simple connection that can impose excitation or inhibition, but not both simultaneously on the receptive neuron. In an adult brain, plasticity may be accounted for by two mechanisms: the creation of new synaptic connections between neurons, and the modification of existing synapses. Axons, the transmission lines and dendrites which is the receptive zones, constitute two types of cell filaments that are distinguished on morphological grounds; an axon which has a smoother surface, fewer branches, and greater length, whereas a dendrite has an irregular surface and more branches. Neurons come in a wide variety of shapes and sizes in different parts of the brain. Fig. 1.2 illustrates the shape of a pyramidal cell, which is one of the most common types of cortical neurons. Like many other types of neurons, it receives most of the inputs through dendritic spines. The pyramidal cell can receive 10,000 or more synaptic contacts and it can project onto thousands of target cells.

The axon, which branches into collaterals, receives signals from the cell body and carries them away through the synapse (a microscopic gap) to the dendrites of neighboring neurons. A schematic illustration of the signal transfer between two neurons through the synapse is shown in
Fig. 1.3b. An impulse, in the form of an electric signal, travels within the dendrites and through the cell body towards the pre-synaptic membrane of the synapse.

Fig. 1.3 (a) Schematic of biological neuron. (b) Mechanism of signal transfer between two biological neuron
Upon arrival at the membrane, neurotransmitters (chemical like) are released from the vesicles in quantities proportional to the strength of the incoming signal. The neurotransmitters diffuse within the synaptic gap towards the post-synaptic membrane, and eventually into the dendrites of neighbouring neurons, thus forcing them (depending on the threshold of the receiving neuron) to generate a new electrical signal. The generated signal passes through the second neuron(s) in a manner identical to that just described.

The amount of signal that passes through a receiving neuron depends on the intensity of the signal emanating from each of the feeding neurons, their synaptic strengths, and the threshold of the receiving neuron. Because a neuron has a large number of dendrites /synapses, it can receive and transfer many signals simultaneously. These signals may either assist (excite) or inhibit the firing of the neuron depending on the type of neurotransmitters are released from the tip of the axons. This simplified mechanism of signal transfer constituted the fundamental step of early neurocomputing development (e.g., the binary threshold unit of McCulloh and Pitts, 1943) and the operation of the building unit of ANNs.

The crude analogy between artificial neuron and biological neuron is that the connections between nodes represent the axons and dendrites, the connection weights represent the synapses, and the threshold approximates the activity in the soma (Jain et. al., 1996). Fig.1.4 illustrates $n$ biological neurons with various signals of intensity $x$ and synaptic strength $w$ feeding into a neuron with a threshold of $b$, and the equivalent artificial neurons system. Both the biological network and
ANN learn by incrementally adjusting the magnitudes of the weights or synaptic strengths (Zupan and Gasteiger, 1993).

**Fig. 1.4 Signal interaction from n neurons and analogy to signal summing in an artificial neuron comprising the single layer perceptron**

### 1.3. Models of a Neuron

In 1958, Rosenblatt introduced the mechanics of the single artificial neuron and introduced the ‘Perceptron’ to solve problems in the area of character recognition (Hecht-Nielsen, 1990). Basic findings from the biological neuron operation enabled early researchers (e.g., McCulloh and Pitts, 1943) to model the operation of simple artificial neurons. An artificial processing neuron receives inputs as stimuli from the
NEURAL NETWORK BASED STUDIES ON SPECTROSCOPIC ANALYSIS AND IMAGE PROCESSING

environment, combines them in a special way to form a 'net' input, passes that over through a linear threshold gate, and transmits the (output, y) signal forward to another neuron or the environment, as shown in Fig. 1.4. Only when the net input exceeds (i.e., is stronger than) the neuron's threshold limit (also called bias, \( b \)), will the neuron fire (i.e., becomes activated). Commonly, linear neuron dynamics are assumed for calculating net input (Haykin, 2003). The net input is computed as the inner (dot) product of the input signals (\( x \)) impinging on the neuron and their strengths (\( w \)) (Basheer and Hajmeer, 2000).

In the context of computation, a neuron is pictured as an information-processing unit that is fundamental to the operation of a neural network. The block diagram sketched in Fig. 1.5 represents the model of a neuron, which forms the basis for designing artificial neural networks. There are three basic elements in the neuronal model:

![Fig. 1.5 Block diagram representing the Nonlinear model of a neuron](image)

1. Input signals
2. Synaptic weights
3. Activation function
4. Output signal

Fig. 1.5 Block diagram representing the Nonlinear model of a neuron
Input signal $\bar{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ weight factor $\bar{W} = \begin{pmatrix} w_{k1} \\ w_{k2} \\ \vdots \\ w_{kn} \end{pmatrix}$

Net output for the $k^{th}$ neuron is:

$$v_k = \bar{X}^T \bar{W} = x_1 w_{k1} + x_2 w_{k2} + x_3 w_{k3} + \cdots + x_n w_{kn} \quad (1.1)$$

- A set of synapses or connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{kj}$.

- An adder for summing the input signals, weighted by the respective synapses of the neuron; the operations described here constitutes a linear combiner.

- An activation function for limiting the amplitude of the neuron output. The activation is also referred to as a squashing function or limiting function in that it squashes (limits) the permissible amplitude range of the output signal to some finite value. Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval $[0,1]$ or alternatively $[-1,1]$ representing unipolar and bipolar cases respectively.

The neuron model of Fig.1.5 also includes an externally applied bias, denoted by $b_k$. The bias $b_k$ has the effect of increasing or lowering the net
input of the activation function, depending on whether it is positive or negative respectively.

