Chapter 1

INTRODUCTION

1.1 An Analogy

Artificial Neural Networks (ANN) have been established as an alternative paradigm of computation vis-à-vis the concept of programmed computation [11, 21, 64, 75, 80, 87, 88, 94, 155, 190, 220, 235, 236, 280, 293] in which (usually procedural) algorithms are designed and sequentially implemented. (See [11, 21] for comparison of ANN's and modern digital machines).

ANNs have been inspired and modeled on the biological neural network (the brain), but the existing neural models have almost no similarities to the biological neural systems.¹ Even the simplest biological brain has the following contra - distinctive features as compared to the von - Neumann architecture of the digital computer [21]:

i) The computation is massively distributed and parallel.

ii) Learning replaces a priori program development.

¹To quote Widrow and Lehr [280] - “There is some reason to consider attacking certain problems by designing naturally parallel computers, which process information and learn by principles borrowed from the nervous system of biological creatures. This does not necessarily mean we should attempt to copy the brain part by part. Although the bird served to inspire the development of the airplane, birds do not have propellers, and airplanes do not operate by flapping feathered wings.”
iii) Fault-tolerant to a high degree, that is, death of a few cells during the computation process does not affect the end result of the computation.

Based on this analogy the desired property set of an artificial neural network can be enumerated as [11, 15, 21, 41, 220]:

i) Integration and storage of experience, which could be previous classification or association of learning data. Or, self-organization of experience.

ii) Consideration of new experience in the context of stored experience. Or, a context-addressable structure.

iii) Ability to make accurate predictions about new situations on the basis of previously (self-) organized experiences. Or, generalization capability.

iv) A fault-tolerant architecture, in the sense that loss of a few neurons may be recoverable by adaptation of those remaining (perhaps with additional training) and/or by growth of new neurons or associations between neurons.

To possess these characteristics the ANNs implement a procedure for doing predictive learning (from samples) or statistical estimation, or, ANNs behave as a learning system. It amounts to estimating properties of some (unknown) statistical distribution from known samples or training data. Information contained in the training data (past experiences) can be used to answer questions about future samples [41, 267, 268, 269]. Two stages in the operation of a learning system can be distinguished:

(a) Learning/estimation/training to estimate the unknown dependency (from training samples), and

(b) Operation/prediction, when predictions are made for future or test samples.
LEARNING IN SIGMOIDAL FEEDFORWARD ARTIFICIAL NEURAL NETWORKS

This description assumes that both the training and test data are from the same underlying statistical distribution. In other words, this (unknown) distribution is fixed.

For any learning system, the following questions must be answered [166]:

Q1. What is the structure of the system. (System Architecture).
Q2. How does the system learn. (Training Paradigm/Algorithm).
Q3. What are the tasks solved by the system. (Learning Tasks/Problems).
Q4. How “good” is the learning system in adverse conditions. (Robustness / Generalization / Fault-Tolerance).

1.2 ANN Architecture

ANNs are built by connecting elementary processing units, called nodes or neurons, in a variety of architectures. The process of producing an output from inputs in a generic processing unit is illustrated in Fig. 1.1. A $n$ - dimensional input vector $x = \{x_1, x_2, \ldots, x_n\}$ is presented to the node corresponding to which the neuron produces an output. Each of the inputs ($x_i$'s), is associated with a carrier / connector which takes a portion of the input to the node for processing. How much of the input is transferred to the node, is measured by the strength of this connection / coupling of the input to the node. This connection strength is denoted by $w_i$. The node combines the inputs and produces an output that is a function of the inputs. This output function is known by many names – Activation Function, Squashing Function, Transfer Function or Unit Function. That is, the unit transfer function is in general a function of the inputs and the associated weights, or $f(x, w)$. 
The nodes can be described as belonging to either of the two classes, namely:

(a) **Deterministic models**: which implement the (non)-linear mapping \( \mathbb{R}^d \rightarrow \mathbb{R} \)

where \( \mathbb{R} \) is the one-dimensional real space and \( \mathbb{R}^d \) is the cartesian product of \( d \) one-dimensional real spaces, and

(b) **Stochastic models**: which implement a probabilistic mapping from the input space to the output space of the neuron.

### 1.2.1 Deterministic Unit Models

In this subsection a two part node structure (as they happen to be the most popular kind of nodes) is defined and the rest (all other forms) are grouped together as other structures.

**Two Part Node Structure**

In the two part node structure (Fig. 1.2), the inputs are first combined. Then this net input is transformed by a function, called the activation / transfer / squashing function, into the output. In addition to the actual inputs to the node,
Learning in Sigmoidal Feedforward Artificial Neural Networks

![Neural Network Diagram](image)

Figure 1.2: Two Part Processing Model of a Node.

A bias (also called the threshold term) \( \theta \), is applied to the node, which has the effect of lowering or increasing the net input to the node, depending on its sign. The bias can be modeled as a constant input \( x_0 = 1 \), and an associated weight \( w_0 = \theta \).

The inputs may be combined in more than one way. Common examples are:

Additive: \[
v = \sum_{i=1}^{n} x_i + \theta = \sum_{i=0}^{n} x_i \tag{1.1}\]
in this case \( x_0 = \theta \).

Weighted Additive: \[
v = \sum_{i=1}^{n} w_i x_i + \theta = \sum_{i=0}^{n} w_i x_i \tag{1.2}\]
where \( x_0 = 1 \) and \( w_0 = \theta \).

Multiplicative: \[
v = \prod_{i} x_i \tag{1.3}\]
Figure 1.3: Node Structure for Mean Variance Connections.

In this either the bias is not used or is used as a constant ($\theta = x_0 = \text{constant}$).

$$Mean - Variance : \quad v = \sqrt{\sum_i \left( \frac{\mu_i - x_i}{\sigma_i} \right)}$$  \hspace{1cm} (1.4)

In this case the node has two weights associated with each of the inputs, as shown in Fig. 1.3. One of these dual connections is to allow one element of the pair to represent the mean of the class ($\mu_i$) and the other to represent the variance of the class ($\sigma_i$) [148, 207, 236].

Another two part node structure is defined as the Min-Max connections. In this the dual connections is assigned one of the pair (of the weights), say $u_i$, to become the minimum bound for the class and the other, say $w_i$, to become the maximum of the class [234].

