**ANN Concepts and Various Optimization Techniques**

**5.1 Introduction**

A common problem in engineering, economics, or computer science is to estimate a sequence of unknown random variables from a sequence of observations that are statistically related to the hidden variables of interest. In civil engineering, prediction of strength characteristics of a concrete structure from the experimental data, durability of heavy civil structures like bridges, multi storied buildings, damage identification in structures, structure health monitoring, tide forecasting, earthquake induced liquefaction are the ones where Artificial Neural Networks (ANN) can be applied. The application of ANN models to predict the highly desirable strength characteristics of different configurations of concrete structures like SIFCON has great potential in arriving at solution to inverse problems that are nonlinear in nature and proves advantageous over the conventional methods. The mathematical model of neural network is composed of a large number of processing elements organized into networks. The origin of ANNs is in the field biology. The biological brain consists of billions of highly interconnected neurons forming a Neural Networks (NN). Human information processing depends on this connectionist system of nervous cells. The difference between information processing in brain and computer is based on their ability of pattern recognition and learning. The computer can perform calculation involving large numbers at high speeds, but it cannot recognize something such as a classification problem, written text, data compression and a learning algorithm. On the contrary, a human easily recognizes and deals with the challenges mentioned above by processing information with highly distributed transformations through thousands of interconnected neurons in the brain. The accuracy of ANN model depends on transfer function, size of training sample, network topology, and adjusting the weights using optimisation techniques such as Steepest Descent (SD), Levenberg-Marquardt (LM), Conjugate Gradient (CG) and Genetic Algorithms (GA). This chapter provides an overview of neural network architecture, and explains how a typical neural network is constructed using various learning (or optimization/weights adjustment) techniques.

The structure of the chapter is as follows. Section 5.2 begins with a brief introduction to the evolution of neural networks. Section 5.3 reviews some of the applications of ANN in the field of structural analysis and design followed by the ANN concepts in section 5.4. Section 5.5 explains the various optimization techniques used to optimal weight updation in the ANN model development. Finally the conclusions have been summarized in section 5.6

**5.2 History of ANNs**

During the late 19th and early 20th centuries, the background work in the field of neural networks started and the scientists named Hermann von Helmholtz, Ernst Mach and Ivan Pavlov conducted an interdisciplinary work in physics, psychology and neurophysiology. This early work emphasized on general theories of learning, vision, conditioning, etc., and did not include specific mathematical models of neuron operation.

With the work of Warren McCulloch and Walter Pitts, the modern view of neural networks began in the year 1940, wherein it was shown that networks of artificial neurons can be used to compute any arithmetic or logical function by creating threshold switches based on neurons. Their work is often acknowledged as the origin of the neural network field (McCulloch W.S and Pitts W, 1943).

The work of McCulloch and Pitts was followed by Donald Hebb, where he proposed that classical conditioning (as discovered by Pavlov) is present because of the properties of individual neurons. He proposed a classical Hebbian rule which is a simple and basic mechanism for learning in biological neurons (Donald O. Hebb, 19).

Later in late 1950s, Frank Rosenblatt and his colleagues formulated the perceptron network and associated learning rule. They built a perceptron network and demonstrated its ability to perform pattern recognition (Rosenblatt F, 1958). This early success generated a great deal of interest in neural network research and their work is considered as the first practical application of artificial neural networks. But, it was later shown that the basic perceptron network could solve only a limited class of problems. Also around the same time, Bernard Widrow and E. Hoff introduced the ADALINE (ADAptive LInear NEuron), a new fast and precise learning algorithm and used it to train adaptive linear neural networks, which were similar in structure and capability to Rosenblatt’s perceptron network. The Widrow-Hoff learning rule is still in use today (Widrow B and Hoff M E, 1960).

The scientists Marvin Minsky and Seymour Papert observed that both Rosenblatt’s and Widrow’s networks suffered from the same inherent limitations. Rosenblatt and Widrow were aware of these limitations and accordingly proposed new networks that would overcome these limitations. However, they were not able to successfully modify their learning algorithms in order to train the more complex networks.

During the late 1960s, with the influence of Minsky and Papert and in addition, with the lack of new ideas and powerful digital computers with which to experiment, the interest in neural networks had faltered and for a decade neural network research was largely suspended (Minsky M and Papert S, 1969).