In mathematical terms, we may describe a neuron $k$ by writing the following pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$  \hspace{1cm} (1.2)$$

and

$$y_k = \varphi(u_k + b_k)$$  \hspace{1cm} (1.3)$$

where $x_1, x_2, \ldots, x_m$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{km}$ are the synaptic weights of neuron $k$; $u_k$ is the linear combiner output due to the input signals; $b_k$ is the bias; $\varphi(.)$ is the activation function; and $y_k$ is the output signal of the neuron. The use of the bias $b_k$ has the effect of applying an affine transformation to the output $u_k$ of the linear combiner in the model of Fig. 1.5 as shown by

$$v_k = u_k + b_k$$  \hspace{1cm} (1.4)$$

In particular, depending on whether the bias $b_k$ is positive or negative, the relationship between the induced local field or activation potential $v_k$ of neuron $k$ and the linear combiner output $u_k$ is modified in the manner illustrated in Fig.1.6. The bias $b_k$ is an external parameter of artificial neuron $k$ and is an important parameter in describing the dynamics of the neuron.
Fig. 1.6 Affine transformation produced by the presence of a bias

Fig. 1.7 Another Nonlinear model of a neuron including the effect of bias accounted as a input signal fixed at +1.
Combinations of Eqs. (1.2) and (1.4) as follows:

\[ v_k = \sum_{j=0}^{m} w_{kj} x_j \]  

(1.5)

and

\[ y_k = \phi(v_k) \]  

(1.6)

In Eq. (1.4) a new synapse is added. Its input is

\[ x_{\mu} = +1 \]  

(1.7)

and its weight is

\[ w_{k\mu} = b_k \]  

(1.8)

Therefore the model of the neuron \( k \) is reformulated as in Fig. 1.7. In this figure, the effect of the bias is accounted as: adding a new input signal fixed at +1, and adding a new synaptic weight equal to the bias \( b_k \).

### 1.4 Types of activation functions

The activation function may be a linear or a nonlinear function. The activation function, denoted by \( \phi(v) \), defines the output of a neuron in terms of the induced local field \( v \). The activation function generates either unipolar or bipolar signals. In the following sections various types of function used for activating the neuron activities are described.

#### 1.4.1 A step function

It is a unipolar function and is also referred to as a threshold function. This function is shown in Fig. 1.8(a) and is defined as:
In engineering literature, this is a threshold function referred to as Heaviside function. Correspondingly, the output of neuron \( k \) employing such a threshold function is expressed as

\[
\phi(v) = \begin{cases} 
1 & \text{if } v \geq 0 \\
0 & \text{if } v < 0
\end{cases} \quad (1.9)
\]

where \( v_k \) is the induced local field of the neuron; so that

\[
v_k = \sum_{j=1}^{n} w_{kj} x_j + b_k \quad (1.11)
\]

Eqn.1.11 represents a neuron referred to in the literature as the McCulloch-Pitts model, in recognition of the pioneering work done by McCulloch and Pitts (1943). In this model, the output of the neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise. This statement describes the all-or-none property of the McCulloch-Pitts model.

### 1.4.2 Piecewise-Linear Function

This is also a unipolar function. The piecewise-linear function described in Fig.1.8(b) is defined as:
where the amplification factor inside the linear region of operation is assumed to be unity. The following two situations may be viewed as special form of the piecewise-linear function:

- A linear combiner arises if the linear region of operation is maintained without running into saturation.
- The piecewise-linear function reduces to a threshold function if the amplification factor of the linear region is made infinitely large.

### 1.4.3 Sigmoid Function

The sigmoid function, whose graph is S shaped, is also a unipolar function and is the most common form of activation function used in the construction of artificial neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behaviour. An example of the sigmoid function is the logistic function, defined by

\[
\varphi(v) = \frac{1}{1 + \exp(-av)}
\]  (1.13)
Fig. 1.8 Various types of activation functions (a) step function (b) piece-wise linear function (c) sigmoid function
where $a$ is the slope parameter of the sigmoid function.

By varying the parameter $a$, sigmoid functions of different slopes are obtained, as illustrated in Fig.1.8(c). In fact, the slope at the origin equals $a/4$. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. Whereas a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. Moreover the sigmoid function is differentiable, unlike in the case of other threshold functions. Differentiability is an important feature of neural network theory.

All the above mentioned activation functions are unipolar, which are varying between 0 and 1. It is sometimes desirable to have the activation function range from -1 to +1, in which case the activation function assumes an antisymmetric form with respect to the origin; that is, the activation function is an odd function of the induced local field. Specifically, the threshold function of Eq.(1.9) is now defined as

$$
\varphi(v) = \begin{cases} 
1, & \text{if } v > 0 \\
0, & \text{if } v = 0 \\
-1, & \text{if } v < 0
\end{cases}
$$

which is commonly referred to as the signum function. For the corresponding form of the sigmoid function, the hyperbolic tangent function is used, which is defined by:

$$
\varphi(v) = \tanh(v)
$$
1.5 Perceptrons

The perceptron (Fig. 1.7) can be trained on a set of examples using a special learning rule (Hecht-Nielsen, 1990). The perceptron weights (including the threshold) are changed in proportion to the difference (error) between the target (correct) output, \( Y \), and the perceptron solution, \( y \), for each example. The error is a function of all the weights and forms an irregular multidimensional complex hyperplane with many peaks, saddle points, and minima. Using a specialized search technique, the learning process strives to obtain the set of weights that corresponds to the global minimum. Rosenblatt (1962) derived the perceptron rule that will yield an optimal weight vector in a finite number of iterations, regardless of the initial values of the weights.

This rule, however, can perform accurately with any linearly separable classes (Hecht-Nielsen, 1990), in which a linear hyperplane can place one class of objects on one side of the plane and the other class on the other side. Fig. 1.9 (a) shows linearly and nonlinearly separable two-object classification problems. In order to cope with nonlinearly separable problems, additional layer(s) of neurons placed between the input layer containing input nodes) and the output neuron are needed leading to the multilayer perceptron (MLP) architecture (Hecht-Nielsen, 1990), as shown in Fig. 1.9 (b). Since these intermediate layers do not interact with the external environment, they are called hidden layers and their nodes called hidden nodes. The addition of intermediate layers revived the perceptron model by extending its ability to solve nonlinear classification problems. Using similar neuron dynamics, the hidden neurons process the
information received from the input nodes and pass them over to output layer.

Fig. 1.9. (a) Linear vs. nonlinear separability. (b) Multilayer perceptron showing input, hidden, and output layers and nodes with feedforward links.

The learning of MLP is not as direct as that of the simple perceptron. For example, the backpropagation network (Rumelhart et al., 1986) is one type of MLP trained by the delta learning rule (Zupan and Gasisteiger, 1993). However, the learning procedure is an extension of the simple perceptron algorithm so as to handle the weights connected to the hidden nodes (Hecht-Nielsen, 1990).
1.6 Learning Processes

Learning is a process by which the free parameters of a neural network are adapted through a process of simulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place. The above definition of learning process implies the following sequence of events:

1. The neural network is stimulated by an environment.
2. The neural network undergoes changes in its free parameters as a result of this simulation.
3. The neural network responds in a new way to the environment because of the changes that have occurred in its internal structure.

