CHAPTER 6
MPPT OF PV MODULE BY ADVANCED METHODS

6.1 FUZZY LOGIC BASED MPPT METHOD

Microcontrollers have made usage of fuzzy logic control popular for MPPT. Fuzzy logic controllers have the advantages of working with imprecise inputs, not needing an accurate mathematical model and handling nonlinearities. Fuzzy logic control generally consists of three stages: fuzzification, rule base table lookup, and defuzzification. During fuzzification, numerical input variables are converted into linguistic variables based on a membership function similar to Fig 6.1. Fuzzy levels used are of the type: NB (Negative Big), NS (Negative Small), ZE (Zero), PS (Positive Small), and PB (Positive Big). In Fig 6.1, a and b are based on the range of values of the numerical variable.

Fig: 6.1 Membership function for input and output of FLC

The inputs to a MPPT fuzzy logic controller (FLC) are usually an error \( E \) and a change in error \( \Delta E \). The user has the flexibility of choosing how to compute \( E \) and \( \Delta E \). Since \( dP/dV \) vanishes at the MPP, the following approximation is used.

\[
E(n) = \frac{P(n) - P(n-1)}{V(n) - V(n-1)} \quad \ldots \quad (6.1)
\]

\[
\Delta E(n) = E(n) - E(n-1) \quad \ldots \quad (6.2)
\]

Once \( E \) and \( \Delta E \) are calculated and converted to the linguistic variables, the fuzzy logic controller output, which is typically a change in duty ratio \( \Delta D \) of the power converter, is looked up in a rule base table such as Table 6.1. The linguistic variables
assigned to $\Delta D$ for the different combinations of $E$ and $\Delta E$ are based on the knowledge of the user. If for example, the operating point is far to the left of the MPP, that is $E$ is PB, and $\Delta E$ is ZE, then the duty ratio is largely increased, that is $\Delta E$ is PB to reach the MPP.

In the defuzzification stage, the fuzzy logic controller output is converted from a linguistic variable to a numerical variable still using a membership function as in Fig 6.1. This provides an analog signal that will control the power converter to the MPP. MPPT fuzzy logic controllers have shown to perform well under varying atmospheric conditions. However, their effectiveness depends on the knowledge of the user or control engineer in choosing the right error computation and coming up with the rule base table.

Table 6.1: Fuzzy rule base table

<table>
<thead>
<tr>
<th>$\Delta E$</th>
<th>NB</th>
<th>NS</th>
<th>ZE</th>
<th>PS</th>
<th>PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NB</td>
<td>ZE</td>
<td>ZE</td>
<td>NB</td>
<td>NB</td>
<td>NB</td>
</tr>
<tr>
<td>NS</td>
<td>ZE</td>
<td>ZE</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ZE</td>
<td>NS</td>
<td>ZE</td>
<td>ZE</td>
<td>ZE</td>
<td>PS</td>
</tr>
<tr>
<td>PS</td>
<td>PS</td>
<td>PS</td>
<td>PS</td>
<td>ZE</td>
<td>ZE</td>
</tr>
<tr>
<td>PB</td>
<td>PB</td>
<td>PB</td>
<td>PB</td>
<td>ZE</td>
<td>ZE</td>
</tr>
</tbody>
</table>

6.1.1 Implementation

To overcome some of the disadvantages mentioned in previous MPPT methods, fuzzy logic controller is used for maximum power tracking of the PV Panel. The main difference from the previous methods is that the accurate description of the system to be controlled is not required. Fuzzy logic allows the determination of the rule base by linguistic terms and therefore, the tuning of the controller is a very simple way which
is qualitatively different from conventional design techniques. Furthermore, fuzzy control is nonlinear and adaptive in nature, which gives it robust performance under parameter variation, load and supply voltage disturbances.

Fuzzy logic is increasingly used in present times as a convenient tool to model control systems which are nonlinear in nature, like the solar PV array to track the maximum power. The control inputs to the fuzzy logic controller are error and change of error, while the output is the change of control signal for the pulse width-modulation generator. A pulse width modulator is used to supply a pulse train to the switching MOSFET. The MOSFET is biased into a conducting state when the pulse voltage is high and into a non-conducting state when the pulse voltage is low.

The duty cycle of the pulse determines the effective impedance seen by the solar cell. Thus by simply adjusting the duty cycle of the switch, the current flow to the battery may be controlled. A suitable charge controller to achieve the control described above is a dc-to-dc controller with pulse width modulation control. A buck type converter, which steps down cell voltage to a lower value, is shown in Fig 6.2.
The inputs to the fuzzy controller are change in PV array power ($\Delta P_{pv}$) and change in PV array voltage ($\Delta V_{pv}$) corresponding to the two sampling time instants. The two inputs are processed by the fuzzy controller and the output of the fuzzy controller is the incremental reference voltage ($\Delta V_{ref}$), which varies in magnitude and polarity depending on which region of the $I_{pv} \times V_{pv}$ curve, the system is operating on. This output is given to PWM generator, which outputs the reference voltage to the buck converter. The fuzzy based scheme used outputs an incremental reference voltage of appropriate polarity and variable magnitude. Thus during transient conditions the fuzzy logic controller outputs a larger incremental reference voltage to speed up the transient response but outputs almost zero incremental reference voltage near the peak power region to reduce oscillations about the MPP.

### 6.1.2 FLC Description And Design

Fuzzy model of the system is designed based on prior expert knowledge of the system. The fuzzy logic controller is divided into four sections: Fuzzification, rule-base, inference and defuzzification. The inputs to the fuzzy logic controller are change in PV array power ($\Delta P_{pv}$) and change in PV array voltage ($\Delta V_{pv}$) and the output is the change in reference voltage ($\Delta V_{ref}$).

![Block diagram of the FLC](Image)

Fig: 6.3 Block diagram of the FLC
6.1.2.1 Fuzzification

The fuzzy model is developed on a trial-and-error basis to meet the desired performance criteria.

![Membership functions for the fuzzy model](image)

- **a) Variation of power ($\Delta P_{pv}$)**
- **b) Variation of voltage ($\Delta V_{pv}$)**
- **c) Variation of reference voltage ($\Delta V_{ref}$)**

Fig: 6.4 Membership functions for the fuzzy model (a) input $\Delta P_{pv}$, (b) Input $\Delta V_{pv}$ (c) output $\Delta V_{ref}$

The universe of discourse for input variable 1 ($\Delta P_{pv}$) is divided into seven Fuzzy sets: PL (Positive Large), PM (Positive Medium), PS (Positive Small), Z (Zero), NS (Negative Small), NM (Negative Medium) and NL (Negative Large). In the present
work, the Fuzzy set PS assumes a membership value greater than zero beginning at
the origin, in order to speed up the start-up process and at the same time prevent
variation of the reference voltage at the MPP. Additional Fuzzy sets PM and NM have
been added to improve the control surface.

The universe of discourse for input variable 2 ($\Delta V_{pv}$) is divided into 7 fuzzy sets:
PL (Positive Large), PM (Positive Medium), PS (Positive Small), Z (Zero), NS
(Negative Small), NM (Negative Medium) and NL (Negative Large). The universe of
discourse for the output variable ($\Delta V_{ref}$) is divided into 7 fuzzy sets: PL (Positive
Large), PM (Positive Medium), PS (Positive Small), Z (Zero), NL (Negative Large),
NM (Negative Medium) and NS (Negative Small). The membership functions for the
input and output variables are shown in Fig 6.4.

The membership functions for the input and output variables are designed to model
the unsymmetrical nature of the PV panel $I_{pv} \times V_{pv}$ curve. The membership functions
are denser at the center to provide greater sensitivity in the region near the MPP. Input
membership functions are normalized and suitable tuning gains are used to match the
inputs to the respective universes of discourse.