Teuvo Kohonen and James Anderson (1972) independently and separately developed new neural networks that could act as memories. They proposed a “linear associator”, a model for associative memory (T. Kohonen, 1972; J. A. Anderson, 1972). Further, Teuvo Kohonen described the self-organizing feature maps also known as Kohonen maps. This is a mechanism which involves self-organization in the brain (Teuvo Kohonen, 1989). Stephen Grossberg was also very active during this period in the investigation of self-organizing networks (S. Grossberg, 1980). During the 1980s, both of the impediments pointed by Minsky and Papert were overcome, and research in neural networks increased dramatically. New personal computers and workstations, which rapidly grew in capability, became widely available. In addition, important new concepts were introduced. Two new concepts were most responsible for the rebirth of neural networks. The first was the use of statistical mechanics, described by physicist John Hopfield which explains the operation of a certain class of recurrent network, which could be used as an associative memory (J. J. Hopfield,, 1982). The second key development of the 1980s was the back propagation algorithm for training multilayer perceptron networks, which was discovered independently by several different researchers. The most influential publication of the back propagation algorithm was by David Rumelhart and James McClelland (D. E. Rumelhart and J. L. McClelland, 1986). This algorithm was the answer to the criticisms made by Minsky and Papert in the 1960s. These new developments reinvigorated the field of neural networks. Since 1980s, neural networks have found countless applications, and the field has been buzzing with new theoretical and practical work.

**5.3 Evolution of Neural Networks**

Modern conventional digital computers are of Von Neumann architecture which has powerful processing units and memory. They are able to perform any complex numerical computation by sequentially executing an algorithm at a very high speed. Without an algorithm a computer cannot solve any problem. But, there are some categories of problems, where the problem has many factors which are difficult to analyze, making it difficult to formulate an algorithm. This indicates that digital computers are only suitable for numeric computation and they are not adaptive. In case of humans, even though they are not fast in numeric computation, they are able to solve some problems with ease. The significant difference is, in computers most of the units remain idle and only the processing units perform the task. But in case of humans, the neurons in the brain continuously reorganize themselves and work in parallel to learn and expertise. In general all biological neural functions, including memory are stored in the neurons and in the connections between them (David Kriesel, 2005). Even though biological neurons are very slow when compared to electrical circuits (10-3s compared to 10-10s), the brain is able to perform many tasks (pattern reorganization, prediction, optimization etc.,) much faster than any conventional computer (Martin T. Hagan, et.al, 2014).

**5.3.1 Computers vs Neural Networks**

Several efforts have been made to make the computer mimic the brain so that it can learn and behave like biological neurons. The realized biological neurons are called artificial neurons. These artificial neurons are realized as elements in a program or circuits made of silicon. Networks of these artificial neurons do not have a fraction of power of the human brain but they can train to perform useful functions by operating all the neurons at the same time (Martin T. Hagan, et.al, 2014). Some of the differences between digital computers and the realized biological NNs are summarized below (Table 5.1).

Table 5.1 Performance of Digital computers vs. realised biological NNs

|  |  |
| --- | --- |
| **Digital Computers** | **NNs** |
| Deductive Reasoning. We apply known rules to input data to produce output. | Inductive Reasoning. Given input and output data (training examples), we construct the rules. |
| Computation is centralized, synchronous, and serial. | Computation is collective, asynchronous, and parallel |
| Memory is packetted, literally stored, and location addressable. | Memory is distributed, internalized, short term and content addressable |
| Not fault tolerant. One transistor goes and it no longer works. | Fault tolerant, redundancy, and sharing of responsibilities. |
| Exact. | Inexact. |
| Static connectivity. | Dynamic connectivity. |
| Applicable if well-defined rules with precise input data. | Applicable if rules are unknown or complicated, or if data are noisy or partial. |

In the following sections various aspects of biological neurons, artificial neurons and analogy between them are presented.

**5.3.2 Biological Neuron structure**

Each biological neuron has three principal components (dendrites, cell body and axon) as shown in Fig. 5.1a. The dendrites are tree-like receptive networks of nerve fibres that carry electrical signals into the cell body. 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. A neuron is connected to other neurons through about 10,000 synapses (Fig. 5.1b).

A neuron receives input from other neurons and then all inputs are combined. Once input exceeds a critical level, the neuron discharges a spike ‐ an electrical pulse that travels from the body, down the axon, to the next neuron(s). The axon endings almost touch the dendrites or cell body of the next neuron. Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters. Neurotransmitters are chemicals which are released from the first neuron and which bind to the second. This link is called a synapse. The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

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.

Axon from another cell

Axon

Synapse

Nucleus

Synapses

Cell Body or soma

Dendrite

**Dendrites**: Input

**Cell body**: Processor

**Synaptic**: Link

**Axon**: Output

Fig. 5.1 a Biological neuron

Neural structures continue to change throughout life. These later changes tend to consist mainly of strengthening or weakening of synaptic junctions. New memories are formed by modification of these synaptic strengths. Thus, neurons have the capability to memorize, learn and expertise the data.