A prescribed set of well-defined rules for the solution of a learning problem is called learning algorithm. As one would expect, there is no unique learning algorithm for the design of neural networks. Rather, there is a kit of tools represented by a diverse variety of learning algorithms, each of which offers advantages of its own. Basically, learning algorithms differ from each other in the way in which the adjustment to a synaptic weight of a neuron is formulated. Another factor to be considered is the manner in which a neural network, made up of a set of interconnected neurons, relates to its environment.

Hebb’s postulate of learning is the oldest and the most famous of all learning rules; it is named in honour of the neuropsychologist Hebb (1949). His postulate states that:
When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Hebb proposed this change as a basis of associative learning which would result in an enduring modification in the activity pattern of a spatially distributed assembly of nerve cells.

The Hebb's postulate can be expanded and rephrased as a two-part rule:

- If two neurons on either side of a synapse are activated simultaneously, then the strength of that synapse is selectively increased.
- If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

Such a synapse is called a Hebbian synapse. More precisely, a Hebbian synapse is a synapse that uses a time-dependent, highly local and strongly interactive mechanism to increase synaptic efficiency as a function of the correlation between the presynaptic and postsynaptic activities.

1.7 A Brief History

In this section, in order to make the thesis self-contained, an overview of the historical evolution of ANNs and neurocomputing is briefly presented. Anderson and Rosenfeld (1988) provide a detailed history
along with a collection of the many major classic papers that affected ANNs evolution. Nelson and Illingworth (1990) divide 100 years of history into six notable phases: (1) Conception, 1890–1949; (2) Gestation and Birth, 1950s; (3) Early Infancy, late 1950s and the 1960s; (4) Stunted Growth, 1961–1981; (5) Late Infancy I, 1982–1985; and (6) Late Infancy II, 1986–present. The era of conception includes the first development in brain studies and the understanding of brain mathematics. It is believed that the year 1890 was the beginning of the neurocomputing age in which the first work on brain activity was published by William James (Nelson and Illingworth, 1990). Many (e.g., Hecht-Nielsen, 1990) believe that real neurocomputing started in 1943 after McCulloch and Pitts (1943) paper on the ability of simple neural networks to compute arithmetic and logical functions. This era ended with the book 'The Organization of Behavior' by Donald Hebb in which he presented his learning law for the biological neurons' synapses (Hebb, 1949). The work of Hebb is believed to have paved the road for the advent of neurocomputing (Hecht-Nielsen, 1990).

The gestation and birth era began following the advances in hardware/software technology which made computer simulations possible and easier. In this era, the first neurocomputer (the Snark) was built and tested by Minsky at Princeton University in 1951, but it experienced many limitations (Hecht-Nielsen, 1990). This era ended by the development of the Dartmouth Artificial Intelligence (AI) research project which laid the foundations for extensive neurocomputing research (Nelson and Illingworth, 1990).

The era of early infancy began with John von Neuman's work which was published a year after his death in a book entitled 'The Computer and
the Brain’ (von Neuman, 1958). In the same year, Frank Rosenblatt at Cornell University introduced the first successful neurocomputer (the Mark I perceptron), designed for character recognition which is considered nowadays the oldest ANN hardware (Nelson and Illingworth, 1990). Although the Rosenblatt perceptron was a linear system, it was efficient in solving many problems and led to what is known as the 1960s ANNs hype. In this era, Rosenblatt also published his book ‘Principles of Neurodynamics’ (Rosenblatt, 1962). The neurocomputing hype, however, did not last long due to a campaign led by Minsky and Pappert (1969) aimed at discrediting ANNs research to redirect funding back to AI. Minsky and Pappert published their book ‘Perceptrons’ in 1969 in which they over exaggerated the limitations of the Rosenblatt’s perceptron as being incapable of solving nonlinear classification problems, although such a limitation was already known (Hecht-Nielsen, 1990; Wythoff, 1993). Unfortunately, this campaign achieved its planned goal, and by the early 1970s many ANN researchers switched their attention back to AI, whereas a few ‘stubborn’ others continued their research. Hecht-Nielsen (1990) refers to this era as the ‘quiet years’ and the ‘quiet research’.

With the Rosenblatt perceptron and the other ANNs introduced by the ‘quiet researchers’, the field of neurocomputing gradually began to revive and the interest in neurocomputing renewed. Nelson and Illingworth (1990) list a few of the most important research studies that assisted the rebirth and revitalization of this field, notable of which is the introduction of the Hopfield networks (Hopfield, 1984), developed for retrieval of complete images from fragments. The year 1986 is regarded a cornerstone in the ANNs recent history as Rumelhart et al. (1986)
rediscovered the backpropagation learning algorithm after its initial development by Werbos (1974). The first physical sign of the revival of ANNs was the creation of the Annual IEEE International ANNs Conference in 1987, followed by the formation of the International Neural Network Society (INNS) and the publishing of the INNS Neural Network journal in 1988. It can be seen that the evolution of neurocomputing has witnessed many ups and downs, notable among which is the period of hibernation due to the perceptron’s inability to handle nonlinear classification. Since 1986, many ANN societies have been formed, special journals published, and annual international conferences organized. At present, the field of neurocomputing is blossoming almost daily on both the theory and practical application fronts.

1.8 Learning Rules

A learning rule is a procedure to modify the weight and biases of a network. This is also referred to as a training algorithm. The purpose of the learning rule is to train the network to perform some task. There are many types of neural network learning rules. They fall into three broad categories: supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, the learning rule is provided with a set of examples (the training set) of proper network behaviour:
[\{x_1, t_1\}, \{x_2, t_2\}, \{x_3, t_3\}, \ldots, \{x_q, t_q\}], \quad (1.16)

where \(x_q\) is an input to the network and \(t_q\) is the corresponding correct (target) output.

---

**Fig. 1.10** (a) Block diagram of learning with a teacher (b) Block Diagram of reinforcement learning
As the inputs are applied to the network, the network outputs are compared to the targets (Hagan et. al., 2002). The learning rule is then used to adjust the weight and biases of the network in order to move the network outputs closer to the targets. This kind of learning is also known as learning with a teacher. Fig.1.10 (a) shows a block diagram that illustrates this form of learning. Suppose now that the teacher and the neural network are both exposed to a training vector drawn from the environment. By the virtue of built-in-knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the optimum action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. The error signal is defined as the difference between the desired signal and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher; the emulation is presumed to be optimum in some statistical sense. In this way knowledge of the environment available to the teacher is transferred to the neural network through training as fully as possible. When this condition is reached, the teacher is dispensed with and let the neural network deal with the environment completely by itself.