6.1.2.2 Rule Base

<table>
<thead>
<tr>
<th>$\Delta V_{pv}/ \Delta P_{pv}$</th>
<th>NL</th>
<th>NM</th>
<th>NS</th>
<th>Z</th>
<th>PS</th>
<th>PM</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>PL</td>
<td>PL</td>
<td>PL</td>
<td>PL</td>
<td>NM</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>NM</td>
<td>PL</td>
<td>PL</td>
<td>PL</td>
<td>PM</td>
<td>PS</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>NS</td>
<td>PL</td>
<td>PM</td>
<td>PS</td>
<td>PS</td>
<td>PS</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>PL</td>
<td>PM</td>
<td>PS</td>
<td>Z</td>
<td>NS</td>
<td>NM</td>
<td>NL</td>
</tr>
<tr>
<td>PS</td>
<td>Z</td>
<td>Z</td>
<td>NM</td>
<td>NS</td>
<td>NS</td>
<td>NM</td>
<td>NL</td>
</tr>
<tr>
<td>PM</td>
<td>Z</td>
<td>Z</td>
<td>NS</td>
<td>NM</td>
<td>NL</td>
<td>NL</td>
<td>NL</td>
</tr>
<tr>
<td>PL</td>
<td>Z</td>
<td>Z</td>
<td>NM</td>
<td>NL</td>
<td>NL</td>
<td>NL</td>
<td>NL</td>
</tr>
</tbody>
</table>
The Fuzzy algorithm tracks the maximum power based on the master-rule: If the last change in the reference voltage \( V_{ref} \) has caused the power to increase keep changing the reference voltage in the same direction; else if it has caused the power to drop, move it in the opposite direction. A rule base consisting of 49 rules is designed as shown in Table 6.2.

### 6.1.2.3 Inference Method

The Inference method determines the output of the fuzzy controller. Mamdani’s inference method is used in the considered system along with the max-min composition method. This is because this method is computationally more efficient and has better interpolative properties than methods based on other implication functions. Hence, Mamdani’s inference method is usually popular for most control engineering applications.

### 6.1.2.4 Defuzzification

The output of the fuzzy controller is a fuzzy set. However a crisp output value is required. Hence the output of the fuzzy controller should be defuzzified. The centroid method is one of the commonly used defuzzification methods and is the one being employed for the proposed system. This method has good averaging properties and simulation results have shown that it provides the best results.

### 6.1.3 Simulation And Results

Simulink model of fuzzy logic based maximum power tracking for 60W PV panel is shown in Fig 6.5. Simulation is run in Matlab/Simulink environment to verify the performance of the proposed scheme with simplified model of a PV panel, buck converter and a resistive load.

The goal of designed FLC is to track maximum power irrespective of panel voltage variations. Consequently FLC uses two input variables: change in PV array Power
(ΔP_{pv}) and change in PV array voltage (ΔV_{pv}) corresponding to the two sampling time instants. Equations (6.3), (6.4), (6.5) determine required system equations. The output variable is the change in control variable (ΔV_{ref}). (ΔV_{ref}) is integrated to achieve desired V_{ref} value. Here V_{ref} is the reference voltage determining duty cycle of DC-DC converter designed in this proposed scheme.

\[ ΔP_{pv} = [P_{pv}(k) - P_{pv}(k-1)] * K1 \quad \text{.... (6.3)} \]

\[ ΔV_{pv} = [V_{pv}(k) - V_{pv}(k-1)] * K2 \quad \text{.... (6.4)} \]

\[ ΔV_{ref} = [V_{ref}(k) - V_{ref}(k-1)] * K3 \quad \text{.... (6.5)} \]

Here K1, K2, K3 are gain coefficients and k is a time index.

To calculate FLC output value, the inputs and outputs are converted from crisp values into linguistic form. Fuzzy membership functions are used to perform this conversion. Here, all membership functions are defined between -1 and 1 interval by means of input scaling factors K1 and K2 and the output scaling factor K3. Thus, since simple numbers are now processed in controller after scaling, fuzzy computation is performed in a shorter time. The linguistic terms for input and output values are represented by seven membership functions as shown in Fig 6.4.

The two inputs (ΔP_{pv}) and (ΔV_{pv}) are processed by the FLC and outputs a control variable (ΔV_{ref}) based on the control rules as shown in Table 6.2. The fuzzy output is given to the PWM generator which outputs the reference voltage to the buck converter. Fundamentally, the operating principle of PWM generator is based on the comparison of two signals. One of the signals is a triangular waveform and the other one is fixed linear signal, which represents time equivalent of triggering voltage. Consequently, reference voltage time signal and triangular signal are U_1 and U_2 variables of ‘IF’ block used in simulation model as shown in Fig 6.5.
The output of the pulse width modulator is used to supply a pulse train to the switching MOSFET. The MOSFET is biased into a conducting state when the pulse voltage is high and into a non-conducting state when the pulse voltage is low. Thus the fuzzy algorithm tracks the maximum power based on the defined rules. The simulation results are shown below.

![FLC simulink model](image)

Fig: 6.5 FLC simulink model

![PV source block](image)

Fig: 6.6 PV source block
Fuzzy logic controller model is simulated in Matlab/Simulink environment to track the maximum power point and the value of the maximum power tracked is 59.9 W.

6.2 NEURAL NETWORKS FOR MPP TRACKING

The human brain mainly inspires artificial neural networks. This doesn't mean that Artificial Neural Networks are exact simulations of the biological neural networks inside our brain because the actual working of human brain is still a mystery. Neural Network is a machine that is designed to model the way to which the brain performs a particular task. The network is implemented using electronic components or is simulated in software on digital computer. Neural networks perform usual computations through process of learning.
6.2.1 Background Of Neural Networks

Neural network is a massively parallel distributed processor made up of simple processing units, which has a natural property of storing experimental knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network from its environment through learning process.
- The interneuron connection strengths, known as synaptic weights are used to store the acquired knowledge.

The primary significance of the neural network is the ability of the network to learn from its environments and to improve its performance through learning. It learns about its environment through an interactive process of adjustments applied to its synaptic weights and biases. The network becomes more knowledgeable about its environment after each iteration of learning process.

The definition of learning process implies the sequence of events:

- The neural network is stimulated by an environment.
- Neural network undergoes changes in its free parameters as results of this stimulation.
- Neural network responds in a new way to the environment because of the changes that have occurred in its internal structure.

6.2.1.1 Neural Network Applications

- Neural networks are information processing systems. In general, neural network can be thought of a “black box” device that accepts input and produces output. Some of the operations that neural network perform include
  - Classification- an input is passed to the network and the network produce a representative class as output.
Pattern matching- an input pattern is passed to the network and the network produces the corresponding output pattern.

Noise removal- a noise-corrupted input pattern is passed to the network and the network removes some or all of the noise and produces a cleaner version of the input pattern as output.

Optimization- an input pattern representing the initial values for a specific optimization problem is presented to the network and the network produces a set of variables that represents a solution to the problem.

Control- an input pattern represents the current state of a controller and the desired response for the controller and the output is the proper command sequence that will create the desired response.

6.2.1.2 Biological Neuron

Brain contains about $10^{10}$ basic units called neurons. Each neuron is connected to about $10^4$ other neurons. A neuron is a small cell that receives electrochemical signals from various sources and in turn responds by transmitting electrical impulses to other neurons. Artificial neural network draw much of their inspiration from biological nervous system. Most living creatures, which have the ability to adapt to a changing environment, need a controlling unit that is able to learn.

![Biological Neuron Architecture](image-url)
Humans use complex networks of highly specialized neurons to perform this task.

The control unit is divided into different anatomic and functional sub-units, each having certain tasks like vision, hearing, motor and sensor control. Fig 6.9 shows the biological neuron architecture.

The four basic components of a biological neuron are described below:

- **Dendrites** - Dendrites are hair-like extensions of a neuron and each dendrite brings some input to the neuron in the form of electrical signal and (from neurons in the previous layer) these inputs are given to the cell body.

- **Cell body** - Cell body is responsible for processing these inputs and the output is provided to other neurons through the axon and synapses.

- **Axon** - Axon is responsible for carrying the output of cell body to other neurons, through the synapses.