The activation function maps the (possibly infinite) input domain to a (in general, pre-specified and/or finite) range. The inputs can be digital (discrete) or analog (real), in some cases the inputs may be piece-wise continuous. In some instances, some of the components of the input vector may be discrete while some
Figure 1.4: Unit Transfer Function Shapes.

may be continuous in nature. Although the number of the activation function is infinite, functions commonly used are [58, 21, 79, 83, 87, 220, 230]:

A. **Threshold (Step) Function**: One of the common unit functions is the step function (Fig. 1.4a, equation 1.5). If the combination function is a simple sum of all its inputs (equation 1.1) including the bias, then for boolean inputs, this defines the McCulloch and Pitts neuron [163] which was the first artificial neuron postulated. The output of this is given by:

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

(1.5)
where \( v \) is given by equation 1.1. A generalization of the McCulloch and Pitts model is the incorporation of weights in the connection or \( v \) is given by equation 1.2. Sometimes instead of the Heaviside’s step function (equation 1.5) the signum function (Fig. 1.4b with \( \alpha = 1 \) and \( \beta = -1 \), equation 1.6) is used (where \( v \) is the net input to the node):

\[
 f (v) = \begin{cases} 
 1 & \text{if } v \geq 0 \\
 -1 & \text{if } v < 0 
\end{cases} \tag{1.6}
\]

The general class of these functions may be defined (Fig. 1.4b, equation 1.7), for positive \( \alpha \) and \( \beta \), as:

\[
 f (v) = \begin{cases} 
 \alpha & \text{if } v \geq 0 \\
 -\beta & \text{if } v < 0 
\end{cases} \tag{1.7}
\]

The equations 1.5, 1.6, and 1.7 describe two state nodes where the input domain is infinite. The input domain is sometimes limited to a specific (in, general symmetric) domain and equation 1.7 is modified as (for \( Max > 0 \)):

\[
 f (v) = \begin{cases} 
 \alpha & \text{if } 0 \leq v \leq Max \\
 -\beta & \text{if } -Max \leq v < 0 \\
 0 & \text{otherwise} 
\end{cases} \tag{1.8}
\]

This can be further generalized to a k-state node (Fig. 1.4c) as:

\[
 f (v) = \alpha_i \quad \text{if } v \in \text{Domain}(v) \tag{1.9}
\]
where
\[ i = 1, 2, \ldots, k \] \hspace{1cm} (a)
\[ \alpha_i < \alpha_{i+1} \] \hspace{1cm} (b)
\[ l_0 < l_1 < \ldots < l_k \] \hspace{1cm} (c)
\[ l_0 = -\infty \text{ or } \text{Minimum} \] \hspace{1cm} (d)
\[ l_k = \infty \text{ or } \text{Maximum} \] \hspace{1cm} (e)
\[ \text{Domain}(v) = (-\infty, \infty) \text{ or } [\text{Minimum, Maximum}] \]
\[ = [l_0, l_1) \cup \ldots \cup [l_{k-1}, l_k] \] \hspace{1cm} (f)

where usually $|\text{Minimum}| = |\text{Maximum}|$, for symmetry reasons and generally, the intervals $[l_i, l_{i+1}]$ are equal sized (except the leftmost and/or the rightmost interval for the case when the input domain is infinite in one or both the direction of the real line).

B. **Linear Function:** produces a linearly modulated output (Fig. 1.4d, equation 1.11) from the inputs. Typically additive mixing (equation 1.1 and 1.2) of inputs is done to produce the net input, $v$.

\[ f(v) = \alpha v \] \hspace{1cm} (1.11)

where $\alpha$ is a positive constant (generally taken between 0 and 1). In case $\alpha = 1$, this effectively removes the activation function, $f$ from the Fig. 1.2.

C. **Piecewise Linear Function:** The piecewise linear function, described in Fig 1.4e, is defined as:

\[ f(v) = \begin{cases} 
\alpha & \text{if } v \geq \epsilon \\
\eta v & \text{if } \delta < v < \epsilon \\
-\beta & \text{if } v \leq \delta
\end{cases} \] \hspace{1cm} (1.12)
where $\alpha$, $\eta$ and $\beta$ are positive constants. $\alpha$ and $-\beta$ are the saturation values of the function, and at the points $v = \epsilon$ and $x = \delta$ the function has discontinuities. This form of unit function can be viewed as an approximation to a non-linear (amplifier) function. The linear activation function results in the limit $\epsilon \rightarrow \infty$ and $\delta \rightarrow -\infty$, while the threshold unit function arises for $\eta \rightarrow \infty$.

**Sigmoid Function:** The sigmoid unit function (Fig. 1.4f) is a continuous version of the piecewise linear function. The sigmoid function is a bounded, monotonic, non-decreasing function that provides a graded response within a (pre-specified) range.

The most common sigmoid function is the log - sigmoid or the logistic function (equation 1.13), where $\beta > 0$ (usually taken as $\beta = 1$ or $\beta = 1/2$), which provides an output from 0 to $A$ (generally $A = 1$ is used) for the input interval $v \in (-\infty, \infty)$.

$$f(v) = \frac{A}{1 + e^{-\beta v}} \quad (1.13)$$

When $\beta \rightarrow \infty$ in equation 1.13, the slope of the function becomes infinite and in effect, the function becomes the threshold function (equation 1.5). Two commonly used alternatives to the log - sigmoid function is the tangent hyperbolic function (equation 1.14) which ranges between -1 and 1 ($\beta > 0$, generally the value of $\beta = 1$ is used).

$$f(v) = \tanh(\beta v) = \frac{e^{\beta v} - e^{-\beta v}}{e^{\beta v} + e^{-\beta v}} \quad (1.14)$$

and the augmented ratio of squares ($\beta > 0$, generally $\beta = 1$ is used) (equation
1.15): \[
f(v) = \begin{cases} 
\frac{(\beta v)^2}{1 + (\beta v)^2} & \text{if } v \geq 0 \\
0 & \text{if } v < 0 
\end{cases} 
\] (1.15)

E. Gaussian Function: The Gaussian function (Fig. 1.4g) is a radial function (symmetric about the origin) whose form is given by equation 1.16 (Fig. 1.4g).

\[f(v) = e^{-\alpha v^2}\] (1.16)

where $\alpha > 0$ (usually the value of $\alpha = 1$ or $\alpha = 1/2$ are used). In the specific case of the Mean-Variance node (eq. 1.4), the value used is $\alpha = 1/2$.