In supervised learning, the process takes place under the tutelage of a teacher. But, in the paradigm known as learning without a teacher, there is no teacher to oversee the learning process. That is to say, there are no labelled examples of the function to be learned by the network. Under this
paradigm, two subdivisions are identified: one is the reinforcement learning and the other unsupervised learning.

In reinforcement learning, the learning of an input-output mapping is performed through continued interaction with the environment in order to minimize the scalar index of performance. Fig. 1.10 (b) shows the block diagram of one form of a reinforcement learning system built around a critic that converts a primary reinforcement signal received from the environment into a higher quality reinforcement signal called the heuristic reinforcement signal, both of which are scalar inputs. The system is designed to learn under delayed reinforcement, which means that the system observes a temporal sequence of state vectors also received from the environment, which eventually result in the generation of the heuristic reinforcement signal. The goal of learning is to minimize a cost-to-go function, defined as the expectation of the cumulative cost of actions taken over a sequence of steps instead of simply the immediate cost. It may turn out that certain actions taken earlier in that sequence of time steps are in fact the best determinants of overall system behaviour. The function of the learning machine, which constitutes the second component of the system, is to discover these actions and to feed them back to the environment.

In unsupervised learning or self-organized learning there is no external teacher or critic to oversee the learning process, as indicated in Fig. 1.11. Rather, provision is made for a task independent measure of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the
input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically. To perform unsupervised learning, a competitive learning rule is used. For that a neural network consisting of two layers— an input layer and a competitive layer are employed. The input layer receives the available data. The competitive layer consists of neurons that compete with each other (in accordance with a learning rule) for the opportunity to respond to features contained in the input data. In its simplest form, the network operates in accordance with a winner-takes-all strategy.

![Block diagram of unsupervised learning](image)

**Fig. 1.11 Block diagram of unsupervised learning**

### 1.9 Learning Algorithms

In the formative years of the neural network (1943-1958), several researchers stand out for their pioneering contributions:

- McCulloch and Pitts (1943) for introducing the idea of neural network as computing machines.
- Hebb (1949) for postulating the first rule for self-organised learning.
- Rosenblatt (1958) for proposing the perceptron as the first model for supervised learning.
In the present research work only supervised learning is used. Therefore in the following sections only supervised learning algorithms are dealt with. The perceptron is the simplest form of a neural network used for the classification of patterns which are said to be linearly separable. Basically, it consists of a single neuron with adjustable synaptic weights and bias. Indeed, Rosenblatt proved that if the patterns used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyper plane between the two classes (Rosenblatt, 1962). The proof of convergence of the algorithm is known as the perceptron convergence theorem. The perceptron built around a single neuron is limited to performing pattern classification with only two classes. In the following sections the Perceptron algorithm and the backpropagation algorithms are given.

1.9.1 The Perceptron Algorithm

The Perceptron, an invention of Rosenblatt (1962), was one of the earliest neural network models. A perceptron models a neuron by taking a weighted sum of the inputs and sending the output 1 if the sum is greater than some adjustable threshold value (otherwise it sends 0). Fig. 1.12 shows the device.

The inputs $(x_1, x_2, x_3, ...., x_n)$ and connection weights $(w_1, w_2, w_3, ...., w_n)$ in the figure are typically real values, both positive and negative. If the presence of some feature $x_i$ tends to cause the perceptron to fire, the weight $w_i$ will be positive; if the feature $x_i$ inhibits the perceptron, the weight $w_i$ will be negative. The perceptron itself consists
of the weights, the summation processor, and the adjustable threshold processor. Learning is a process of modifying the values of the weights and the thresholds (bias). It is convenient to implement the bias as just another weight $w_0$. This weight can be thought of as the propensity of the perceptron to fire irrespective of its inputs (Rich and Knight, 1994).

**Step 0**

Set up the neural network model as shown in Fig. 1.12

Initialize the weights and bias.

Set learning rate, $\eta$ ($0 < \eta < 1$)

Set minimum error value for stopping.

**Step 1**

While the stopping condition is false, do steps 2 - 6

**Step 2**

For each training pair $u:t$ do steps 3 - 5

**Step 3**

Set activations of input units: $x_i = u_i, i = 1, 2, 3, \ldots n$

**Step 4**

Compute the response of output unit:

$$s = b + \sum_{i} x_i w_i \quad (1.17)$$

**Step 5**

Update weights and bias

---

*Fig. 1.12 A Perceptron neuron model*
\[ w_i^{(\text{new})} = w_i^{(\text{old})} + \eta (t - s) x_i \]  
\[ b^{(\text{new})} = b^{(\text{old})} + \eta (t - s) \]  
\[ i = 1, 2, 3, \ldots, n \]  

Step 6  
Test for stopping condition:  
*If the largest weight change that occurred in step is smaller than a specified tolerance, then stop; else continue.*

### 1.9.2 The Backpropagation Algorithm

Fig. 1.13 shows a fully connected, layered, feedforward network. In this figure, weights on connections between the input and hidden layers are denoted by \( w_i \), while weights on connections between the hidden and output layers are denoted by \( w_h \). This network has three layers, although it is possible and sometimes useful to have more. Each unit in one layer is connected in the forward direction to every unit in the next layer. Activations flow from the input layer through the hidden layer and then on to the output layer. The knowledge of the network is encoded in the weights on connection between units. The existence of hidden units allows the network to develop complex feature detectors, or internal representations.

The units in a backpropagation network require a slightly different activation function from the perceptron. A backpropagation unit will sum its weighted inputs, but unlike the perceptron, it produces a real value between 0 and 1 as output based on a sigmoid function, which is continuous and differentiable, as required by the backpropagation
Like a perceptron, a backpropagation network typically starts out with a random set of weights (Rich and Knight, 1994).

**Fig. 1.13** A feed forward neural network. $w_i$ is the weight of the input layer to the hidden layer, $w_h$ is the weight of the hidden layer to the output layer.

**Given:** A set of input-output $(x:y)$ vector pairs.

**Compute:** A set of weights for a three layer network that maps inputs onto corresponding outputs. ($w_i$ is the weight of the input layer to the hidden layer, $w_h$ is the weight of the hidden layer to the output layer)

**Step 1** Let $A$ be the number of units in the input layer, $B$ be the number of units in the hidden layer, $C$ be the number of units in the output layer.