Axon

Cell Body

Basal dendrites

Segment of dendrite

Synaptic inputs

Dendritic spines

Apical dendrites

Synaptic terminals

Fig. 5.1b Biological Neuron connection to other neuron

**5.3.3 Artificial Neuron**

Each artificial neuron within the artificial neural network is usually a information processing unit which takes one or more inputs and produces an output (Fig. 5.2a). At each neuron, every input has an associated weight which modifies the strength of each input. The neuron simply adds together all the inputs and calculates an output to be passed on (Fig. 5.2b).

x1

x2

x3

x4

x5

w1

w2

w3

w4

w5

Output is fed to other neurons

Inputs

Fig. 5.2a model of Artificial Neuron

........................

∑=X1+ X2+............. Xm = y

Input

Processing

Output

Fig. 5.2b Input and output of Artificial Neuron

**5.3.4 Analogy between Biological neurons and artificial neurons**

An artificial neuron is an imitation of a human neuron. In ANN, the weight corresponds to the strength of a synapse, the cell body is represented by the summation and the transfer function, and the neuron output represents the signal on the axon (Simyon Haykin, 1999). Figure 5.3 shows the simplified schematic diagram of analogy between biological neuron and artificial neuron.

The model of the neuron which forms the basis of the design of artificial neural network has

1. The set of synapses or connecting links, each of which is characterized by weight or strength of its own (xj × wj ). Specifically a signal xj at the input synapse j connected to the neuron k is multiplied by the weight, wj.
2. An adder for summing the input signals, weighted by respective synapses of the neuron, the operation described here constitute a linear combiner (∑xiwi = x0w0+x1w1 + ….+xnwn = output)
3. An activation function for limiting the amplitude of output of the neuron (Fig.5.3b).

**Artificial Neuron**

Y0=W0X0

Y1=W1X1

......

YN=WNXN

W0

W1

......

WN

X0

X1

......

XN

Output

Weights

Processing Element

Activation Function

Interconnects

Soma

Axon

Dendrites

Conduction

**Biological Neuron**

Fig. 5.3 Biological Neuron and its equivalent artificial neuron

**5.3.5 Evolution of ANN from Biological neural system**

The ANNs which are motivated by the biological neural systems, are constructed with very simple but numerous neurons that work massively parallel and have the capability to learn. There is no need to explicitly program a neural network. ANN can learn from training samples. Learning is a procedure in which the network is given with set of inputs and their corresponding set of outputs (Fig. 5.4).

The result from this learning procedure is the capability of neural networks to generalize and associate the data. After successful training a neural network can find reasonable solutions for similar problems of the class that were explicitly trained. This in turn results in a high degree of fault tolerance against noisy data.

Input layer with 11 nodes

hidden layer with 4 nodes

Output layer with 2 nodes

node 1

node 2

node 10

node 11

Fig. 5.4 A three layer ANN formation

**5.3.6 ANN Capabilities**

The capabilities of the neural network that make them to be popular are given below.

1. Non-linearity: An artificial neuron can be linear or non-linear. A neural network, made up of interconnections of non-linear neurons, is itself non-linear. Non –linearity is a special property that is distributed throughout the network.
2. Input-Output mapping: In the process of learning mechanism a set of examples of inputs and their desired outputs are applied to the neural network for training. The training process is continued until the network stabilizes at which there are no significant changes in the synaptic weights. Thus the network learns from set of examples by constructing input and output mapping.
3. Adaptivity: Neural networks have built in capability called adaptivity in which the synaptic weights change with the real world. The property of adaptivity makes the system to be stable and robust.
4. Fault tolerance: This property makes the network to be tolerant against internal and external errors.
5. VLSI capability: the massively parallel nature of the network makes it potentially fast for computation of certain tasks. This feature makes the network well suited for VLSI technology.

**5.4 Application of ANNs in civil engineering**

Examples found in the Literature of general applications of neural networks within civil engineering include a wide array of topics such as (Jeng et.al., 2003):

1. Horizontal formwork selection
2. Control of structures under
3. Simple truss design
4. Structural damage detection
5. Prediction of tower guy pretension
6. Dynamic analysis of bridges
7. Nondestructive examination of concrete

In addition, neural networks are most suitable for applications that have the following features:

1. A complex problem with a large number of governing parameters;
2. A need for an alternative to a mathematical formulation of a solution to the problem.