F. Min - Max Functions: The concept of using combinations of minimum (min) and maximum (max) functions generates families of other unit functions, for example:

\[f(v) = \min\{1, e^{\alpha v}\}\] (1.17)

\[f(v) = \max\{0, 1 - e^{\alpha v}\}\] (1.18)

Other type of unit functions that can be put in the two-part model of a node:

[G.1] Any rational function which is not a polynomial. [51, 52]

[G.2] Any root $v^\alpha$, provided $\alpha$ is not a natural number. [51, 52]

[G.3] The logarithm (for any base $b > 1$). [51, 52]

[G.4] Exponential function $e^{\alpha v}$ [51, 52]

[G.5] The radial basis functions $(1 + x^2)^\alpha$, $\alpha < 1, \alpha \neq 0$. [51, 52]
Sine, Cosine and other trigonometric functions. [51, 52]

Any other well behaved (bounded) function of single variable. [51, 52]

Triangular activation functions. [51, 52, 157]

Spiking Neurons. [51, 157]

Though the class of potential activation functions is infinite, the above enumerated activations are the usually used activations. In this thesis, the emphasis is on the sigmoidal activations. The commonly used sigmoidal activations appear as distinct points in the sigmoidal activation space. This hampers the comparison of the suitability of the activation function for use in ANNs [58]. Here, one of the central themes is the development of one parameter sigmoidal activation classes which would allow an investigation of the efficiency / suitability of the activation(s) in ANNs.

Exceptions to the Two Part Model

Many unit models do-not allow visualization in two parts. An example of such a unit model is the polynomial unit in which the output is formed by computing some polynomial of the inputs (as equation 1.19, or some variant thereof):

$$f(v) = \sum_{i} \alpha_i x_i^\beta_i + \gamma$$  \hspace{1cm} (1.19)

where $\alpha_i$'s $\in \mathbb{R}$, and $\beta_i$'s $\in \mathbb{R}$ or $\mathbb{R}^+$. A multiplicative unit where the output is given by equation 1.20 is another case where the two part model of the (linear) combination followed by (non) linear mapping cannot be used.

$$f(v) = \prod_{i} x_i^{\beta_i}$$ \hspace{1cm} (1.20)

where $\beta_i$'s are positive constants, generally taken as 1.
But $\beta$'s can be taken as adaptive parameter and can then be adjusted during the training of the ANN. In this formalism then the $\beta$'s becomes the weight of connection. A final example is the radial basis unit [58, 72, 87].

1.2.2 Stochastic Model of a Neuron

For some applications of neural networks it is desirable to base the analysis on a stochastic neural model [5, 46, 47, 153, 254, 274]. This node structure has been inspired by thermodynamics of statistical systems [136, 255]. In this approach the simplest model occurs when the activation of the McCulloch-Pitts model is given a probabilistic interpretation. Specifically, a node is permitted to stay in only one of the two states: say +1 or -1. The decision for a node to switch its state is probabilistic. Let $z$ denote the state of the neuron, and $P(v)$ denote the probability of state switch, where $v$ is the net input to the node.

Then the probability of a state change is given by the relation:

$$z = \begin{cases} 
1 & \text{with probability } P(v) \\
-1 & \text{with probability } P(v)
\end{cases} \quad (1.21)$$

A standard choice for $P(v)$ is the sigmoid shaped function(s) including the logistic and the trigonometric tangent inverse function [47, 46, 94, 156].

$$P(v) = \frac{1.0}{1 + e^{-\beta v}} \quad (1.22)$$

$$P(v) = \frac{1.0}{\pi} \tan^{-1} \left( \frac{\pi \beta v}{2} \right) + \frac{1}{2} \quad (1.23)$$
Figure 1.5: Types of Feedforward ANN's.
1.2.3 Ann Taxonomy

ANNs can be classified broadly into feedforward and recurrent networks. Feedforward networks (Fig. 1.5) are those architectures in which the computation flows from the input node(s) to the output node(s) in a sequence. The nodes are arranged in a sequential layers of nodes which have no lateral connection (the nodes of a layer do not exchange/share information for computation). The nodes of one layer are connected to all or a subset of the nodes of the previous layer nodes as well as the subsequent layer nodes; this can be only the immediate previous or immediate next, or it can be all of the previous and subsequent layers. The input layer is also known as the 0th (zeroth) layer and has a linear output (is used to transfer input values to the subsequent computational layers). The last layer is known as the output layer. The activation function of the output layer is either a linear function or any other (non)-linear function described in section 1.1.1. The layers in-between the input and the output are known as hidden layers. A n-layer ANN actually has n + 1 layer including the 0th layer. The numbering/naming nomenclature is illustrated in Fig. 1.5, where by convention computation flows from left to the right (i.e., there is no feedback loop) The feedforward networks are typically in regresional tasks like function approximation and multivariate/bivariate regression as well as classification tasks. Feedforward networks are more often than not found within the supervised learning paradigm. Recurrent networks (Fig 1.6) are those network in which closed loops exist in the network (graph) topology. These networks are used primarily for auto-associative (content) addressable memory [3, 7, 102, 134, 127], optimization and constraint satisfaction problems [44, 45, 81], principal component analysis [9, 56, 82, 186, 187], temporal signal processing [83, 282] etc. The recurrent network can arise from symmetric/bidirectional connections being present (for example, Hopfield networks [94, 102]), lateral connections in feedforward networks (competitive networks) [186, 187], feedback loops in a feedforward netwo-
rks from higher layers to lower layers (Elman and Jordan Networks which are used for temporal analysis) [59, 94, 87]. For a more detailed architectural taxonomy see Lippman[155] and Patterson[190]. The network architecture of the system (including node type(s)) is decided in advance before network training or the architecture is decided by various network construction techniques, which then are a part of the learning process [63, 99, 147, 167, 201]. Next the learning paradigms and learning rules on which the training algorithms (learning processes) are based, are described.

1.3 Learning Paradigm and Training Algorithms

The presence or absence of a guide and the nature of the guidance provided during the ANN learning mechanism / training can characterize the learning paradigms.

There are, primarily speaking, two types of learning paradigm, the supervised learning paradigm and the unsupervised learning paradigm. These represent the extreme points of the learning paradigm spectrum which includes the re-inforcement learning paradigm in which there is a critic which tells whether the system is responding correctly or wrongly to a given stimuli. In re-inforcement learning indirect inputs are provided to the learning system which tells it whether it is learning in the right directions or not [175, 283, 284]. Inbetween these two (the supervised and the unsupervised learning paradigms) lie the various techniques of hybrid learning which incorporate both supervised and unsupervised learning for some facet of the learning algorithm [90, 91, 210].

1.3.1 Learning With a Teacher or Supervised Learning

In this paradigm, both the inputs and the desired outputs are known. The network is exposed to the input and the ANN output is compared against the desired output to calculate an error measure. The actual output is known (for a given
(a) Architectural graph of Hopfield Network.