**Step 2** $w_{ij} = \text{random}(-0.1, 0.1)$ for all $i = 0, \ldots, A$, $j = 1, \ldots, B$
\( wh_{ij} = \text{random}(-0.1, 0.1) \) for all \( i = 0 \ldots B, j = 1 \ldots C \)

**Step 3** Initialize the activation of the threshold units. The values of the threshold units should never change. Set the learning rate, \( \eta \)

**Step 4** Choose an input-output pair. Assign activation levels to the input units.

**Step 5** Propagate the activations from the units in the input layer to the units in the hidden layer using the activation function

\[
h_j = \frac{1}{1 + e^{-\sum_{i} w_{ij} x_i}} \quad \text{for all } j = 1 \ldots B \quad (1.19)
\]

**Step 6** Propagate the activations from the units in the hidden layer to the units in the output layer

\[
o_j = \frac{1}{1 + e^{-\sum_{i} w_{ij} h_i}} \quad \text{for all } j = 1 \ldots C \quad (1.20)
\]

**Step 7** Compute the errors of the units in the output layer, \( \delta_{2j} \)

\[
\delta_{2j} = o_j (1 - o_j)(y_j - o_j) \quad \text{for all } j = 1 \ldots C \quad (1.21)
\]

**Step 8** Compute the errors of the units in the hidden layer, \( \delta_{1j} \)

\[
\delta_{1j} = h_j (1 - h_j) \sum_{i=1}^{C} \delta_{2i} w_{ji} \quad \text{for all } j = 1 \ldots B \quad (1.22)
\]

**Step 9** Adjust the weights \( w_i \) and \( w_h \)
The algorithm generalizes straightforwardly to networks of more than three layers. For each extra hidden layer, insert a forward propagation step between steps 6 and 7, an error computation step between steps 8 and 9, and a weight adjustment step between steps 10 and 11. Error computation for hidden units should use the equation in step 8, but with \( i \) ranging over the units in the next layer, not necessarily the output layer (Rich and Knight, 1994).

1.10 Digital Image Processing:

1.10.1 Image Representation, Sampling, Quantization

An image may be defined as a two-dimensional function, \( f(x,y) \), where \( x \) and \( y \) are spatial coordinates, and the amplitude of \( f \) at any pair of coordinates \( (x,y) \) is called the intensity or gray level of the image at that point. When \( x,y \) and the amplitude values of \( f \) are all finite, discrete quantities, it is called as a digital image. The field of digital image
processing refers to processing digital images by a digital computer. A digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, pels and pixels. In general, the fundamental steps in digital image processing consist of components like image acquisition, image enhancement, image restoration etc.

Image acquisition is the first process. Note that acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling. The types of images in which we are interested are generated by the combination of an illumination source and the reflection or absorption of energy from that source by the elements of the scene being imaged.

When an image is generated from a physical process, its values are proportional to energy radiated by a physical source. As a consequence, \( f(x,y) \) must be nonzero and finite; that is,

\[
0 < f(x,y) < \infty \tag{1.25}
\]

The function \( f(x,y) \) may be characterized by two components: (1) the amount of source illumination incident on the scene being viewed, and (2) the amount of illumination reflected by the objects in the scene. Appropriately, these are the illumination and reflectance components and are denoted by \( i(x,y) \) and \( r(x,y) \) respectively. The two functions combine as a product to form \( f(x,y) \):

\[
f(x,y) = i(x,y)r(x,y) \tag{1.26}
\]
where

\[ 0 < i(x, y) < \infty \]

(1.27)

and

\[ 0 < r(x, y) < 1 \]

(1.28)

Eq. (1.28) indicates that reflectance is bounded by 0 (total absorption) and 1 (total reflectance). The nature of \( i(x, y) \) is determined by the illumination source, and \( r(x, y) \) is determined by the characteristics of the imaged objects.

The intensity of a monochrome image at any coordinates \((x_n, y_n)\) determines the gray level \(l\) of the image at that point. That is,
From Eqs. (1.26) through (1.29), it is evident that $l$ lies in the range

$$L_{\text{min}} \leq l \leq L_{\text{max}}$$

In theory, the only requirement on $L_{\text{min}}$ is that it be positive, and on $L_{\text{max}}$ that it be finite. In practice, $L_{\text{min}} = i_{\text{min}} r_{\text{min}}$ and $L_{\text{max}} = i_{\text{max}} r_{\text{max}}$. The interval $[L_{\text{min}}, L_{\text{max}}]$ is called the gray scale. Common practice is to shift this interval numerically to the interval $[0, L-1]$, where $l = 0$ is considered black and $l = L-1$ is considered white on the gray scale. All the intermediate values are shades of gray varying from black to white.

The output of most sensors is a continuous voltage waveform whose amplitude and spatial behaviour are related to the physical phenomenon to be sensed. To create digital image, the continuous sensed data should be converted to digital form. This involves two processes: sampling and quantization. The basic idea behind sampling and quantization is illustrated in Fig. 1.15. Fig. 1.15 (a) shows a continuous image, $f(x,y)$, that is to be converted to digital form. An image may be continuous with respect to the $x$- and $y$- coordinates, and also in amplitude. To convert to digital form, the function has to be sampled in both coordinates and in amplitude. Digitizing the coordinate values is called sampling. Digitizing the amplitude values is called quantization.

The one-dimensional function shown in Fig. 1.15 (b) is a plot of amplitude values (gray level) of the continuous image along the line segment AB in Fig. 1.15 (a). The random variations are due to image...
Generating a digital image. (a) Continuous image (b) A scan line from A to B in the continuous image (c) sampling and quantization (d) digital scan line

noise. To sample this function, equally spaced samples along line AB as shown in Fig. 1.15(c) are taken. The location of each sample is given by a vertical tick mark in the bottom part of the figure. The samples are shown as small white squares superimposed on the function. The set of these discrete locations gives the sampled function. However, the values of the samples still span (vertically) a continuous range of gray-level values. In order to form a digital function, the gray level values also must be converted (quantized) into discrete quantities. The right side of Fig.1.15(c) shows the gray-level scale divided into eight discrete levels,
ranging from black to white. The vertical tick marks indicate the specific value assigned to each of the eight gray levels. The continuous gray levels quantized simply by assigning one of the eight discrete gray levels to each sample. The assignment is made depending on the vertical proximity of a sample to a vertical tick mark. The digital samples resulting from both sampling and quantization are shown in Fig.1.15 (d). Starting at the top of the image and carrying out this procedure line by line produces a two-dimensional digital image.