**5.5 ANN basic concepts**

An ANN consists of interconnections between processing elements called neurons. Neurons are the basic computing elements that perform data processing inside a network. In order to provide reliable predictions for new situations which contain partial information, the neurons learn incrementally from their experimental data to capture the linear and nonlinear trends in the complex data. The processing ability of the network is stored in the interneuron connection strengths called parameters or weights. The weights are acquired by adaptation process.

* + 1. **Input and output of a Neuron**

The variables applied to a neuron are called its inputs and the output of the neuron is its value. The graphical representation of a simple basic neuron is shown in Fig.5.5. The neuron’s output (y) is a nonlinear combination of the inputs, {xi}. These inputs are weighed by the parameters, {wji}, often termed weights, or synaptic weights. The neuron’s output can be written as

y = *f*(x1, x2... xi; vj; wj1, wj2, . . . ,wji)

Function [*f*(.)] is the activation function and vj is termed as the bias input. The bias vj which a neuron has is summed with the weighted inputs to form the net input. The function [*f*(.)] that is performed by neurons depends on the weight vector on the neurons. The weight vectors are usually determined in so called “training phase” using a learning algorithm. These weights determine the output of the neural network; therefore, it can be said that the connection weights form the memory of the neural network. Figure 5.5 shows the details of the elementary neuron structure with input as *xi*. Each input is weighted with an appropriate weight wji. The sum of the weighted inputs and the bias, vj forms the input to the transfer function [*f*(.)]. Neurons can use any differentiable transfer function, [*f*(.)] to generate their output and is given by

Output (y) = f( input1 × weight1 + input2 × weight2 + ... ) = 

x 1

bias vj

x 2

x i

xn

wji

wj2

f (.) transfer function

jth neuron output  **y**

wj1

Fig.5.5 Neuron inputs and outputs

In mathematical terms, ANN is defined as a directed diagram with the following properties:

1. An input vector xj associated with each node j (referred as neuron j)
2. A real valued weight wji is associated with a link (referred as synapses) between two nodes j and i.
3. A real valued bias (referred as activation threshold) vj is associated with each node j
4. A transfer function *f*j [xi, wji, vj, ( i ≠ j)] is defined for each node j, which determines the state of the node as function of its bias, the weights of its incoming links and the states of the nodes connected to it by these links.

Nodes without links towards them are called input neurons and the output neurons are those without a link leading away from them. A feed forward network is one whose topology has no closed paths. The transfer function takes the form as given below.

 (5.1)

where, *f*(z) is either a discontinuous step function or smoothly increasing generalization known as a sigmoidal function.

**5.5.2 Neural Network architecture**

A single neuron cannot learn all the training sets. Therefore, interconnection of neurons or design of an ANN is a must to perform the desired task. Two types of ANNs are used for training the experimental data. They are i) Feed forward ANN, also called as multilayer perceptron and ii) Recurrent neural networks.

Feed forward networks are very popular and are used in our design. The first term, feed forward describes how this ANN processes and recalls patterns. In this network, neurons are only connected forward. Each layer of the ANN contains connections to the next layer. Feed forward ANNs have significant computational power because the interconnections have no closed paths or loops, whereas connections between neurons form a directed cycle in case of recurrent ANNs. For feed forward neural networks, the back propagation rule is the most widely used learning algorithm. It is a form of supervised training.

In feed forward networks, the network must be provided with the sample inputs and expected outputs. The expected outputs are compared against the actual outputs for a given input. From the anticipated data, the Back-propagation training algorithm calculates the error and adjusts the weights of the various layers backwards from the output layer to the input layer.

**5.5.3 Three Layer feed forward Neural Network Model**

The three layers in a feed forward model are the input, a parallel hidden layer denoted as ‘’ and an output layer denoted as ‘’. The input layer is an “isolated” layer. This layer is distributed to the entire hidden layer neurons. No real processing is done in this layer. The key to the operation of the neural network model is the hidden layer ‘’. Each of the hidden nodes is nothing but a single neuron. It implements its own decision surface. The output layer is a collection of decision surfaces. In this layer, each of its neurons has decided as to the part of the decision space in which the input vector lays. The role of the output layer is essential to combine all of the “values” of the hidden layer neurons and decide upon the overall classification of the vector. The nonlinearity provided by the nonlinear activation functions of the hidden and output neurons allows network to solve complex classification problems that are not linearly separable. This is done by forming complex decision surfaces by a nonlinear combination of the hidden layer’s decision surfaces.





Bias =1

Bias =1

Input layer

Hidden layer

Output layer

Fig.5.6 Three layer feed forward network

..............