(b) Architectural graph of a simple competitive learning network. Lateral connections are inhibitory in nature.

(c) A representative Jordan’s Network.

(d) A representative Elman’s Network.

Figure 1.6: Some Recurrent ANN Architectures.
input) as an output from the "Actual System", which acts as the teacher [68]. The learning methodologies, therefore, are geared towards the minimization of the error measure by adjustment of the network parameters. Popular error measures are the Mean Squared Error measure (MSE, L₂ norm) and other Lₚ norm (where p = 1, 2, ..., n or ∞) based error measures, and weighted L₂ norm and other non-Euclidean norms (equations 1.24) (for the kth exemplar).

\[
E_k = \begin{cases} 
\sum_i \| y_i^{(k)} - y_i^{(n)} \| & (a) \\
\max \left\{ y_i^{(k)} - y_i^{(n)} \right\} & (b) \\
\sum_i \left( y_i^{(k)} \ln \left( \frac{y_i^{(k)}}{y_i^{(n)}} \right) + (1 - y_i^{(k)}) \ln \left( \frac{1 - y_i^{(k)}}{1 - y_i^{(n)}} \right) \right) & (c) \\
- \sum_i \left( y_i^{(k)} \ln \left( \frac{y_i^{(k)}}{y_i^{(n)}} \right) - (1 - y_i^{(k)}) \ln \left( 1 - y_i^{(n)} \right) \right) & (d)
\end{cases}
\]

In equation 28(c) the y's are restricted to [0, 1] and the assumption that 0 ln(0) = 0 is made. This error function was used by Matsuoka and Yi [162], and was shown to have lesser number of local minima as compared to MSE, similarly in equation 28(d) y's are restricted to (0, 1)(264). The two forms of the cross-entropy functions are used as loss/error functions, generally for classification tasks [75].

1.3.2 Unsupervised Learning

In this learning paradigm, the network is not provided any stimuli from the environment for learning. Rather, it uses local information present in the training exemplars to learn. It organizes presented data and discovers its emergent collective properties. This is done through 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. In this system there is no teacher, therefore this learning is also called self-learning.
The learning paradigms are implemented by the training algorithms, and lead to changes in the free parameters (the set of weights and biases and other parameter, together called the weights of the ANN) of the ANN to solve the task for which the network is being trained. Thus, ANNs may be thought of as structures that embed a class of (weight) parameterized functions (or, are mathematically equivalent to a set of functions which is parameterized by the weights of the network). Thus, the training of ANNs may be thought of as the mechanism of identifying the appropriate function from this class of functions. Typically, the training is done by starting from a random weight configuration and searching for the appropriate weight configuration. The search rule are also known as weight updation rule or training rules or learning rule and can be broadly classified as:

1.3.3 Error Correcting Learning Rule

Let the there be $N$ exemplars on which the $i^{th}$ exemplar presented to the input of the ANN be $x^{(i)}$ (the $j^{th}$ input component of this exemplar is represented by $x_j^{(i)}$) corresponding to which the output vector is $y^{(i)}$ ($y_j^{(i)}$). To this input $x^{(i)}$, let the ANN produce an output $y^{(i)}$ then the error for this input ($x^{(i)}$), for each of the output component, is defined as:

$$e_j^{(i)} = y_j^{(i)} - y'_j^{(i)}$$ (1.25)

This error together with the desired and the obtained response is used by the learning system to adjust its free parameters (connection strengths) to minimize the $e_j^{(i)}$, for all $i$ and $j$, (generally in an iterative fashion). This is done by minimizing an appropriate measure of the component errors:

$$\mathcal{E} = f \left( e_j^{(i)}(n) \right)$$ (1.26)
where \( n \) denotes the component error in the \( n^{th} \) iteration of the learning cycle. The commonly used error measures are given in equation 1.24, the net error is the sum of the componentwise error for all the exemplars presented in an epoch of training for batch training or is the sum of the componentwise error for a single exemplar presentation to the ANN, given in equation 1.24 for on-line training.

The error measurement functional can be a metric (similar to equation 24a and 24b) or can be any other function with the following set of properties:

1. \( \forall i, j; e_{j}^{(i)} \to 0 \Rightarrow E \to 0 \)
2. \( E \) is monotonic increasing function in \( e_{j}^{(i)} \) (usually the requirement is for \( E \) to be symmetric, or a monotonic increasing function in \( |e_{j}^{(i)}| \)).
3. \( E \) as well as \( f \) (eq. 1.26) is continuous in \( e_{j}^{(i)} \).
4. \( E \) is differentiable in \( e_{j}^{(i)} \)’s and the parameters of the networks (this may have to be relaxed in the case of discrete networks where network parameters are constrained to be discrete).
5. An auxiliary condition that may be applied on the error functional is the requirement for the the error functional to be finite in the limit \( e_{j}^{(i)} \to \infty \), or, \( e \to \infty \Rightarrow E \to \text{Constant} \neq 0, \text{Constant} \in (0, \infty)^{2} \).
6. Additional requirements may also be imposed, like the derivative \( (df/d\lambda, \) where \( \lambda \) is any network parameter) of the error functional should be finite everywhere and tend to zero in the limit \( e_{j}^{(i)} \to 0 \) and \( e_{j}^{(i)} \to \infty \).

Examples of learning algorithms that implement the error correcting rule are the Rosenblatt’s Perceptron Learning Rule [209], the Widrow-Hoff’s Least Mean Squared (LMS) algorithm, the generally used error measure, the squared error function (corresponding to the L2 norm), is unbounded or \( e_{j} \to \infty \Rightarrow E \to \infty \), and is sensitive to outliers. This condition is similar to the condition on the error measure(s) used (sometimes) in robust regression.

1.3.4 Correlation Based Learning or Constructive Learning Rule

This learning rule is derived from the observation by Hebb((89, p. 62)) that when the nodes of the two ends of a neural connection fire in tandem then connection is strengthened else its weakened. This implies that the behaviour of the nodes at the two ends of the connection are correlated and this correlation is “measured” by the strength of the connection.