The result of sampling and quantization is a matrix of real numbers. Assume that an image \( f(x,y) \) is sampled so that the resulting digital image has \( M \) rows and \( N \) columns. The values of the coordinates \( (x,y) \) now become discrete quantities. The complete \( M \times N \) digital image can be written in a compact matrix form as:

\[
f(x,y) = \begin{bmatrix}
  f(0,0) & f(0,1) & \cdots & f(0,N-1) \\
  f(1,0) & f(1,1) & \cdots & f(1,N-1) \\
  \vdots & \vdots & \ddots & \vdots \\
  f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1)
\end{bmatrix}
\]

The right side of the equation is by definition is a digital image. Each element of this matrix array is called an image element, picture element, pixel or pel.

The digitization process requires decision about the values of \( M \), \( N \), and for the number, \( L \), of discrete gray levels allowed for each pixel. There are no requirements on \( M \) and \( N \) other than that they have to be positive integers. However, due to processing, storage, sampling and hardware considerations, the number of gray levels typically is an integer power of 2:
It is assumed that the discrete levels are equally spaced and that they are integers in the interval \([0, L-1]\). Sometimes the range of value spanned by the gray scale is called the dynamic range of the image, and the images whose gray levels span a significant portion of the gray scale are referred to as those having high dynamic range. When an appreciable number of pixels exhibit this property, the image will have high contrast. Conversely, an image with low dynamic range tends to have a dull washed out gray look.

The number, \(b\), of bits required to store a digitized image is

\[ b = M N k \]  

(1.33)

When \(M = N\), this equation becomes

\[ b = N^2 k \]  

(1.34)

Sampling is the principal factor determining the spatial resolution of an image. Basically, spatial resolution is the smallest discernible detail in an image. Gray-level resolution refers to the smallest discernible change in gray level (Gonzalez and Woods, 2002).

### 1.11 Various tools for Digital Image Processing

#### 1.11.1 The Two-Dimensional DFT and its Inverse

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement
techniques is to bring out detail that is obscured, or simply to highlight features of interest in an image. The main objective of enhancement is to process an image so that the result is more suitable than the original image for a specific application. Image enhancement techniques fall into two broad categories: spatial domain methods and frequency domain methods. The term spatial domain refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. Frequency domain processing techniques are based on modifying the Fourier transform of an image.

In the present research work, frequency domain spatial enhancement techniques are dealt with. Hence the focus is mostly on a discrete formulation of the Fourier transform. The discrete Fourier transform of a function (image) $f(x,y)$ of size $M \times N$ is given by the equation

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \quad (1.35)$$

This expression is computed for values of $u = 0, 1, 2, \ldots, M-1$, and also for $v = 0, 1, 2, \ldots, N-1$. The inverse Fourier transform is given by the expression:

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \quad (1.36)$$

For $x = 0, 1, 2 \ldots, M-1$ and $y = 0, 1, 2, \ldots, N-1$. Equations (1.35) and (1.36) comprise the two-dimensional, Discrete Fourier Transform pair.
The variables \( u \) and \( v \) are the transform or frequency variables, and \( x \) and \( y \) are the spatial or image variables. The location of \( 1/MN \) constant in Eqn.1.34 is not important. Sometimes it is located in front of the transform. Other times it is found split into two equal terms of \( 1/\sqrt{MN} \) multiplying the transform and its inverse.

The Fourier spectrum, phase angle, and power spectrum are defined as:

\[
|F(u, v)| = \left[ R^2(x, y) + I^2(x, y) \right]^{1/2} \quad (1.37)
\]

\[
\varphi (u, v) = \tan^{-1}\left[ \frac{I(u, v)}{R(u, v)} \right] \quad (1.38)
\]

and

\[
P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v) \quad (1.39)
\]

where \( R(u, v) \) and \( I(u, v) \) are the real and imaginary parts of \( F(u, v) \), respectively.

It is common practice to multiply the input image function by \((-1)^{x+y}\) prior to computing the Fourier transform. Due to the properties of exponentials it can be proved that:

\[
\sum \left[ f(x, y)(-1)^{x+y} \right] = F(u-M/2, v-N/2) \quad (1.40)
\]
where $\mathcal{F}[.]$ denotes the Fourier transform of the argument. This equation states that the origin of the Fourier transform of $f(x,y)(-1)^{x+y}$ is located at $u = M/2$ and $v = N/2$. In other words, multiplying $f(x,y)$ by $(-1)^{x+y}$ shifts the origin of $F(u,v)$ to frequency coordinates $(M/2, N/2)$, which is the centre of the $M \times N$ area occupied by the 2-D DFT. This area of the frequency domain is referred to as the frequency rectangle. It extends from $u = 0$ to $u = M-1$, and from $v = 0$ to $v = N-1$. In order to guarantee that these shifted coordinates are integers, usually $M$ and $N$ are taken to be even integers. When implementing the Fourier transform in a computer, the limit of summations are from $u = 1$ to $M$ and $v = 1$ to $N$. The actual centre of the transform will then be at $u = (M/2) + 1$ and $v = (N/2) + 1$. The value of the transform at $(u,v) = (0,0)$ is, from Eq.(1.35):

$$F(0,0) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)$$

(1.41)

which is the average of $f(x,y)$. In other words, if $f(x,y)$ is an image, the value of the Fourier transform at the origin is equal to the average gray level of the image. Because both frequencies are zero at the origin, $F(0,0)$ sometimes is called the dc component of the spectrum. If $f(x,y)$ is real, its Fourier transform is conjugate symmetric; that is,

$$F(u,v) = F^*(-u,-v)$$

(1.42)

where "*" indicates the standard conjugate operation on a complex number. From this, it follows that
which says that the spectrum of the Fourier transform is symmetric (Gonzalez and Woods, 2002).

1.12 Image Interpolation techniques

1.12.1 Nearest Neighbour Interpolation

Nearest-neighbour interpolation (also known as proximal interpolation or point sampling in some contexts) is a simple method of multivariate interpolation in 1 or more dimensions. Interpolation is the problem of approximating the value for a non-given point in some space, when given some values of points around that point. The nearest neighbour algorithm simply selects the value of the nearest point, and does not consider the values of other neighbouring points at all, yielding a piecewise-constant interpolant. The algorithm is very simple to implement, and is commonly used in real-time 3D rendering to select colour values for a textured surface.