- **Synapses** - The point of contact between the axon of one cell and dendrite of another cell is called synapses.

The connections between neurons are possible because of synapses and dendrites.

### 6.2.1.3 Designing of Neural Network Controller

![Model of neuron](image)
A neuron is a information processing unit that is basic unit to the operation of neural network. The block diagram given above shows the model of a neuron.

An artificial neuron consists of various inputs, much like the biological neuron. Three basic elements of the neuron model are identified, as described below:

A set of synapses of connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{kj}$. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers. The weight $w_{kj}$ is positive if the associated synapse is excitatory; it is negative if the synapse is inhibitory.

Summation unit: When inputs are fed to the neuron, the summation unit will initially find the Net-value. The Net value is the product of input value and corresponding connection weight. i.e, input value $x(i)$ of each input to the neuron is multiplied with the associated connection weight $w(i)$. In simplest case, these products are summed and fed to the transfer function. Also, a neuron has a bias value, which affects the net value. A bias of a neuron is set to a random value, when the network is initialized. The connection weights and bias of all neurons in the network is changed (other than neurons in the input layer) during training phase.

The model of a neuron also includes an externally applied threshold $\theta_k$ that has the effect of lowering the net input of the activation function. On the other hand the net input of the activation function is increased, by employing a bias term rather than a threshold. The bias is the negative of the threshold.

In mathematical terms, a neuron $k$ may be described by following equations:
\( u_k = \sum_{j=1}^{p} w_{kj} x_j \) \hspace{1cm} \text{.... (6.6)}

\( y_k = \varphi(u_k - \theta_k) \) \hspace{1cm} \text{.... (6.7)}

where, \( x_1, x_2, \ldots, x_p \) are the input signals; \( w_{k1}, w_{k2}, \ldots, w_{kp} \) are the synaptic weights of neuron \( k \); \( u_k \) is the linear combiner output; \( \theta_k \) is the threshold; \( \varphi(.) \) is the activation function and \( y_k \) is the output signal of the neuron. The use of the bias \( b_k \) has the effect of applying an affine transformation to the output \( u_k \) of the linear combiner in the model.

\( v_k = u_k + b_k \) \hspace{1cm} \text{.... (6.8)}

Following briefs on the types of activation functions.

An activation function is for limiting the amplitude of the output of a neuron. The activation is a linear or non-linear function. A particular activation is chosen to satisfy some specification of the problem that the neuron is attempting to solve. The activation function is denoted by \( \varphi(.) \). The most commonly used activation functions are Simple thresholding (hard limiting), Squashing function (sigmoid), Gaussian function and Linear function

- **Simple thresholding (Hard limiting)**

The hard limit transfer function sets the output of the neuron to 0 if the function argument is less than 0 and to 1 if its argument is greater than or equal to 0.

Typically, it can be represented by

\[
f(x) = \begin{cases} 
1 & \text{if } x \geq 0 \\
0 & \text{if } x < 0 
\end{cases} \hspace{1cm} \text{.... (6.9)}
\]

\[
f(x) = \begin{cases} 
1 & \text{if } x \geq 0 \\
0 & \text{if } x < 0 
\end{cases} \hspace{1cm} \text{.... (6.10)}
\]

The output of the this activation function is

\( y_k = \text{hardlim}(v_k) \)
- Squashing function (Sigmoid) This transfer function takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 to 1, according to the expression:

\[ y_k = \frac{1}{1+e^{-av_k}} \quad \ldots (6.11) \]

where a is the slope parameter of the sigmoid function.

The log- sigmoid activation function is commonly used in multilayer networks that are trained using the back propagation algorithm.

- Gaussian function

The Gaussian function is a radial function that requires a variance value \( v > 0 \) to shape the Gaussian function. In some networks the Gaussian function is used in conjunction with a duel set of connections and in other instances the variance is predefined. In the later instance, the activation function is

\[ f(x) = \exp(-x^2/v) \quad \ldots (6.12) \]

where \( x \) is the mean and \( v \) is the predefined variance.

- Linear function

The output of a linear activation function equals to its input: \( y_k = v_k \)

Neurons with this activation function are used in the ADALINE networks.

Fig: 6.11 Different types of activation functions: (a) Threshold
(b) Piecewise linear (c) Sigmoid (d) Gaussian
Learning rules: The procedure used to perform the learning process is known as learning algorithm. Its main function is to modify the synaptic weights of the network in an orderly fashion to attain desired design objective.

Learning methods in neural networks are broadly classified into three types. Supervised learning, Unsupervised learning and Reinforced learning.

- Supervised learning

In supervised learning the learning rule is provided with a training set of proper network behavior:

\{P_1, t_1\}, \{P_2, t_2\}... \{P_Q, t_Q\},

Where \(P_q\) is an input to the network and \(t_q\) is the corresponding correct target output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets. The perceptron learning rule falls in this supervised learning category.

- Unsupervised learning

In unsupervised learning, the weights and biases are modified in response to network inputs only. There are no target outputs available. Most of these algorithms perform some kind of clustering operation. They learn to categorize the input patterns into a finite number of classes. This is especially useful in applications such as vector quantization.

- Reinforced learning

Reinforcement learning is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade is a measure of the network performance over some
sequence of inputs. This type of learning is currently much less common than supervised learning. It appears to be most suited to control system applications.

### 6.2.1.4 Summary of ANN algorithms

Table 6.3 shows the different algorithms under supervised learning paradigms, Table 6.4 shows the different algorithms under unsupervised learning paradigms and hybrid paradigms.

**Table 6.3: Different algorithms under supervised paradigms**

<table>
<thead>
<tr>
<th>Learning paradigm</th>
<th>Learning rule</th>
<th>Architecture</th>
<th>Learning algorithm</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>Error correction</td>
<td>Single-or multi-layer Perceptron</td>
<td>Perceptron learning algorithm, Backpropagation, adaline &amp; madaline</td>
<td>Pattern classification, function approximation, control</td>
</tr>
<tr>
<td></td>
<td>Boltsmann</td>
<td>Recurrent</td>
<td>Boltsmann Learning Algorithm</td>
<td>Pattern classification</td>
</tr>
<tr>
<td></td>
<td>Hebbians</td>
<td>Multilayer feedforward</td>
<td>Linear Discriminant Analysis</td>
<td>Data analysis, Pattern classification</td>
</tr>
<tr>
<td></td>
<td>competitive</td>
<td>Competitive</td>
<td>Learning Vector Quantisation</td>
<td>Within-class categorization, Data compression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ART network</td>
<td>ARTMAP</td>
<td>Pattern Classification, Within-class categorisation</td>
</tr>
<tr>
<td>Learning paradigm</td>
<td>Learning rule</td>
<td>Architecture</td>
<td>Learning algorithm</td>
<td>Task</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>Error correction</td>
<td>multi-layer feedforward</td>
<td>Samson’s projection</td>
<td>DATA analysis</td>
</tr>
<tr>
<td></td>
<td>Hebbians</td>
<td>feedforward or Competitive</td>
<td>Principal component analysis</td>
<td>Data analysis and Data Compression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hope field net</td>
<td>Associate memory learning</td>
<td>Associate memory</td>
</tr>
<tr>
<td>Competitive</td>
<td>Competitive</td>
<td>Vector Quantization</td>
<td>Categorization Data compression</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kohonen SOM</td>
<td>Kohonen SOM</td>
<td>Categorization Data analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ART network</td>
<td>ART1,ART2</td>
<td>Categorization</td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td>Error correction and competitive</td>
<td>RBF network</td>
<td>RBF Learning algorithm</td>
<td>Pattern Classification Function approximation control</td>
</tr>
</tbody>
</table>
6.2.1.5 Multilayer Feedforward Networks

A multilayer feedforward network consists of a set of sensory units (Source nodes) that constitute the input layer, one or more hidden layers of computation nodes and an output layer. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These neural networks are commonly referred to as multilayer perceptrons. The function of the hidden neurons is to intervene between the external input and the network output. By adding one or more hidden layers, the network is enabled to extract higher-order statistics, for the network acquires a global perspective despite its local connectivity by virtue of the extra set of synaptic connections and the extra dimensions of neural interactions. The ability of hidden neurons to extract higher-order statistics is particularly valuable when the size of the input layer is large.