Let \( x_j \) be the \( j^{th} \) input to the \( k^{th} \) node in the network and its associated weight be \( w_{kj} \), and let the output from the node be \( y'_k \). It is assumed that the adjustment to the weights is done at discrete intervals of time and is indexed by \( n \), then the weight correction in this formulation of the learning rule would be, in general:

\[
\Delta w_{kj}(n) = F(y'_k(n), x_j(n))
\]  

(1.27)

The eq. 1.27 admits of many forms, all which qualify as Hebbian. In the simplest form, it is described by:

\[
\Delta w_{kj}(n) = \eta y'_k(n) x_j(n)
\]  

(1.28)

where \( \eta \) is a positive constant, known as the learning rate. This rule implies that the repeated application of the weight correction can lead to weight blow-outs. To take care of this it has been proposed [227, 228] to replace equation 1.28 by:
\[ \Delta w_{kj}(n) = \eta \left( y'_k(n) - \langle y'_k(n) \rangle \right) \langle x_j(n) - \langle x_j(n) \rangle \rangle \]  

(1.29)

where \( \langle \cdot \rangle \) represents the time average operation/operator. This allows both weight increase and decrease as well as weight stabilization at \( y'_k(n) = \langle y'_k(n) \rangle \) and \( x_j(n) = \langle x_j(n) \rangle \). Also various other techniques used to limit weight growth are "clamping of weights", a forgetting term to reduce and limit weight growth or a renormalization term. Forgetting or decay term modifications take the following forms:

\[ \Delta w_{kj}(n) = \eta \left( x_j(n) y'_k(n) - \sum_i y'_i(n) w_{ji} \right) \]  

(1.30)

or,  

\[ \Delta w_{kj}(n) = \eta \left( x_j(n) y'_k(n) - \sum_i y'^2_i(n) w_{ji} \right) \]  

(1.31)

The Hebbian learning rule is adaptable to various algorithmic implementations. Some of the error-correcting rules can be thought to be a type of Hebbian learning rule of the more general nature, where the exact form of the weight update equation may be written in the form (where the desired output is suppressed in the equation):

\[ \Delta w_{kj} = f'(e_k, x_j) = f(y'_k, x_j) \]  

(1.32)

Hebbian learning rules are used for both supervised and unsupervised learning. The most important applications of these rules arise in the Rosenblatt's Perceptron training rule [209], Associative Memory Learning Rules (see [260]
for Hebbian rules and variants), and unsupervised non-competitive learning rules including Principal Component Analysis [154, 186, 187, 218, 290].

1.3.5 Competitive Learning or Inhibitory Learning

This is also known as winner-takes-all learning [216, 288]. In this there are groups (clusters) of nodes at the same level of network organization, that are similar in nature but with different input weights (which to begin with may be initialized randomly). The weights are restricted to be positive semi-definite and the norm of the weight "vector" associated with each node \((j^{th})\) is normalized (usually) to 1,

\[\sum_i w_{ji} = 1, \quad \forall j \text{ or } \|w_j\| = 1; \quad \forall j \] (1.33)

The nodes compete with each other to associate the input patterns with themselves. Inhibitory connections exist which make only one node respond to a particular input pattern, these connections make the node with the largest net input, for a particular pattern, have the output 1 while all other node output are driven to zero (inhibited).

\[y'_k = \begin{cases} 1 & \text{if } v_k > v_j; \forall j \neq k \\ 0 & \text{otherwise} \end{cases} \] (1.34)

and the standard learning rule is:

\[\Delta w_{kj} = \begin{cases} \eta (x_j - w_{kj}) & \text{if node } k \text{ wins} \\ 0 & \text{if node } k \text{ loses} \end{cases} \] (1.35)

where \(\eta\) is the learning-rate parameter.
This rule has the overall effect of moving the weight vector $w_k$ of the winning node $k$ towards the input vector $x$. No learning takes place among the loser nodes of the cluster. A single layer competitive network is shown in Fig. 1.6b. The learning rule is used for the design of training algorithms for cluster analysis, pattern recognition and synthesis etc.

Early work in this area was done by Grossberg [76], von der Malsburg [272], Kohonen [126, 130, 129] and others. Competitive learning algorithms are implemented by Grossberg's Competitive Learning Theory [77, 78], Adaptive Resonance Theory (ART) learning Algorithm [22, 24, 23, 27, 26, 25], Associative Memories [127, 128, 133], Self-Organizing Maps [126, 127, 128, 130, 129] Learning Vector Quantization (LVQ1, LVQ2, LVQ3) etc.

The above described three learning rules were deterministic in nature. The last learning rule that is described is stochastic in nature (for a more detailed explanation of learning rules including stochastic learning see [75]).

### 1.3.6 Boltzmann or Stochastic Learning

In this, the nodes (stochastic in nature) are (fully) connected to each other using symmetric weights. The network is trained by minimizing the energy (equation 1.36) of the network:

$$E = \frac{1}{2} \sum_{i,j;i \neq j} w_{ij} y_i^t y_j^t; w_{ij} = w_{ji}; w_{ii} = 0$$

where $w_{ij}$ is the weight between node $i$ and $j$, while $y_i^t$ denotes the state of the $i$th node ($y_i^{t+1} = \{-1, 1\}$ or $y_i^{t+1} = \{0, 1\}$). Training is done by simulated annealing [125] or the Gibbs sampler [70], using the probability function:

$$P(y_i^t \rightarrow -y_i^t) = \frac{1.0}{1 + e^{-\Delta E_k}}$$
where, $\Delta E_k$ is the energy change for one state flip. The first stochastic
learning network was the Boltzmann Machine [2, 97, 100, 215, 252], followed
by the Cauchy Machine [190, 255], Sigmoid Belief Networks or Logistic Belief
Network [180], which was followed by the Helmholtz machine [53, 54, 96].

Mean field theory or approximation is sometimes used in these networks
to calculate the average quantities in the system, this derives the deterministic
version of these networks [94].

1.3.7 Training Algorithm

The learning algorithms can be classified into the following groups:

1. Supervised Learning Algorithms

   (i) Backpropagation and its variants [12, 61, 62, 80, 86, 108, 111, 119,
       147, 160, 167, 171, 194, 195, 201, 204, 203, 215, 213, 214, 278, 289,
       294]. These are based on recursive application of the chain rule (for
details see [21, 88, 87, 215, 220, 235, 278, 293]).

   (ii) Network construction algorithms [63, 251, 150, 167, 188] use either
       the covariance matrix [63] or some other measure (generally heuris-
tic) to add neurons for better results. These start with a minimal
       network and slowly build up the network to the desired accuracy.

   (iii) Network pruning algorithms [86, 138, 141, 147, 166, 201] work in the
       reverse manner vis-à-vis the network construction techniques, here
       nodes are removed using heuristics like – “Is the net input to the
       node below a threshold”, etc.

   (iv) Recurrent Networks [8, 59, 114, 194, 195] are generally trained us-
ing an online version of the standard back-propagation training al-
gorithm.