..............

Fixed input x0 = +1

x1

x2

xm

wk0 =bk vk (bias)

Inputs

Synaptic weights

(Including bias)

Summing junction

f(.)

vk

Output

yk

Activation function

Fig. 5.7 Details of the neuron function in a feed forward network

A three-layer feed forward network is of Multilayer perceptron (MLP) model and its details are shown in Fig. 5.6 and Fig. 5.7 respectively. The feed forward equations corresponding to the inputs and outputs of Fig. 5.6 are described by the matrix Eq. 5.2.

 (5.2)

where,

is input vector with size p and one bias input with size ((p + 1) × 1)).

W=  matrix of weights between input- hidden nodes

U = matrix of weights between hidden-output nodes

y = output vector of size

*f*(.) **=** multivariate activation function.

p = number of real input nodes in the input layer.

M = number of real hidden nodes in the hidden layer ‘’.

K = number of output nodes in the output layer ‘’.

**Evaluation procedure to minimize the risk function**

The detail evaluation of Eq.5.2 pertaining to a feed forward neural network in order to minimize the risk function using back propagation algorithm is presented here.

Let  are the external inputs applied to the input layer and its output is same as the input. These’s act as input to the next layer. The inputs where  are applied to nodes in the hidden layer and the output of the hidden nodes are denoted aswhere. Here the bias inputs are considered to be zero for easy computation. The detailed diagram of Fig.5.6 is shown in Fig.5.8.

Inputs

Hidden Layer

Output Layer

Feed Forward

bk

h=*f* (*wx*+b)

y=*f* (*u*h+b)

Back Propagation

yk

*u*kj

*u*k1

*u*kM

*w*Mp

*w*11

bM

bj

*x*p

*x*4

*x*3

*x*2

*x*1

h1

hj

hM

∑

∑

∑

∑

*f*

*f*

*f*

*f*

b1

Fig.5.8 Details of three layer feed forward network

Now to compute the output at each node of hidden layer, i.e.  first the net input is computed and it is passed through the activation function as shown in Fig. 5.9 below.

 (5.3)









j



Fig.5.9 summation and activation function of neuron j at L-1th layer

Here indicates the node in layer L-1. The net input is the summation of the product of inputs and their corresponding weights that are connected to that node.

This net input is passed through the activation function to get the output of the node  at layer, and is given by

 i.e., (5.4)

Here  is the total number of inputs from input layer  connected to the hidden layer.

Similarly, the outputs of layer L - 1 act as inputs to the next output layer and the internal view of each neuron is shown in Fig.5.10. To obtain the output at layer, the net input is computed as follows

, (5.5)











k

Fig.5.10 Summation and activation function in neuron k

and the output of the layer  is given by

 (5.6)

i.e., From Eq. (5.5) we have



(here is the total number of inputs from layer connected to the layer  ).

Here are the final outputs of the network and they are the functions of all the weights and external inputs. The hidden layer can be thought of as computing an internal representation so that now a single unit can correctly predict the output.

By training the network, it learns the optimum weights of the network. Here a risk (objective or cost) function is the square of the error between the output of the network and the desired output for the training sample.

 (5.7)

Here is the risk function or loss function. Now, the objective is to minimize, which indicates that output of the network is close to the desired output. Here,  is minimized using the gradient descent algorithm. To apply gradient decedent,  should be differentiable. That is the activation function should be non-linear differentiable function.

Now the NN needs to learn the weights of the network connections using back-propagation algorithm. In terms of individual weights the gradient descent algorithm is given by

 (5.8)

As  is the squared error between the computed output and targeted output, is the differentiation of the error with respect to. Thus, we need to calculate the partial derivative of the square of the error for any training sample with respect to the weights in the network.

Any weight,  can affect  only by affecting the final output. In a layered network, the weight,  can affect the final output only through its affect on  in the next layer.

At the output layer,

Now,  (5.9)

The weights  in the hidden layer affect the net input () to the next layer.

It is known that (recall Eq. (5.5)),

 i.e.,

then,   
 (5.10)

Now, the error rate is defined as  (5.11)

Here,  cannot be differentiated with, Eq. (5.11) is given by

 (5.12)

Here,  (5.13)

And from Eq. (5.6)  (5.14)

Substitute Eq. (5.13) and Eq. (5.14) in Eq. (5.11)

 (5.15)

at the output is computed as the error times the differentiation of activation function with respect to net input.

Substituting Eq. (5.10) and Eq. (5.15) in Eq. (5.9) we get

 (5.16)

Substituting Eq. (5.16), previous weights and with a known step size in Eq. (5.8) we can obtain the next update of weights.