1.12.2 Bilinear Interpolation

In mathematics, bilinear interpolation is an extension of linear interpolation for interpolating functions of two variables on a regular grid. The key idea is to perform linear interpolation first in one direction, and then again in the other direction.

Suppose that we want to find the value of the unknown function $f$ at the point $P = (x, y)$. It is assumed that we know the value of $f$ at the four
points \( Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1), \) and \( Q_{22} = (x_2, y_2) \). First do linear interpolation in the \( x \)-direction. This yields

\[
\begin{align*}
Q_{12} & \approx (x_2 - x_1) f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \\
Q_{22} & \approx \frac{x_2 - x_1}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).
\end{align*}
\]

where \( R_i = (x_i, y_i) \). \hspace{1cm} (1.44)

\[
\begin{align*}
f(R_1) & \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \\
f(R_2) & \approx \frac{x_2 - x_1}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).
\end{align*}
\]

where \( R_2 = (x, y_2) \). \hspace{1cm} (1.45)
We proceed by interpolating in the $y$-direction.

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

(1.46)

This gives us the desired estimate of $f(x, y)$.

$$f(x, y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)} (x - x_1)(y_2 - y) + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)} (x - x_1)(y_2 - y_1)$$

$$+ \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)} (x - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)} (x - x_1)(y - y_1)$$

(1.47)

If we choose a coordinate system in which the four points where $f$ is known are $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$, then the interpolation formula simplifies to

$$f(x, y) \approx f(0, 0)(1 - x)(1 - y) + f(1, 0)x(1 - y) + f(0, 1)(1 - x)y + f(1, 1)xy$$

(1.48)

Or equivalently, in matrix operations:

$$f(x, y) \approx [1 - x \quad x] \begin{bmatrix} f(0, 0) & f(0, 1) \\ f(1, 0) & f(1, 1) \end{bmatrix} \begin{bmatrix} 1 - y \\ y \end{bmatrix}$$

(1.49)

Contrary to what the name suggests, the interpolant is not linear. Instead, it is of the form

$$(a_1 x + a_2)(a_3 y + a_4)$$

(1.50)
so that it is a product of two linear functions. Alternatively, the interpolant can be written as

\[ b_1 + b_2 x + b_3 y + b_4 xy \]  

(1.51)

where

\[ b_1 = f(0, 0) \]
\[ b_2 = f(1, 0) - f(0, 0) \]
\[ b_3 = f(0, 1) - f(0, 0) \]
\[ b_4 = f(0, 0) - f(1, 0) - f(0, 1) + f(1, 1) \]  

(1.52)

In both cases, the number of constants (four) corresponds to the number of data points where \( f \) is given. The interpolant is linear along lines parallel to either the \( x \) or the \( y \) direction, equivalently if \( x \) or \( y \) is set constant. Along any other straight line, the interpolant is quadratic. The result of bilinear interpolation is independent of the order of interpolation. If we had first performed the linear interpolation in the \( y \)-direction and then in the \( x \)-direction, the resulting approximation would be the same.

In computer vision and image processing, bilinear interpolation is one of the basic resampling techniques. It is a texture mapping technique that produces a reasonably realistic image, also known as bilinear filtering or bilinear texture mapping. An algorithm is used to map a screen pixel location to a corresponding point on the texture map. A weighted average of the attributes (colour, alpha, etc.) of the four surrounding texels is
computed and applied to the screen pixel. This process is repeated for each pixel forming the object being textured.

When an image needs to be scaled-up, each pixel of the original image needs to be moved in certain direction based on scale constant. However, in scaling up an image, there are pixels (i.e. Hole) that are not assigned to appropriate pixel values. In this case, those holes should be assigned to appropriate image values so that the output image does not have non-value pixels.

Typically bilinear interpolation can be used where perfect image transformation, matching and imaging is impossible so that it can calculate and assign appropriate image values to pixels. Unlike other interpolation techniques such as nearest neighbour interpolation and bicubic interpolation (described below), bilinear interpolation uses the four nearest pixel values which are located in diagonal direction from that specific pixel in order to find the appropriate color intensity value of a desired pixel.

1.12.3 Bicubic Interpolation

In mathematics, bicubic interpolation is an extension of cubic interpolation for interpolation of data points on a two dimensional regular grid. The interpolated surface is smoother than corresponding surfaces obtained by bilinear or nearest neighbour interpolation. Bicubic interpolation can be accomplished using Lagrange polynomials, cubic splines or cubic convolution algorithm.

In image processing, bicubic interpolation is often chosen over bilinear interpolation or nearest neighbor in image resampling, when
speed is not an issue. Images resampled with bicubic interpolation are smoother and have fewer interpolation artifacts.

### 1.12.3a Bicubic spline interpolation

Suppose that the function values \( f \) and the derivatives \( f_x, f_y \) and \( f_{xy} \) are known at the four corners \((0,0), (1,0), (0,1), \) and \((1,1)\) of the unit square. The interpolated surface can then be written

\[
p(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j
\]  

(1.53)

The interpolation problem consists of determining the 16 coefficients \( a_{ij} \). Matching \( p(x,y) \) with the function values yields four equations,

1. \( f(0,0) = p(0,0) = a_{00} \)
2. \( f(1,0) = p(1,0) = a_{00} + a_{10} + a_{20} + a_{30} \)
3. \( f(0,1) = p(0,1) = a_{00} + a_{01} + a_{02} + a_{03} \)  
(1.54)
4. \( f(1,1) = p(1,1) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} \)

Likewise, eight equations for the derivatives in the \( x \)-direction and the \( y \)-direction

1. \( f_x(0,0) = p_x(0,0) = a_{10} \)
2. \( f_x(1,0) = p_x(1,0) = a_{10} + 2a_{20} + 3a_{30} \)
3. \( f_x(0,1) = p_x(0,1) = a_{10} + a_{11} + a_{12} + a_{13} \)
4. \( f_x(1,1) = p_x(1,1) = \sum_{i=1}^{3} \sum_{j=0}^{3} a_{ij}i \)  
(1.55)
5. \( f_x(0,0) = p_x(0,0) = a_{00} \)
6. \( f_x(1,0) = p_x(1,0) = a_{01} + a_{10} + a_{21} + a_{31} \)
7. \( f_x(0,1) = p_x(0,1) = a_{01} + 2a_{02} + 3a_{03} \)
8. \( f_y(l, 1) = p_y(l, 1) = \sum_{i=0}^{3} \sum_{j=1}^{3} a_{ij} j \)