The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computational nodes) in the second layer (i.e. the first hidden layer). The output signals of the second layer are used as input to the third layer, and so on for the rest of the network. Typically, the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer.

The architectural graph of Fig 6.12 illustrates the lay out of a typical multilayer feed forward neural network. The network of Fig 6.12 is referred to as 3-6-3 network in that it has 3 source nodes, 6 hidden neurons and 3 output neurons. The neural
network of Fig.6.12 is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer.

Fig: 6.12 Fully connected multilayer feedforward artificial neural network with input layer, one hidden layer and output layer

### 6.2.1.6 Back-Propagation Algorithm

To ease the understanding of the algorithm, a summary of notations used in the back-propagation training algorithm is given below:

The indices i, j and k refer to different neurons in the network; with signals propagating through the network from left to right, neuron j lies in a layer to the right of neuron i, and neuron k lies in a layer to the right of neuron j when neuron j is a hidden unit.

The iteration n refers to the nth training pattern presented to the network. E(n) refer to the instantaneous sum of error squares at iteration n. The average of E(n) refers to
the instantaneous sum of error squares at iteration \( n \). The average of \( E(n) \) over all values of \( n \) (i.e. the entire training set) yields the average squared error \( E_{av} \).

- \( e_j(n) \) refers to the error signal at the output neuron \( j \) for iteration \( n \).
- \( d_j(n) \) refers to the desired response for neuron \( j \), is used to compute \( e_j(n) \).
- \( y_j(n) \) refers to the function signal appearing at the output neuron \( j \) at iteration \( n \).
- The symbol \( w_{ji}(n) \) denotes the synaptic weight connecting the output of neuron \( i \) to the input of neuron \( j \) at iteration \( n \). The correction applied to this weight at iteration \( n \) is denoted by \( \Delta w_{ji}(n) \).
- The net internal activity level of neuron \( j \) at iteration \( n \) is denoted by \( v_j(n) \).
- The activation function describing the input-output functional relationship associated with neuron \( j \) is denoted by \( \varphi_j(\cdot) \).
- The threshold applied to neuron \( j \) is denoted by \( \theta_j \); its effect is represented by a synapse of weight \( w_{jo} = \theta_j \) connected to a fixed input equal to \( -1 \).
- The \( i \)th element of the input vector is denoted by \( x_i(n) \).
- The \( k \)th element of the overall output vector is denoted by \( o_k(n) \).
- The learning – rate parameter is denoted by \( \eta \).

Basically, the error back – propagation process consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network and its effect propagates through the network, layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the network are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with the error-correction rule. Specifically, the actual response of the network is subtracted from a desired response to produce an error signal. This error signal is then propagated backward
through the network, against the direction of the synaptic connections – hence the name “error back-propagation”.

The synaptic weights are adjusted so as to make the actual response of the network move closer to the desired response. The learning process performed with the algorithm is called the back-propagation learning.

The error signal at the output of neuron j at iteration n is defined by:

$$E_j(n) = d_j(n) - y_j(n)$$  .... (6.13)

Neuron j is the output node.

The instantaneous value of the squared error for neuron j is defined as $$\frac{1}{2}e_j^2(n)$$. Correspondingly, the instantaneous value E(n) of the sum of squared errors is obtained by summing $$\frac{1}{2}e_j^2(n)$$ over all neurons in the output layer; these are the only “visible” neurons for which error signals are calculated. The instantaneous sum of squared errors of the network is thus written as:

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$  .... (6.14)

Where the set C includes all neurons in the output layer of the network. Let N denote the total number of patterns contained in the training set. The average squared error is obtained by summing E(n) over all n and then normalizing with respect to the set size N as:

$$E_{av} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$  .... (6.15)

The instantaneous sum of error squares E(n), and therefore the average squared error $$E_{av}$$, is a function of all the free parameters of the network. For a given training set, $$E_{av}$$ represents the cost function as the measure of training set learning performance. The objective of the learning process is to adjust the free parameters of
the network so as to minimize $E_{av}$. The arithmetic average of these individual weight changes over the training set and is an estimate of the true change that would result from modifying the weights based on minimizing the cost function, $E_{av}$, over the entire training set.

In a practical application of the back–propagation algorithm, learning results from the many presentations of a prescribed set of training examples to the multilayer perceptron. One complete presentation of the entire training set during the learning process is called an epoch. The learning process is maintained on an epoch-by-epoch basis until the synaptic weights and threshold levels of the network stabilize and the average squared error over the entire training set converges to some minimum value. It is good practice to randomize the order of presentation of training examples from one epoch to the next.

For a given training set, back-propagation learning thus proceeds in one of the following ways:

**Pattern Mode:** In the pattern mode of back–propagation learning, weight updating is performed after the presentation of each training example. Consider an epoch consisting of $N$ training examples arranged in the order $[x(1),d(1),\ldots,[x(N),d(N)]$. The first example $[x(1),d(1)]$ in the epoch is presented to the network, and the sequence of forward and backward computations described previously is performed, resulting in certain adjustments to the synaptic weights and threshold levels of the network. Then, the second example $[x(2),d(2)]$ in the epoch is presented, and the sequence of forward and backward computations is repeated, resulting in further adjustment to the synaptic weights and threshold levels. This process is continued until the last example $[x(N), d(N)]$ in the epoch is accounted for.
Batch Mode: In the batch mode of back-propagation learning, weight updation is performed after the presentation of all the training examples that constitute an epoch.

From an “on-line” operational point of view, the pattern mode of training is preferred over batch mode, because it requires less local storage for each synaptic connection. Moreover, given that the patterns are presented to the network in a random manner, the use of pattern-by-pattern updating of weights makes the search in weight space stochastic in nature, which, in turn, makes it less likely for the back-propagation algorithm to be trapped in a local minimum. On the other hand, the use of batch mode of training provides a more accurate estimate of the gradient vector. In the final analysis, however, the relative effectiveness of the two training modes depends on the problem at hand. For the pattern mode of back-propagation training the algorithm cycles through the training data \([x(n), d(n)]; n = 1, 2, \ldots, N\) is as follows:

Initialization: Start with a reasonable network configuration and set all the synaptic weights and threshold levels of the network to small random numbers that are uniformly distributed.

Presentation of training examples: Present the network with an epoch of training examples. For each example in the set ordered in some fashion, perform the following sequence of forward and backward computations.

Forward computation: Let a training example in the epoch is denoted by \([x(n), d(n)]\), with the input vector \(x(n)\) applied to the layer of sensory nodes and the desired response vector \(d(n)\) presented to the output layer of computation nodes. Compute the activation potentials and function signals of the network by proceeding forward through the network, layer by layer. The net internal activity level \(v_j^{(l)}(n)\) for neuron \(j\) in layer \(l\) is:
\[ y_j^{(l)}(n) = \sum_{i=0}^{p} W_{ji}^{(l)}(n) y_i^{(l-1)}(n) \] .... (6.16)

Where \( y_j^{(l-1)}(n) \) is the function signal of neuron i in the previous layer l-1 at iteration n and \( W_{ji}^{(l)}(n) \) is the synaptic weight of neuron j in layer l that is fed from neuron i in layer l-1. For i=0, we have \( y_0^{(l-1)}(n) = 1 \) and \( w_{j0}^{(l)}(n) = \theta_j^{(l)}(n) \), where \( \theta_j^{(l)}(n) \) is the threshold applied to neuron j in layer l. Assuming the use of a logistic function for the sigmoidal nonlinearity, the function (output) signal of neuron j in layer l is

\[ y_j^{(l)}(n) = \frac{1}{1 + \exp(-y_j^{(l)}(n))} \] .... (6.17)

If neuron j is in the hidden layer, set \( y_j^{(0)}(n) = x_j(n) \)

Where, \( x_j(n) \) is the jth element of input vector \( x(n) \)

If neuron j is in the output layer, set \( y_j^{(L)}(n) = o_j(n) \)

Hence, compute the error signal \( e_j(n) = d_j(n) - o_j(n) \)

Where, \( d_j(n) \) is the jth element of the desired response vector \( d(n) \).