At the hidden layer, the error rate is calculated as

  (5.17)

The first term which is already computed  (5.17a)

The second term  = (5.17b)

The third term  (5.17c)

Substituting Eq. (5.17a), Eq. (5.17b) and Eq. (5.17c) in  i.e., in Eq. (5.17)

 (5.18)

As there are no desired outputs for the hidden layer,  is computed as the differentiated output at times the sum of products of the  already obtained in the output layer with the corresponding weights  in the same layer. In this way the weights are updated at each and every iteration and thus, minimization of risk function i.e., Eq. (5.7) is achieved.

Thus, the Equation (5.2) can be expressed in detail by Equation (5.19), (5.20).

 (5.19)

 (5.20)

Here  is the output of each node in hidden layer

 is the output of each node in output layer

is the risk or loss function

 is the weight connecting between hidden and output node

 is the weight connecting between input and hidden node

 is the net input at node in the output layer 

 is the net input at node in the hidden layer 

The activation function, *f*(z) used in Equations (5.19) and (5.20) are defined by a nonlinear function given by Equation (5.21). This function is applied to the hidden and output neurons and introduces the nonlinearity into the ANN.

 (5.21)

Where,  is the net input applied to the neuron

e is the base of the natural logarithm whose value is the actual numerical  value that you want to transform.

**5.6 Various optimization techniques**

Optimization means finding the minimum or maximum value. In mathematical terms, optimization means finding the point where the derivative is zero. Optimization seeks to improve the performance of design variable to reach optimal point of an objective function. There are two distinct ways of optimization. In case of constrained optimization, the constraints represent some functional relationships among the design variables and other design parameters satisfying certain physical phenomenon and certain resource limitations. The nature and number of constraints to be included in the formulation depend on the user. Constraints may or may not have exact mathematical expressions. In case of unconstrained optimization there is no relationship between the design variables and design parameters, so the design variables can be optimized without any restriction. The various optimization techniques that can be used to optimize the ANNs have been described in following paragraphs.

**5.6.1 Steepest Descent or Gradient descent algorithm**

Steepest descent or Gradient descent is an iterative optimization algorithm to find a local minimum of an objective function. The method relies on first-order derivative of the objective function. The basic approach of this method to find the local minimum is to obtain the direction of descent i.e. gradient and then taking steps proportional to the negative of the gradient of the function at the current point and this process repeats until the minimum of the function is obtained. So this procedure is known as gradient descent also known as steepest descent (Alan R. et.al, 2013). The local maximum of the function can also be obtained if the steps are proportional to the positive gradient of the function. This procedure is known as Gradient ascent.

Let us define a function  which is differentiable in a neighborhood at a point, then decreases if it moves away from w in the direction of negative gradient of. The term  is subtracted from, as it moves against the gradient, down to the minimum. The algorithm to implement gradient descent is as follows.

1. Start with an initial guess  which further converges to the minimum value and initialize the iteration counter to zero and convergence value.
2. Find the gradient of  i.e.which is called direction of descent. In case of multivariable function Jacobin matrix is calculated.
3. Initialize a fixed step size which gives the length of jump to the next value. The step size may be adjusted at each and every iteration using the line search methods to obtain faster convergence.
4. To find the next value of the design variable such that. Subtract the gradient with the previous value as  i.e., 
5. Increment the iteration counter. Test for convergence as 

If the test satisfies the condition then stop the process else go to step ii.

Repeating this process continuously the function reaches to a local minimum point, at which either of neighborhood of function has greater value than the function at the local minimum point. The gradient descent method is simple, computationally less expensive and useful when the information is of higher order derivatives of the objective function i.e., Hessian matrix is not available. This method takes more number of steps to reach the local optimum value.

**5.6.2 Conjugate Gradient descent algorithm**

This is an iterative method and is similar to gradient descent method. In gradient descent method only the negative gradient of the objective function is used to calculate the local optimum value. Even though it is a simple method, it takes more number of iterations to get the optimum value. In conjugate gradient descent method the conjugate of the vector is also used along with the gradient of the function.

1. Start with an initial guess  which further converges to the minimum value and initialize the iteration counter to zero and convergence value.
2. Find the gradient of  i.e., which is called direction of descent. In case of multivariable function Jacobin matrix is calculated.
3. The first conjugate vector is considered in the opposite direction of the gradient and makes the step. During the next iterations, the directions have to be orthogonal to the previous conjugate vector and gradient.