2. Unsupervised Learning Algorithms
(i) Clustering algorithms [24, 23, 76, 128, 126, 130, 129].

(ii) Vector Quantization algorithms [128, 127, 130, 129, 186, 187, 288].

(iii) Associative Memories [128, 127, 133].

(iv) Stochastic Algorithms [2, 53, 54, 70, 75, 99]

1.3.8 Generalization and Fault-Tolerance

An ANN is trained to estimate a function using a finite number of training samples. The finite number of training exemplars imply that any estimate of an unknown function is always inaccurate (biased). In the absence of any assumptions about the nature of the function (its behaviour between the samples) the learning problem is ill-posed, that is, given the samples, there could be many (even infinite) system/models that could have generated the samples. This problem is common to all parameterized modeling methodologies. Meaningful estimation is possible only for sufficiently smooth functions where the function smoothness is measured with respect to sampling density of the training data. For high-dimensional functions it becomes difficult to collect enough samples to attain this density. This problem is commonly referred to as the “curse of dimensionality.” This problem manifests itself as the “generalization” or the “overfitting” problem. That is, the network has a tendency to over-fit the data – the errors on the training samples are low while on new inputs the network gives high errors.

Additional constraints, reflecting a-priori knowledge about the system for which the ANN model is to be built, are used during training to impose a smoothness constraint on possible solutions in order to come up with a unique solution. This method of model complexity control is also known as the regularization method. Thus, regularization increases the generalization capabilities (the capability to give “correct” outputs for data on which training has not occurred) of the network. [41, 72, 172, 173, 206, 263, 267, 268, 269]

\footnote{For the mean squared error, this is reflected in the so-called bias-and-variance dilemma.}
Due to the distributed nature of processing in neural networks, it is generally thought that ANNs are inherently fault-tolerant. Current research indicates that ANNs are not inherently fault-tolerant, and the fault tolerance has to be improved by an adequate scheme [8, 29, 42, 50, 60, 84, 112, 113, 141, 161, 170, 184, 193, 192, 225, 226]. Techniques for fault tolerance training range over the following:

(a) Noise injection,
(b) Fault injection during training,
(c) Sensitivity analysis and pruning of node,
(d) Sensitivity analysis and addition of nodes and
(e) A combination of these techniques.

It has been found that the generalization capabilities of an ANN are correlated with its fault-tolerance properties, that is networks that generalize well are also fault-tolerant and vice-versa. [6, 20, 42, 50, 60, 84, 85, 101, 113, 259]

1.4 Learning Tasks

The set of all learning problems can be subdivided into four classes of common problems [41]:

- Classification,
- Regression,
- Density Estimation, and
- Clustering / Vector Quantization.
1.4.1 Classification

In a (n-class) classification problem the output of the system takes on only \( n \) (symbolic) values \( y = \{1, 2, 3, \ldots, n\} \). In this case the output of the learning machine need only take on discrete values, i.e. corresponding to a given input \( x \), a particular value of \( y \) is output, more than one particular value of input \( x \) may correspond to the same \( y \) value. A commonly used loss function for this problem is (equation 1.38):

\[
L(y, y') = \begin{cases} 
1 & \text{if } y = y' \\
0 & \text{if } y \neq y' 
\end{cases} \tag{1.38}
\]

Instead of discrete values of the output function, sometimes it is required to associate a continuous range of output values to belong to a particular class. That is, one may partition the continuous interval \([a, b]\) into \( n \) sub-intervals \([\gamma_0, \gamma_1), [\gamma_1, \gamma_2), \ldots, [\gamma_{n-2}, \gamma_{n-1}), [\gamma_{n-1}, \gamma_n]\), such that:

\[
\begin{align*}
\gamma_0 &= a & \quad & (a) \\
\gamma_n &= b & \quad & (b) \\
\gamma_0 &< \gamma_1 < \ldots < \gamma_{n-1} < \gamma_n & \quad & (c) \\
\text{and } [a, b] &= \left( \bigcup_{i=0}^{n-1} [\gamma_i, \gamma_{i+1}) \right) \bigcup \left[\gamma_{n-1}, \gamma_n\right] & \quad & (d)
\end{align*}
\tag{1.39}
\]

and label each of the above defined sub-intervals as:

\[
s_i = \begin{cases} 
[\gamma_{i-1}, \gamma_i) & i = 1, 2, \ldots, n - 1 \\
[\gamma_{n-1}, \gamma_n] & i = n
\end{cases} \tag{1.40}
\]

Then \( s_i \)'s can act as the labels of the classification with the index of \( s \) acting as the classification index (sometimes a band of values is used such that they represent the values for which indeterminate classification is assumed to
happen, that is the boundaries of the classes are pushed apart and the classes do not have a common boundary)[63]. In this case one may define the loss / error function as:

\[
L(y, y') = \begin{cases} 
1 & \text{if } \exists i \text{ s.t. } y \in s_i \land y' \in s_i \land i \in \{1, 2, \ldots, n\} \\
0 & \text{otherwise}
\end{cases}
\]  

(1.41)

1.4.2 Regression

Regression is the process of estimating a real-valued function based on a finite set of “noisy” samples. Thus, from the exemplars set \((y_i, x_i)\) one attempts to fit a model

\[
y = R(x) + \epsilon
\]

(1.42)

where \(\epsilon\) is the error in measurement of \(y\) and takes into account the effect of all unknown factors on \(y\). This assumes that the independent variables of the models, the \(x\)'s are measured without error. But in general, this may not be the case, and if,

\[
x = x' + \delta
\]

(1.43)

where \(\delta\) is the vector of the error in the measurement of \(x\). Then equation 1.42 may be re-written as:

\[
y = R(x') + \epsilon
\]

(1.44)
Learning in Sigmoidal Feedforward Artificial Neural Networks

This leads to the measurement error models [36]. Regression is a special case of measurement error models in which \( \delta = 0 \). As is clear, the case of function approximation (\( \delta = 0, \epsilon = 0 \)) is a special case of the regressional problem. In this thesis, FFANN's are viewed from the point of function approximation tasks, primarily.