**Backward computation:** The \( \delta \)'s of the network are computed by proceeding backward, layer by layer.

\[ \delta_j^{(L)}(n) = e_j^{(L)}(n) o_j(n)[1-o_j(n)] \]

for neuron j in output layer L

\[ \delta_j^{(l)}(n) = y_j^{(l)}(n)[1-y_j^{(l)}(n)] \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) \] .... (6.18)

for neuron j in hidden layer l.

Hence, adjust the synaptic weights of the network in layer l according to the generalized delta rule:
\[ W^{(i)}_{j}(n+1) = W^{(i)}_{j}(n) + \alpha \left[ W^{(i)}_{j}(n) - W^{(i)}_{j}(n-1) \right] + \eta \delta^{(i)}_{j}(n) y^{(i-1)}_{j}(n) \]  \hspace{1cm} \text{.... (6.19)}

where \( \eta \) is the learning rate parameter and \( \alpha \) is the momentum constant.

**Iteration:** Iterate the computation by presenting new epochs of training examples to the network until the free parameters of the network stabilize their values and the average squared error \( E_{av} \) computed over the entire training set is at a minimum or acceptable small value. The order of presentation of training examples should be randomized from epoch to epoch. The momentum and the learning rate parameter are typically adjusted as the number of training iterations increases.

### 6.2.2 Implementation

A three layered feed forward network is created using `newff` command. The network is created with two neurons in the input layer, 50 neurons in the hidden layer and one neuron in the output layer. The two inputs to the network are the insolation and the temperature. Totally 23 sets of input data samples are given to the network for training. The targets of the network are specified to be the values of voltages that are obtained from modeling for a particular input set among training sets. The activation functions used at the input and output layers are `tansig` and `purelin` functions respectively. The `tansig` activation function calculates its output according to:

\[ \text{tansig}(n) = \frac{2}{1+\exp(-2*n)} - 1 \]  \hspace{1cm} \text{.... (6.20)}

The `purelin` activation function calculates its output according to:

\[ \text{purelin}(n) = n \]  \hspace{1cm} \text{.... (6.21)}

The basic structure of the neural network is shown below in Fig 6.13.
The error criterion that is considered for training is Mean Square Error. The learning function is taken to be trainlm. “Trainlm” is a network training function that updates weight and bias values according to Levenberg-Marquardt (LM) optimization. This is one of the most popularly used algorithms which is a type of numerical optimization technique that has many advantages over its counterparts.

The main advantage of this algorithm is that it requires less number of data for training the network and achieves accurate results. The other advantage of using LM optimization method is that it produces accurate results even if the system is not completely controllable and observable. Also it need not compute the Hessian matrix. When the performance function has the form of a sum of squares, then the Hessian matrix is approximated as

$$H = J^T J \quad \text{....(6.22)}$$

and the gradient is computed as

$$g = J^T e. \quad \text{.... (6.23)}$$

The Jacobian matrix(J) is computed through a standard back-propagation (BP) that is much less complex than computing the Hessian matrix. The LM method uses
this approximation to the Hessian matrix in the following Newton-like update as follows.

\[ x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e. \] .... (6.24)

When the scalar \( \mu \) is zero, this is just Newton's method, using the approximate Hessian matrix. When \( \mu \) is large, this becomes gradient descent with a small step size. Newton's method is faster and more accurate near an error minimum, so the aim is to shift towards Newton's method as quickly as possible. Thus, \( \mu \) is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function will always be reduced at each iteration of the algorithm.

The error BP learning is implemented for updating the weights of the network in order to minimize the mean square error. The BP algorithm consists of two passes.
- Forward pass
- Backward pass

In the forward pass, 23 pairs of insolation and temperature are given to the network. As these are given all at a time, this type of training is called Batch training. These inputs propagate through the network layer-by-layer and the output voltage is generated. These inputs are used for computing the gradient and updating the network weights and biases.

Now the output voltage is compared with the corresponding target value and then the error which is the difference between these two values is propagated through the network in the backward pass. During this process, the weights of the network are updated recursively. The BP algorithm uses the following rule for updating the weights of the network.

\[ w(k + 1) = w(k) - \eta g(k) \] .... (6.25)
Where, $\eta$----learning rate

g(k)----gradient vector

The gradient vector is computed in the backward pass by applying the chain rule. The training parameters for trainlm are epochs, show, goal, time, min_grad, max_fail, mu, mu_dec, mu_inc, mu_max. Training occurs according to the trainlm's training parameters as shown below.

net.trainParam.epochs -----------Maximum number of epochs to train
net.trainParam.goal -----------Performance goal
net.trainParam.max_fail ------- Maximum validation failures
net.trainParam.mem_reduc------Factor used for memory/speed purpose
net.trainParam.min_grad ----- Minimum performance gradient
net.trainParam.mu ----------- Initial Mu
net.trainParam.mu_dec-------- Mu decrease factor
net.trainParam.mu_inc-------- Mu increase factor
net.trainParam.mu_max -------Maximum Mu
net.trainParam.show ----------- Epochs between showing progress
net.trainParam.time ----------- Maximum time to train in seconds.

The performance goal is specified as 0.15. The maximum number of epochs are set to 30. This is done on trial basis. When the epoch number is mentioned to be low than this number, training is stopped without reaching the performance. Maximum validation failures is kept as 5. Mu decrease factor is mentioned as 0.01. The time interval for showing the Epoch’s progress is set to 50. This shows the training curve for every 50 epochs. The other parameters are set to their default values. The default values for these parameters are shown below.

net.trainParam.mu_inc --------10
net.trainParam.mu_max ------1e^-10
net.trainParam.mem_reduc----1
net.trainParam.min_grad ------1e^-25

The parameter mu is the initial value for µ. This value is multiplied by mu_dec whenever the performance function is reduced by a step. It is multiplied by mu_inc whenever a step would increase the performance function. If mu becomes larger than mu_max, the algorithm is stopped. The trained network thus obtained is tested using 9 sets of data.

6.2.3 Algorithm For ANN Based MPPT

Step-1: Construct the network and initialize the synaptic weights with random values.

Step-2: Apply the input sets to the network

Step-3: Set the parameters of the network and calculate the corresponding output values by training the network.

Step-4: Compare the actual outputs with the desired outputs and determine a measure of the error.

Step-5: Determine the amount by which each weight is to be changed and make corrections to each weight.

Step-6: Repeat Step-3 to Step-4 until the error for the sets in the training set is reduced to an acceptable value.

Step-7: Validate the so formed network using testing sets

A neural network simulink block is generated using gensim(net) command. After the network is generated, 9 sets of insolation and temperature are given to the network for validating the network. Each data set is given to the network at different instances of time with an interval of 1 sec. The network is simulated. The curves for voltage, current and power are obtained. The overall simulink model is shown in Fig 6.14 .
Fig: 6.14 Over all simulink model

The structure of neural network is shown below in Fig 6.15. It consists of two layers. The structures of the first and the second layers are shown in Fig 6.16 and Fig 6.17 respectively.

Fig: 6.15  Structure of neural network

Fig: 6.16  Structure of first layer of neural network
6.2.4 Results

The training data used is shown below. These training sets were chosen to cover all the typical input space in order to get good performance where temperature ranges from -40 degrees centigrade to 52 degree centigrade and solar irradiation ranged from 50 to 1000 W/m². The first column shows the various values of insolation, second column corresponds to various temperatures and the third column corresponds to the target voltages.