The main advantage of this method is that it requires only last search direction and last gradient instead of full matrix. This method is used to solve problems with n number of variables in n iterations (Edwin.K.P.Chong and Stanishlaw H.Zak, 2011).

**5.6.3. Gauss Newton algorithm**

Gauss Newton is a first derivative optimization algorithm in which it minimizes the function by taking negative gradient of the function. This algorithm is similar to LMS algorithm. This algorithm is also used to solve nonlinear least squares problems. Using this algorithm, the change in value at each step is calculated as follows

1. Start with an initial guess  which further converges to the minimum value and initialize the iteration counter to zero and convergence value.
2. Find the gradient of  i.e.,which is called direction of descent. In case of multivariable function Jacobin matrix is calculated.
3. Initialize a fixed step size which gives the length of jump to the next value. The step size may be adjusted at each and every iteration using the line search methods to obtain faster convergence.
4. Error is calculated as the difference between the observed and the estimated value i.e .
5. To find the next value of the design variable such that. Subtract the gradient with the previous value as  i.e., 
6. Increment the iteration counter. Test for convergence as 

If the test satisfies the condition then stop the process else go to step ii.

Repeating this process continuously the function reaches to a local minimum point. The advantage of this method is the second order derivatives which are complex to compute are not required. However, Gauss Newton is not widely used in practice since it doesn't always converge reliably. Gauss Newton is notable for its fast convergence close to the solution, but like Newton's method, its efficiency depends on having an accurate initial guess.

**5.6.4 Newton Raphson algorithm**

This is an iterative procedure which uses the first and second derivative of the objective function to locate the minimum value. This method is deduced from Taylor series expansion. In this method a starting point is initially and then the quadratic approximation to the objective function is obtained that matches the first and second derivative values at that point. We then minimize the approximate function (quadratic function) instead of the original objective function. The minimizer of the approximate function is used as the starting point in the next step and the procedure is repeated iteratively (Edwin.K.P.Chong and Stanishlaw H.Zak, 2011). The steps of the algorithm are as follows.

1. Start with an initial guess  which further converges to the minimum value and initialize the iteration counter to zero and convergence value.
2. Find the first order and second order derivative of  which gives the direction of descent. In case of multivariable function Jacobin and Hessian matrices are calculated.
3. Initialize a fixed step size which gives the length of jump to the next value. The step size may be adjusted at each iteration using the line search methods to obtain faster convergence.
4. Find the next value of the design variable such that.
5. Subtract the gradient with the guess value as

 i.e., 

1. Increment the iteration counter. Test for convergence as 
2. If the test satisfies the condition then stop the process else go to step ii.

Newton’s method relies on second order derivatives i.e., Hessian matrix which contains much more information about the objective function unlike the gradient descent method which relies only on first order derivatives of the objective function, this makes the Newton’s method to be more efficient in finding the optimum value with required accuracy and it also converges faster by taking less number of iterations than the gradient descent method. If the step size is too small the objective function slowly reduces to minimum and if the step size is too large then the minimum of the function may be skipped. The step size is to be adjusted properly in such a way that it speeds up the iteration process without missing the local minimum value.

**5.6.5 Levenberg-Marquardt algorithm**

Newton’s method and conjugate gradient methods converge to local optimum in fewer iterations but the cost of each iteration is very high because of Hessian matrix. Therefore an alternative method is required to be implemented in order to overcome the disadvantage of the above methods.

The Levenberg-Marquardt (LM) algorithm is an iterative technique that locates the minimum of a multivariate function. It is a standard technique for nonlinear optimization to solve non-linear least-squares problems. The primary application of LM algorithm is least squares curve fitting problem (Manolis I. A. Lourakis, 2005). LM is a blending of steepest descent and the Gauss-Newton method and uses only first order derivative of the objective function without using the second order derivatives. In the gradient descent method, the sum of the squared errors is reduced by updating the parameters in the steepest-descent direction. In the Gauss-Newton method, the sum of the squared errors is reduced by assuming that the least squares function is locally quadratic, and finding the minimum of the quadratic. This technique uses a blending factor which determines the shift between of the two methods. When the current solution is far from the correct one, the algorithm behaves like a steepest descent method. The slow, but guaranteed to converge update equation is. When the current solution is close to the correct solution, it becomes a Gauss-Newton method and the update equation is . We can use steepest descent type method until it reaches to a minimum value, then gradually switch to the quadratic rule (Richard et.al., 2011). The steps are as mentioned below.

1. Update LM algorithm equation as.

Here is the approximation of the Hessian matrix given as , J is the Jacobian matrix, λ is the damping parameter (adaptive balance between the 2 steps), I is the identity matrix and is the direction of descent i.e., gradient of the objective function.