And the following four equations represent the cross derivative

1. \( f_{xy}(0,0) = p_{xy}(0,0) = a_{11} \)
2. \( f_{xy}(1,0) = p_{xy}(1,0) = a_{11} + 2a_{21} + 3a_{31} \)
3. \( f_{xy}(0,1) = p_{xy}(0,1) = a_{11} + 2a_{12} + 3a_{13} \)
4. \( f_{xy}(l, 1) = p_{xy}(l, 1) = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} ij \)

The expressions above have used the following identities,

\[
p_x(x, y) = \sum_{i=1}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j
\]

\[
p_y(x, y) = \sum_{i=0}^{3} \sum_{j=1}^{3} a_{ij} x^i y^j
\]  \( \quad \) (1.57)

\[
p_{xy}(x, y) = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} x^{i-1} j y^j
\]

This procedure yields a surface \( p(x, y) \) on the unit square \([0, 1] \times [0, 1]\) which is continuous and with continuous derivatives. Bicubic interpolation on an arbitrarily sized regular grid can then be accomplished by patching together such bicubic surfaces, ensuring that the derivatives match on the boundaries.
If the derivatives are unknown, they are typically approximated from the function values at points neighbouring the corners of the unit square, i.e. using finite differences.

1.12.3b Bicubic convolution algorithm

Bicubic spline interpolation requires the solution of the linear system described above for each grid cell. An interpolator with similar properties can be obtained by applying convolution with the kernel in both dimensions:

\[
W(x) = \begin{cases} 
1 & \text{for } x = 0 \\
(a + 2)|x|^3 - (a + 3)|x|^2 + 1 & \text{for } 0 < |x| < 1 \\
a|x|^3 - 5a|x|^2 + 8a|x| - 4a & \text{for } 1 < |x| < 2 \\
0 & \text{otherwise}
\end{cases} \quad (1.58)
\]

where \(a\) is usually set to -0.5 or -0.75. Note that \(W(0) = 1\) and \(W(n) = 0\) for all nonzero integers \(n\).

This approach was proposed by Keys who showed that \(a = -0.5\) (which corresponds to cubic Hermite spline) produces the best approximation of the original function. If we use the matrix notation for the common case \(a = -0.5\), we can express the equation in a friendlier manner:

\[
p(t) = \frac{1}{2} \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ a_2 \end{bmatrix} \quad (1.59)
\]
for $t$ between 0 and 1 for one dimension. For two dimensions first applied once in $x$ and again in $y$:

$$b_{-1} = p(t_x, a_{(-1,-1)}, a_{(0,-1)}, a_{(1,-1)}, a_{(2,-1)})$$

$$b_0 = p(t_x, a_{(-1,0)}, a_{(0,0)}, a_{(1,0)}, a_{(2,0)})$$

$$b_1 = p(t_x, a_{(-1,1)}, a_{(0,1)}, a_{(1,1)}, a_{(2,1)})$$

$$b_2 = p(t_x, a_{(-1,2)}, a_{(0,2)}, a_{(1,2)}, a_{(2,2)})$$

(1.06)

The bicubic algorithm is frequently used for scaling images and video for display. It preserves fine detail better than the common bilinear algorithm.

1.12.4 Spline Interpolation

In the mathematical field of numerical analysis, spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called a spline. Spline interpolation is preferred over polynomial interpolation because the interpolation error can be made small even when using low degree polynomials for the spline. Using polynomial interpolation, the polynomial of degree $n$ which interpolates the data set is uniquely defined by the data points. The spline of degree $n$ which interpolates the same data set is not uniquely defined, and we have to fill in $n-1$ additional degrees of freedom to construct a unique spline interpolant.

Linear spline interpolation is the simplest form of spline interpolation and is equivalent to linear interpolation. The data points are graphically connected by straight lines. The resultant spline would be a
polygons if the end point is connected to the initial points. Algebraically, each $S_i$ is a linear function constructed as:

$$S_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} (x - x_i)$$

(1.61)

The spline must be continuous at each data point, that is

$$S_i(x_i) = S_i(x_i), \quad i = 1, \ldots, n-1$$

(1.62)

This is the case as we can easily see

$$S_{i-1}(x_i) = y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}} (x_i - x_{i-1}) = y_i$$

(1.63)

$$S_i(x_i) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} (x_{i+1} - x_i) = y_{i+1}$$

(1.64)

Commonly, magnification is accomplished through convolution of the image samples with a single kernel—typically the bilinear, bicubic (Netravali, 1995) or cubic B-spline kernel (Unser M et al., 1991). The mitigation of aliasing by this type of linear filtering is very limited. Magnification techniques based on a priori assumed knowledge are the subject of current research. Directional methods (Bayrakeri and Mersereau, 1995 and Jensen and Anastassiou, 1995) examine an image’s local edge content and interpolate in the low frequency direction (along the edge) rather than in the high-frequency direction (across the edge).
Multiple kernel methods typically select between a few ad hoc interpolation kernels (Darwish and Bedair, 1996). Orthogonal transform methods focus on the use of the discrete cosine transform (DCT) (Martucci, 1995 and Shinbori and Takagi, 1994) and the wavelet transform (Chang et. al., 1995). Variational methods formulate the interpolation problem as the constrained minimization of a function (Karayiannis and Venetsanopoulos, 1991 and Schultz and Stevenson, 1994). It should be noted that these techniques make explicit assumptions regarding the character of the analog image.

With the rapid increase in available computing power, coupled with great strides in image feature analysis, model-based, often highly nonlinear interpolative techniques have become a viable alternative to classic linear methods and have received increasing attention recently. Several examples of model-based approaches to spatial image interpolation can be found in Jensen and Anastassiou (1995), Jensen and Anastassiou (1990), Martinez and Lim (1989), Wang and Mitra (1991). Each of these papers utilizes the concept of an edge in a different fashion to enhance interpolation results. Artificial neural network based interpolation of image processing is still in its infancy stage (Davila and Hunt, 2000). Hence it was thought worthwhile to pursue this technique for image processing applications.

Summary

An introduction to neural networks and digital image processing is given in this chapter. Neural network can be viewed as a computational tool for solving complex real world problems. The advantage of using
neural network is that most of the computational complexities are encountered in the training phase itself. When implemented for a real world problem, the output obtained for the given inputs is only a mapping between the input and the output. The various tools for image processing is also introduced. The various available interpolation techniques are also discussed.

References