**Training data:**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50</td>
<td>16.15</td>
</tr>
<tr>
<td>100</td>
<td>-40</td>
<td>7.95</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>11.82</td>
</tr>
<tr>
<td>180</td>
<td>30</td>
<td>15.69</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>13.75</td>
</tr>
<tr>
<td>220</td>
<td>40</td>
<td>17.04</td>
</tr>
<tr>
<td>600</td>
<td>52</td>
<td>19.45</td>
</tr>
<tr>
<td>700</td>
<td>42</td>
<td>18.58</td>
</tr>
<tr>
<td>770</td>
<td>47</td>
<td>19.15</td>
</tr>
<tr>
<td>830</td>
<td>35</td>
<td>18</td>
</tr>
<tr>
<td>850</td>
<td>50</td>
<td>19.68</td>
</tr>
</tbody>
</table>
The results are shown in Table 6.5. Training is completed for 25 epochs and the performance goal is met. The Mean Square error after training is .146852. The training curve is shown in Fig 6.18.

![Training curve](image)

**Fig: 6.18 Training curve**

The trained network is validated with the following 9 sets of insolation and temperature conditions. The corresponding voltages and powers are tabulated in Table 6.5.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Insolation (W/m²)</th>
<th>Temperature (°C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>10</td>
<td>15.41</td>
</tr>
<tr>
<td>910</td>
<td>12</td>
<td>15.6</td>
</tr>
<tr>
<td>920</td>
<td>13</td>
<td>18.17</td>
</tr>
<tr>
<td>950</td>
<td>23.75</td>
<td>16.96</td>
</tr>
<tr>
<td>955</td>
<td>23.87</td>
<td>16.97</td>
</tr>
<tr>
<td>960</td>
<td>24</td>
<td>16.96</td>
</tr>
<tr>
<td>965</td>
<td>24.12</td>
<td>16.96</td>
</tr>
<tr>
<td>970</td>
<td>24.25</td>
<td>17.02</td>
</tr>
<tr>
<td>975</td>
<td>24.37</td>
<td>17.02</td>
</tr>
<tr>
<td>980</td>
<td>24.5</td>
<td>17.08</td>
</tr>
<tr>
<td>985</td>
<td>24.62</td>
<td>17.13</td>
</tr>
<tr>
<td>995</td>
<td>24.87</td>
<td>17.14</td>
</tr>
</tbody>
</table>
Table 6.5: Results of ANN based MPPT method

<table>
<thead>
<tr>
<th>Insolation (W/m²)</th>
<th>Temperature (°C)</th>
<th>Target Voltage obtained from Modeling(V)</th>
<th>Power obtained from Modeling(W)</th>
<th>Voltage obtained from ANN based MPPT Method(V)</th>
<th>Power obtained from ANN based MPPT Method(W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>960</td>
<td>24</td>
<td>16.96</td>
<td>56.84</td>
<td>16.9698</td>
<td>56.8380</td>
</tr>
<tr>
<td>175</td>
<td>5</td>
<td>13.1</td>
<td>7.892</td>
<td>12.04</td>
<td>7.6394</td>
</tr>
<tr>
<td>1000</td>
<td>25</td>
<td>17.1</td>
<td>59.9159</td>
<td>17.0862</td>
<td>59.8885</td>
</tr>
<tr>
<td>700</td>
<td>42</td>
<td>18.52</td>
<td>44.82</td>
<td>19.2</td>
<td>44.8192</td>
</tr>
<tr>
<td>965</td>
<td>24.125</td>
<td>16.96</td>
<td>57.21</td>
<td>16.98</td>
<td>57.2030</td>
</tr>
<tr>
<td>930</td>
<td>12</td>
<td>16.6</td>
<td>48.64</td>
<td>16.5</td>
<td>48.6104</td>
</tr>
<tr>
<td>995</td>
<td>24.87</td>
<td>17.14</td>
<td>59.41</td>
<td>17.0674</td>
<td>59.4013</td>
</tr>
<tr>
<td>820</td>
<td>40</td>
<td>18.46</td>
<td>53.14</td>
<td>18.05</td>
<td>52.9013</td>
</tr>
<tr>
<td>980</td>
<td>24.5</td>
<td>17.08</td>
<td>58.31</td>
<td>17.02</td>
<td>58.3007</td>
</tr>
</tbody>
</table>

The curves for voltage, current and Power are shown below in Fig 6.19, Fig 6.20 and Fig 6.21 respectively.

Fig: 6.19 Voltage curve for neural network based MPPT method
Over the past decades there has been a strong resurgence in the field of data based nonlinear system modeling and identification involving researchers from diverse disciplines. The real world power system problems may neither fit the assumptions of a single technique nor be effectively solved by the strengths and capabilities of a single technique. One approach to deal with the complex real world problems is to integrate the use of two or more techniques in order to combine their different strengths and overcome each other weakness to generate hybrid solutions. Fuzzy Logic is the powerful tool for modeling uncertainty and approximate reasoning; however the presence of an expert is the most important part of the design. Expert less design requires data which must cover all the possibilities of operation. Neural networks are the best learning machines in the field.
Adaptive Neuro Fuzzy Inference Systems combines the advantage of fuzzy logic and neural networks in one package and gives an optimized fuzzy inference system embedding the whole knowledge of the system behavior (tracking operation).

PV cells behave as non-linear sources depending on climatic parameters. Insolation and Temperature are the most important factors influencing the maximum power delivered from the PV array. Here, a hybrid technique that combines fuzzy logic and neural networks is applied to identify the maximum power point.

6.3.1 Synergy Approaches

The hybrid system techniques are developed in a variety of ways as shown in Fig 6.22 which are classified into four types:

**Combination**: Current intelligent techniques mimic certain brain activities in a complementary way. As a result, a system's problem-solving capabilities are enhanced by combining intelligent techniques. Typical hybrid architecture is the sequential combination of neural networks and expert or fuzzy systems.
Integration: Combination is basic hybrid architecture, but in some cases the integration of other intelligent elements helps to determine the total system behaviour.

Fusion: A distinctive feature of neural network technology is its capability of learning and adaptation. When other techniques incorporate this feature they are able to increase their learning efficiency. From a topological view of hybrid architectures, this type of architecture is a fusion of intelligent techniques.

Association: Flexible intelligent systems require a distributed architecture where each element works autonomously and cooperatively.

6.3.2 Fuzzy Neural Network Hybrids

In Engineering, a class of artificial neural networks is proven to be capable of representing a class of unknown nonlinear input–output mappings with arbitrary small approximation error capability. The advantage of the FL approach is its logicality and transparency, where it is easy to incorporate a priori knowledge about a system into an explicit fuzzy rule base.

A Neuro-Fuzzy theory brings the ideas of ANN and FL together in a cohesive framework, such that the resultant model has structure for learning properties and is associated with a fuzzy rule base about the generated data knowledge.

Neuro-fuzzy systems combine the natural linguistic/symbolic transparency of fuzzy systems with the provable learning and representation capability of linear in the ANNs weights. The combination of qualitative based reasoning via fuzzy logic and quantitative adaptive numeric/data processing via ANNs, is a potentially powerful concept, since it allows within a single framework intelligent qualitative and quantitative reasoning is achieved. Truly intelligent systems must make use of all available knowledge: numerical, expert or heuristic rules and known functional relationships such as physical laws. Neuro-fuzzy systems allow all these.
There are two approaches to fuzzy neural architecture which are referred to as neuro-centric and fuzzy-centric. The neuro-centric approach gives primacy to the neural side of the description. Fuzzy arithmetic is used as a way of improving neuronal and/or entire network behaviour. On the other hand, in the fuzzy-centric approach, neural networks are used as ancillary tools either to determine membership functions, or to change the set of rules adaptively.

### 6.3.2.1 Neuro-Centric Approach

Fuzzy control learning algorithm for multi layer feed forward neural networks:

Although back-propagation is one of the most popular neural network algorithms, deficiencies such as the convergence speed and local minimum are identified. In this respect, a number of techniques are proposed to improve standard BP in one way or another. Among them, the fuzzy logic controller has become a promising alternative. Fig 6.23 presents the block diagram of a hybrid learning system, in which an on-line fuzzy logic controller is used to adapt the learning parameters of a multilayer perceptron with back-propagation learning. The objective is to provide a significant improvement in the rate of convergence of the learning process.