1. Evaluate the error at the new design variable.
2. If the error has increased as a result the update, then retract the step (i.e. reset the value of to their previous values) and increase  by a factor of 10 or some such significant factor. Then go to (i) and try an update again.
3. If the error has decreased as a result of the update, then accept the step (i.e. keep the value of to their new values) and decrease  by a factor of 10 or so.

The intuition is that if the error is increasing, it implies that quadratic approximation is not working well and we are not likely to be a minimum, so we should increase  in order to blend more towards simple gradient descent. Conversely, if error is decreasing, our approximation is working well, which implies that we are getting closer to a minimum so  is decreased to bank more on the Hessian.

The LM algorithm is more robust than the Gauss Newton algorithm (GNA), which means that in many cases it finds a solution even if it starts from value having large offset from the final minimum unlike GNA which performs well only if the initial guess is close to the correct value. For well-behaved functions and reasonable starting parameters, the LM algorithm tends to be a bit slower than the GNA. LM algorithm can also be viewed as Gauss–Newton using a trust region approach.

**5.6.6 Genetic Algorithm**

Genetic Algorithm (GA) is a stochastic global search optimization method associated with high probability to reach a global maxima or minima solution where as in case of other methods they stuck to the local maxima or minima. The main application of GA is to search global maxima and minima for any complex problems like discrete, non-differentiable or the function with any other features. GA is more powerful and efficient than the existing algorithms even though they do not require any derivative information about the objective function. They operate on design variable instead of objective function and start with an initial population which may generated random or seeded by other heuristics. Then select parents from this population for mating. Then apply crossover and mutation operators on the parents to generate new off springs. And finally these off springs replace the existing individuals in the population and the process repeats. In this way genetic algorithm tries to mimic the human evolution to some extent.

Some of the advantages of GA (D.E.Goldberg, 1989) which made them to be popular algorithm are:

1. GAs work with a coding of the parameter set, not with the parameters themselves.
2. GAs search from a population of points and not from the single point.
3. GAs use objective information and they do not require any derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules and not deterministic rules.

The step wise procedure to implement the GA is given below.

1. Start the process by initializing the population of strings at random and then they are concatenated to form chromosome. The size of the population depends on the length of the string. GA can solve up to 10 individual variables.
2. The objective function i.e., fitness function is evaluated and the fitness function value is tested with each chromosome in the population to find out about how well each chromosome can better solve the problem.
3. Reproduction is a step in which selection of chromosomes is done from the current population. It forms a new population with the same total number of chromosomes. Higher fitness value implies high probability of selection of chromosome for the next generation.
4. Reproduction selects good chromosomes and ranks the chromosomes from highest fitness value to lowest fitness value, but does not create new chromosomes. A crossover operator swaps two chromosomes to create two new better offspring chromosomes. The total number of newly generated offspring should be equal to the initial population.
5. Mutation flips the bits from 0 to 1 or vice versa depending on the probability value. This helps the algorithm by maintaining diversity in the population which leads to convergence of GA to reach the global optimum instead of sticking at the local optimum.
6. Go to step ii and evaluate the newly generated population. Repeat the process until the chromosomes best fits with the fitness value and does not change for next generations (Godfrey and Babu, 2004).

**5.7 Conclusions**

ANN is a technique for solving problems by constructing software that works like human brain. It is composed of many artificial neurons (simple processing units) that are linked together according to specific network architecture. The artificial neurons are simplified models of their biological counterparts. The objective of the ANN is to transform the inputs into meaningful outputs. From technical and biological viewpoint, there are two basic reasons why the realtime world is interested in building ANNs: i) Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing and ii) ANNs can be used to replicate and simulate components of the human brain, thereby giving an insight into natural information processing. ANN predicts the unknown random variable from the experimental data that are statistically related to the hidden variables of interest. The important features of ANN which makes it be popularly used are: i) it can compute any computablefunction, by the appropriate selection of the network topology and weights; ii) learn from experience specifically, by trial‐and‐error. The main factors that affect the performance of ANNs are: selection of transfer function, size of training sample, network topology, and weights adjusting algorithm (learning or optimisation techniques). In this chapter, a three layer feed forward network along with the weight updation using back propagation technique has been presented and derivations of various learning algorithms (Gradient descent, Conjugate gradient, Gauss Newton, Newton Raphson, Levenberg-Marquardt, Genetic) and applications to ANN have been are presented in detail. These learning algorithms have subsequently been used in chapter 6 for developing and implementing various ANN models for predicting the strength characteristics of SIFCON.