1.4.3 Density Estimation

In this case the outputs \( f(x, \omega), \omega \in \Omega \), denotes a set of densities. Assuming that the unknown density function \( f(x, \omega_0) \) belongs to this class and given a set of identically and independently distributed (i.i.d.) set of training data \( x = [x^1, x^2, \ldots, x^n] \), the likelihood function (which is to be minimized for the estimation problem) is defined as:

\[
P(x|\omega) = \prod_i f(x_i, \omega)
\]  

and the objective function to be optimized is the log likelihood function (to be maximized) defined as:

\[
\mathcal{R}(\omega) = -\sum_i \ln(f(x_i, \omega))
\]

or the density estimation problem can be "solved" using a (component-wise) loss function defined as:

\[
\mathcal{L}(f(x, \omega)) = -\ln(f(x, \omega))
\]
1.4.4 Clustering and Vector Quantization

Here the aim is to partition the x-space into a prespecified number of regions so that future samples are drawn from a particular region are approximated by a single point. Here, the aim may be to reduce the dimensionality of the problem. The commonly used loss function and the error function to be minimized are, respectively:

\[ \mathcal{L} \equiv ||x - f(x, \omega)||^2 \]  \hspace{1cm} (1.48)

\[ \mathcal{E} \equiv ||x_i - f(x_i, \omega)||^2 \]  \hspace{1cm} (1.49)

1.5 Scope and Contributions of This Thesis

Sigmoidal activation function neural networks are one of the most widely used classes of neural networks. Here, we have considered Sigmoidal Feedforward Artificial Neural Network (SFFANN)s. Feedforward Artificial Neural Network (FFANN)s with one or more non-linear hidden layers\(^4\) can approximate any continuous function arbitrarily well as the number of neurons (basis functions) increases without bound [67, 49, 66, 104, 105, 256, 103, 10, 38, 257, 31, 40]. In addition, they have been successfully applied to many complex problems, including speech recognition [208], handwritten digit recognition [145], financial modeling

\(^4\)The set of suitable activation functions for hidden nodes includes a spectrum of discontinuous and continuous sigmoidal functions [58, 240], the radial basis functions (RBF\(^s\)), the generalized radial basis functions (GRBF\(^s\)) etc. [52, 199, 198, 71, 48]. Moreover, it has been established that any arbitrary non-linearity [137], specifically a non-polynomial activation function [151, 168] is sufficient for a FFANN to have the universal approximation property.
and medical diagnosis [14], target detection [189], time series forecasting [165, 270] among others.

The universal approximation results for SFFANNs are existential in nature. That is, they assure that given a function to be approximated, there exists a FFANN that approximates it arbitrarily well. The practical utility of these results are minimal in the sense that they cannot be used to obtain the approximant. Another important set of results is the qualitative bounds on the approximation achievable by using fixed sized networks (size of the network is represented by the size of the hidden layer). Here the classical result, of $O(n^{-\frac{3}{2}})$ (here represents the number of nodes in the hidden layer, the function to be approximated is assumed to satisfy certain smoothness conditions for this approximation to hold), is due to Barron [10] (but see also [52, 106, 168, 221]). The qualitative behaviour of the function sets representing FFANN’s have been relatively unexplored. In this thesis the emphasis is on the exploration of the qualitative properties of the neural networks, or more precisely, on the set-theoretic and topological properties of the function sets represented by sigmoidal feedforward neural networks. That is, this thesis also makes an attempt to evaluate the qualitative properties of the learning (system) in (embodied by) sigmoidal feedforward neural networks.

The central themes of the thesis are:

- The investigation of the set-theoretic/topological properties of the function sets represented by SFFANNs,
- The theoretical investigation of the fault-tolerance properties of the SFANNs,
- The development of one parameter sigmoidal activation classes,

This holds true even in the case of constructive proofs of the universal approximation theorem(s), as the proof procedures require arbitrary/full information about the function to be approximated through function values at the proof procedure specified input points.
The development of a framework for the empirical investigation of fault-tolerance properties of FFANN and

- The unified presentation of the different noise based regularization techniques for SFFANNs training.

The thesis proposes the following:

- That the finite sized single layer sigmoidal feedforward artificial neural networks with arbitrary weights do not possess the uniform boundedness property, the equicontinuous property, the closed set property, the compactness property and the best approximation property.

- The class of networks in which only the hidden to output node connections are bounded, are universal approximators.

- There exists classes of sigmoidal feedforward artificial neural networks that possess the uniform boundedness property, the equicontinuous property, the closed set property, the compactness property and the best approximation property for the finite sized networks. These networks are required to have bounded weights between the output and the hidden layer nodes. The critical role played by the hidden to output nodes for the possession of the best approximation property is elucidated. For these networks it is established that the best approximant is not unique.

- There exists a class of the sigmoidal feedforward artificial neural networks that not only is an approximator of continuous and bounded functions but also possesses the best approximation property (the best approximation being unique). These networks also have the closed set property, the equicontinuous property, the uniform boundedness property and the compactness property. This class of neural networks is obtained by the imposition of a global criterion on the hidden to output weights (boundedness criteria on the sum of these weights). This class is estab-
lished to be an approximative set for bounded and continuous function. This class is shown to possess the best approximation property. It is also demonstrated that the best approximation is unique in this case.

- Theoretically analyses the fault tolerance of the sigmoidal feedforward artificial neural network to hard faults as well as gaussian noise contamination.

- In this work, the generally used networks are established to be non-fault tolerant inherently. This result is obtained by correlating the equicontinuity property of functional sets represented by the sigmoidal networks with the fault - tolerance properties. A class of sigmoidal networks is given that is robust to hard faults as well as noise contamination. For the different classes of networks, error measures to hard and soft faults are theoretically established.

- The error due to single hard faults is determined.

- The functional relation between the noise contamination and the output error for the classes of networks considered is established.

- A framework for the empirical investigation of fault - tolerance properties of sigmoidal networks is formulated and exemplified by an example.

- Two novel methods for the generation of sigmoidal classes are proposed.

- The first method uses any given sigmoid for the generator of the class. Conditions are established under which even the envelope of the derivatives of the members of this class is a sigmoid. In general the members of the classes thus generated have asymmetric / skewed derivatives, this allows the escape from flat regions near the origins. This provides an alternative to the usual heuristics of using a small “twisting” term to alleviate this problem. The members of these classes also “appear” as translates of themselves, thus effectively shifting the origin
and lead to a solution of the weight saturation problem in feedforward neural network training.

- The self-adaptation of activation function during training is conjectured to be a better method for training. The conjecture is made on the basis of experiments conducted. And a new activation function is proposed.

- Noise injection during training has been one of the recently developed methods for controlling the model complexity during sigmoidal feedforward artificial neural network training. Various techniques have been demonstrated, these techniques are brought together in one unified theory of noise based training.

- The interplay between the various noise mediated terms is explained. It is shown how the introduction of noise in one system parameter leads to the regularization effect on its "dual" parameter.

- The relation between noise based regularization and Tikhonov's regularization paradigm is explained.