![Neuro-centric approach diagram](image)

**Fig: 6.23 Neuro-centric approach**

The main idea behind the fuzzy control of back-propagation learning is the implementation of heuristics in the form of fuzzy ‘IF..., THEN...’ rules that are used...
for the purpose of achieving a faster rate of convergence. The heuristics are driven by
the behavior of the instantaneous sum of squared errors. In the following heuristics,
the change of error (CE) is an approximation to the gradient of the error surface, and
the change of CE (CCE) is an approximation to the second-order gradient information
related to the acceleration of convergence:

If CE is small with no sign changes in several consecutive iterations of the
algorithm, then the value of the learning-rate parameter is increased.

If sign changes occur in CE in several consecutive iterations of the algorithm, then
the value of the learning-rate parameter is decreased. Decreasing is regardless of the
value of CCE.

If CE is small and CCE is small, with no sign changes for several consecutive
iterations of the algorithm, then the values of both the learning-rate parameter and the
momentum constant is increased.

The convergence of fuzzy back-propagation learning is faster and the steady-state
value of the mean-squared error is significantly smaller than in standard back-
propagation learning.

6.3.2.2 Fuzzy-Centric Approach

A fuzzy-centric hybrid system carries out fuzzy inference with NN structure, and
adjusts the fuzzy parameters using NN learning. Fuzzy logic is used to easily translate
heuristic reasoning and knowledge from qualitative descriptions to quantitative
descriptions. It has two important components: fuzzy rules and membership functions.
Although fuzzy logic can easily capture the salient features of a given process, rules
and membership functions are usually difficult to obtain. In this respect, neural
networks are trained to extract the rules and learn membership functions from some
domain data.
6.3.3 Adaptive Fuzzy Systems

The rules governing fuzzy systems are modified according to experience, i.e. the system is tuned. This can be achieved either by an expert or by the use of neural networks. Unsupervised neural networks find the set of rules for a fuzzy system by simply observing an expert's decision. Unsupervised learning is simpler and faster than supervised learning. Supervised training with neural networks, on the other hand, has not in general proved entirely satisfactory beyond the field of fuzzy rule refinement so most adaptive fuzzy systems remain unsupervised. The basic architecture of adaptive fuzzy system is presented in Fig 6.24.

![Adaptive fuzzy system architecture](image)

**Fig: 6.24 Adaptive fuzzy system architecture**

6.3.3.1 ANFIS: Adaptive Networks

An adaptive network is a multi layer feed forward network consisting of nodes and direction links through which nodes are connected. Moreover, part or all of these nodes are adaptive, which means their output depends on the parameters pertaining to these nodes and the learning rule specifies how these parameters are changed to minimize a prescribed error measure.

The basic learning rule of adaptive networks is based on gradient method which is for its slowness and tendency to become trapped in local minima, hence, a hybrid learning rule which speeds up the learning process is used. In order to achieve a
desired input-output mapping, the parameters are updated accordingly to given training data and a gradient based learning procedure described below:

**6.3.3.2 Basic Learning Rule**

Suppose that a given adaptive network has $L$ layers and the $k^{th}$ layer has $k$ nodes. The node in the $k^{th}$ layer can be denoted by $(k,i)$ and its node output by $O^k_i$. Since a node output depends on its incoming signals and its parameter set

$$O^k_i = O^k_i \left( O^{k-1}_{i-1}, \ldots, a, b, c \right)$$

Where a, b, c are the parameters pertaining to this node. Assuming that the given training data set has $P$ entries, the error measure for the $p^{th}$ entry of training data entry is defined as the sum of squared errors:

$$E_p = \sum_{m=1}^{(L)} (T_{m,p} - O^L_{m,p})^2 \quad \ldots \quad (6.26)$$

Where $T_{m,p}$ is the $m^{th}$ component of $p^{th}$ target output vector, and $O^L_{m,p}$ is the $m^{th}$ component of actual output vector produced by the presentation of the $p^{th}$ input vector.

In order to develop learning procedure that implements gradient descent in $E$ over the parameter space, calculate the error rate for $p^{th}$ training data for each node output $O$. The error rate for the output node at $(L, i)$ is calculated as

$$\frac{\partial E_p}{\partial O^L_{i,p}} = -2(T_{m,p} - O^L_{i,p}) \quad \ldots \quad (6.27)$$

For the internal node at $(k,i)$ the error rate is derived by the chain rule:

$$\frac{\partial E_p}{\partial O^k_{i,p}} = \sum_{m=1}^{(k+1)} \left( \frac{\partial E_p}{\partial O^{k+1}_{m,p}} \right) \ast \left( \frac{\partial O^{k+1}_{m,p}}{\partial O^k_{i,p}} \right) \quad \ldots \quad (6.28)$$

Where $1 \leq k \leq L - 1$. That is, the error rate of an internal node is expressed as a linear combination of the error rate of the nodes in the next layer.
If $\alpha$ is a parameter of the given adaptive network, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{\sigma, \sigma'} \left( \frac{\partial E_p}{\partial \sigma} \right) \left( \frac{\partial O'}{\partial \alpha} \right)$$

\hspace{1cm} \text{.... (6.29)}

Where $S$ is the set of nodes whose output depend on $\alpha$. Then the derivative of over all error measure $E$ with respect to $\alpha$ is

$$\frac{\partial E}{\partial \alpha} = \sum_{p} \left( \frac{\partial E_p}{\partial \sigma} \right)$$

\hspace{1cm} \text{.... (6.30)}

Accordingly, the update formula for the generic parameter $\alpha$ is

$$\Delta \alpha = -\eta \left( \frac{\partial E}{\partial \alpha} \right)$$

\hspace{1cm} \text{in which } \eta \text{ is a learning rate which is further expressed as}

$$\eta = \left\{ \frac{k}{\sqrt{\sum_{p} \left( \frac{\partial E_p}{\partial \sigma} \right)^2}} \right\}$$

\hspace{1cm} \text{.... (6.31)}

where $k$ is the step size, the length of each gradient transition in the parameter space. Usually, the value of $k$ is changed to vary the speed of convergence.

### 6.3.3.3 Learning Paradigms for Adaptive Networks

There are two learning paradigms for adaptive networks.

Batch learning (or off-line learning): The updation for parameters takes place only after the whole training data set is presented.

Pattern learning (or On-line learning): The parameters are updated immediately after each input-output pair.

### 6.3.3.4 Hybrid Learning Rule

The hybrid learning rule combines the gradient descent method and least square estimate (LSE) to identify parameters.

Assume that the adaptive network under consideration has only one output

$$\text{Output}=F(I, S)$$
I is the set of input variables and S is the set of parameters. If there exists a function $H^oF$ which is linear in some of the elements of S, then these elements are identified by LSE method. If the parameter set S is decomposed into two sets $S = S_1 \oplus S_2$ where $\oplus$ represents direct sum such that

$$H(\text{output}) = H^oF(I,S)$$  \hspace{1cm} \text{(6.32)}$$

which is linear in elements of $S_2$. Now, for given values of elements of $S_1$, plug training data into Equation 6.32 and obtain a matrix Equation 6.33.

$$AX = B$$  \hspace{1cm} \text{(6.33)}$$

Where $X$ is an unknown vector whose elements are parameters in $S_2$. Least square estimate of $X$, $X^\ast$ is sought to minimize the squared error $\|AX - B\|^2$.

Each epoch of this hybrid learning procedure is composed of a forward pass and a backward pass. In the forward pass, input data is supplied and functional signals go forward to calculate each node output and the parameters in $S_2$ are identified by sequential least squares method. After identifying parameters in $S_2$, the functional signals keep going forward till the error measure is calculated. In the backward pass the error rates propagate from the output end toward the input end and the parameters in $S_1$ are updated by the gradient method. Table 6.6 summarizes the hybrid learning procedure for ANFIS.