- The noise based regularization leads to the modification of the loss / risk functional for training by the addition of both positive definite terms as well as positive indefinite terms. The beneficial effect of the positive (semi-) definite term is explained.

- The relation between weight regularization and noise based regularization is explained.

### Organization of the Thesis

The thesis is organized as follows:

- **Chapter 1.** This chapter presents a summary of the conventional neural network literature, summaries the general topics of this work, and presents a brief overview of the neural networks.
and lead to a solution of the weight saturation problem in feedforward neural network training.

• The self-adaptation of activation function during training is conjectured to be a better method for training. The conjecture is made on the basis of experiments conducted. And a new activation function is proposed.

• Noise injection during training has been one of the recently developed methods for controlling the model complexity during sigmoidal feedforward artificial neural network training. Various techniques have been demonstrated, these techniques are brought together in one unified theory of noise based training.

• The interplay between the various noise mediated terms is explained. It is shown how the introduction of noise in one system parameter leads to the regularization effect on its “dual” parameter.

• The relation between noise based regularization and Tikhonov’s regularization paradigm is explained.

• The noise based regularization leads to the modification of the loss / risk functional for training by the addition of both positive definite terms as well as positive indefinite terms. The beneficial effect of the positive (semi-) definite term is explained.

• The relation between weight regularization and noise based regularization is explained.

1.6 Organization of the Thesis

The thesis is organized as follows:

Chapter 1. This chapter presents a summary of the conventions used in artificial neural network literature, summarizes the different com-
ponents of the artificial neural network model and presents the objectives of this thesis.

Chapter 2. In this chapter the closure properties of the functional sets represented by sigmoidal feedforward artificial neural networks is presented. The results are established for both linear output and sigmoidal output networks. The weight spaces considered are the (a) unbounded weight set, (b) partially bounded (only hidden nodes to output) weight set and (c) the bounded weight set.

Chapter 3. In this chapter investigation of the best approximation property and other associated properties of the single layer, feed forward, sigmoidal artificial neural networks is presented. The obtained results demonstrate the set-theoretic and topological properties of these networks. The importance of the best approximation property for any approximation scheme is stressed and the best approximation properties of the sigmoidal network approximation scheme presented. The uniform boundedness property, the equicontinuous property, the closure property, the compactness property and the best approximation property of the unbounded weight set, the partially bounded weight set and the bounded weight set sigmoidal feedforward artificial neural networks is presented.

A class of bounded weight network is presented that possesses the best approximation property for the approximation of bounded and continuous functions.

Chapter 4. In this chapter the equicontinuous property of the functional sets represented by the sigmoidal feedforward artificial neural networks is related to the fault-tolerance properties of these networks. The absence of robust fault tolerance properties is demonstrated
for the generally used sigmoidal feedforward artificial neural networks. A new class of sigmoidal feedforward artificial neural networks is shown to have robust fault-tolerance properties. The induced error bounds for the induction of a set of faults (both hard and/or soft) is established for the three types of networks.

Chapter 5. In this chapter the learning system model is used to formulate a framework for the empirical investigation of the fault-tolerance properties of the sigmoidal feedforward artificial neural networks. The framework describes a hierarchy of faults and the associated fault measures. The results for a experiment is presented. The experiment establishes the non-fault-tolerant behaviour of feedforward neural networks. Though, the results indicate that partial fault-tolerance is inherently present in the feedforward architecture.

Chapter 6. This chapter presents an unified theory of noise based regularization for model complexity control in sigmoidal feedforward artificial neural network training. The various methods of noise based training are derived from a consistent model that allows the demonstration of the interplay between noise injection in one network parameter leading to regularization of its “dual” parameter. The penalty/regularization terms associated with the noise induction are established. In the small error limit near the “optimal” network, the penalty terms are shown to equivalent to Tikhonov’s regularization terms in the phase space of inputs, outputs, desired outputs weights and the output of the hidden layer nodes.

Chapter 7. In this chapter the log-sigmoidal function is used to generate a class of sigmoidal functions. The class is parameterized by a single
continuous parameter. The properties of the generated class are presented and its suitability for usage in sigmoidal feedforward artificial neural networks shown. Experimental results are described that allow the proposal of a new activation function for the usage in sigmoidal networks. The proposed activation has a skewed derivative as compared to the generally used activations whose derivative is "bell-shaped" and symmetric.

Chapter 8. In this chapter the results of the previous chapter are extended and two novel methods for the generation of one sigmoidal parameter function class is presented. The properties of the classes generated is established. The methods use any given sigmoidal function or any given bounded and uni-modal "bell-shaped" function to generate the sigmoidal classes. The conditions under which the procedure can be followed, that is the conditions to be satisfied by the "generating" functions of the class are obtained.

Chapter 9. In this chapter an argument for the desirability of the self-adaptation of the activation function during sigmoidal feedforward artificial neural network training to increase the learning rate of training is presented.

Chapter 10. This final chapter summarises the theoretical and experimental results. It discusses their relevance and suggests a few directions for further research.

1.7 Conventions Used

In this thesis the term “sigmoidal FFANN’s”, “feedforward networks”, “nets/networks”, etc. are used as the synonym for the set (or a member of the set) represented by sigmoidal FFANN’s. The sigmoidal activation explicitly used in
this work is the log - sigmoidal activation, \( f(x) = (1 + e^{-x})^{-1} \), though the results obtained can easily be proven to be true for a class of activation functions (see Chapter 2-5). This thesis does not consider the case of vector valued FFANN’s. The networks considered in this work are networks with one single hidden layer only. The obtained results can easily be extended for networks with more than one hidden layers. Equations, definitions, theorems, etc. are numbered sequentially (separate sequence for each) in the order of appearance. The proofs are terminated by a small empty box (\(\Box\)). These are the conventions that are generally used in this thesis, where-ever required, other convention components are defined.

1.8 Conclusions

In the thesis, the analysis of the behaviour of the trained networks is conducted from a function approximation point of view. The function sets represented by the different classes of sigmoidal activation feedforward neural networks is investigated. The set-theoretic / topological properties of these networks presented. The fault - tolerance property of these networks is analyzed. A framework for the empirical investigation of the fault tolerance properties of these networks is presented. Two methods for the generation of sigmoidal function classes are obtained. The members of the generated class are established to satisfy the criterion of the universal approximation theorems for these networks and the criterion imposed by the requirements of the weight update rules, that is, the condition of continuity, monotonicity and differentiability. An experiment is described for investigation of the suitability of activation function self-adaptation during training and the directions further research enumerated.