Table 6.6: Two passes in the hybrid learning procedure for ANFIS

<table>
<thead>
<tr>
<th></th>
<th>Forward pass</th>
<th>Backward pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premise parameters</td>
<td>Fixed</td>
<td>Gradient Descent</td>
</tr>
<tr>
<td>Consequent parameters</td>
<td>Least Squares Estimate</td>
<td>Fixed</td>
</tr>
<tr>
<td>Signals</td>
<td>Node outputs</td>
<td>Error rates</td>
</tr>
</tbody>
</table>
6.3.4 ANFIS Architecture

For simplicity assume the fuzzy inference system under consideration has two inputs x and y and one output z. Then the type-3 fuzzy reasoning is illustrated in Fig 6.26 (a) and the corresponding equivalent ANFIS architecture shown in Fig 6.26(b).

Layer 1: Every node i in this layer is a square node with a node function

\[ O_i^1 = \mu_{A_i}(x) \]

Where x is the input to node i, and \( A_i \) is the linguistic label associated with this node function. (or) \( O_i^1 \) is the membership function of \( A_i \) and it specifies the degree to which the given x satisfies the quantifier \( A_i \). \( \mu_{A_i}(x) \) is the bell-shaped function

\[
\mu_{A_i}(x) = \frac{1}{1 + \left( \frac{x - c_i}{a_i} \right)^{2b_i}} 
\]

Where \( \{a_i, b_i, c_i\} \) is the parameter set. As the value of these parameters change, the bell shaped function vary accordingly. Parameters in this layer are referred as premise parameters.

Layer 2: Every node in this layer is a circular node labelled \( \prod \) which multiplies the incoming signals and sends the product out. For instance,

\[ O_i^2 = W_i = \mu_{A_i}(x) \times \mu_{B_i}(y) \quad i=1,2 \]

Each node output represents the firing strength of a rule.

Layer 3: Every node in this layer is a circular node labeled N, The \( i^{th} \) node calculates the ratio of the \( i^{th} \) rule’s firing strength to the sum of all rule’s firing strengths:

\[
\frac{W_i}{\sum_i W_i} = \left\{ \frac{W_i}{\sum_i W_i} \right\}
\]

Outputs of this layer are called as normalized firing strengths.
Layer 4: Every node in this layer is a square node with a node function

\[ O_i^4 = \overline{W_i} f_i = \overline{W_i} (p_i x + q_i y + r_i) \] .... (6.37)

Where \( \overline{W_i} \) is the output of layer 3, and \( \{p_i, q_i, r_i\} \) is the parameter set. Parameters in this layer are referred as consequent parameters.

Layer 5: The single node in this layer is a circle node labeled \( \sum \) that computes the overall output as the summation of all incoming signals, i.e.

\[ O_i^5 = \text{overall output} = \sum_i \overline{W_i} f_i = \left( \frac{\sum_i \overline{W_i} f_i}{\sum_i \overline{W_i}} \right) \] .... (6.38)
Fig: 6.25 (a) Two-input type-3 ANFIS with nine rules
(b) Corresponding fuzzy subspaces

Fig: 6.26 (a) Type-3 fuzzy reasoning (b) Equivalent ANFIS
6.3.5 Neuro-Fuzzy Network Model And Calculation Algorithm

Fuzzy logic modeling approach is used to find out the maximum power point of PV cells. Proper choice of the parameters and rigorous validation of choice are entailed to accomplish an accurate representation. In case of significant levels of missed observations, the selection of the type and values of the parameters of membership function is not easy. In these circumstances, selection is based on the characteristics of input /output data. By utilizing neuro-adaptive techniques it is possible to learn information about the data set so that the parameters of the membership functions are computed in such a way that the associated fuzzy inference system maps the given input /output data properly. ANFIS systems combine the logical influence power of fuzzy systems and the numerical power of neural networks. The applied ANFIS model in this case is a first order Takagi-Sugeno model. The output of each rule is a linear combination of input variables plus a constant term and the final output is the weighted average of each rule’s output.

ANFIS uses a hybrid learning algorithm to identify consequent parameters of Sugeno type fuzzy inference systems. It applies a combination of the least squares method and Back-propagation gradient descent method for training fuzzy inference system membership functions parameters to emulate a given training set.

Fig: 6.27(a) Fuzzy Inference System
This method of Neuro-fuzzy system owes much from the feed forward neural network with supervised learning capability. Fig 6.27(a) shows the fuzzy inference system while Fig 6.27(b) shows the equivalent ANFIS system.

6.3.6 ANFIS Network Specifications

- Inputs: Insolation(G)
  Temperature (T)
- Output: Maximum power point Voltage
- Number of Membership functions:
  Three (3) for Insolation
  Three (3) for Temperature
- Number of Rules: 9 (Sugeno FIS)

6.3.7 Algorithm For Neuro-Fuzzy Based MPPT

STEP 1: Fuzzification Step (Input Step):

The First Layer

This layer is a basic fuzzification layer where the crisp inputs are allocated to relative fuzzy values. The generalized bell shaped membership functions are utilized.

The output of the layer one for \( i^{th} \) membership function is calculated as:
Where $O_i^1$ is the output of first layer, G is the Insolation and T is the temperature (crisp input values) for this application. $A_i$ and $B_i$ are linguistic labels characterized by appropriate membership functions, $a_i, b_i$ and $c_i$ are the premise parameters described from the characteristics of bell shaped function. These premise parameters are fixed in the forward pass and are updated in the adaptation process by gradient descent method in the backward pass.

STEP 2:

Calculation of rules (or) generalization of firing strengths:

The Second Layer

Firing strength means the degree to which an antecedent part of fuzzy rule is satisfied and it shapes the corresponding output function. The And logical operator is applied to obtain an output per rule. The product inference rule is used at fuzzification level to yield output values $\left( O_i^2 \right)$ at this level. The outputs at this node are labelled as $\prod$ as a result of multiplication of inputs from the layer one nodes.

$$O_i^2 = W_i = \mu A_i(G) \times \mu B_i(T)$$ .... (6.41)

Where $i=1,2,\ldots,9$

The value of node output represents the strength of the rule.

STEP 3:

Calculation of The Ratio Of Firing Strengths:
The Third Layer

The nodes in this layer are represented by circular nodes labeled N. The main objective of this step is to calculate the ratio of each \(i^{th}\) rule’s firing strength to the sum of all rule’s firing strength, which is also called as normalized firing strength with the output \(O^3_i\) at this step.

\[
O^3_i = \overline{W_i} = \left\{ \frac{W_i}{\sum_i W_i} \right\} \quad \text{Where } i=1, 2, \ldots, 9 \quad \ldots (6.42)
\]

STEP 4:

Contribution Of Each Rule: The Fourth Layer

Calculation of the contribution of each \(i^{th}\) rule towards the total output \(O^4_i\) is formed at this step.

\[
O^4_i = \overline{W_i}(p_i T + q_i G + r_i) \quad \text{Where } i=1, 2, \ldots, 9 \quad \ldots (6.43)
\]

Where \(\overline{W_i}\) is the output of layer 3 and \(p_i, q_i, r_i\) are consequent parameters which are updated on forward pass by least square estimates and are fixed in backward pass. This layer establishes the relation between weight based premise parameters and linear summation based consequent part of the adapted ANFIS structure.

STEP 5

Summation Step (Defuzzification Step): The Fifth Layer

Each rule’s fuzzy results are transformed into a crisp output \(O^5_i\) at this step. The single output node in this layer sums up all the outputs of the previous layer as the output of ANFIS in this case is the maximum power point voltage of the PV cells.

\[
K_p = \sum_i \overline{W_i} f_i = \sum_i \left( \frac{W_i f_i}{\overline{W_i}} \right) \quad \text{Where } i=1, 2, \ldots, 9 \quad \ldots (6.44)
\]
3.8 Results For Neuro- Fuzzy Based MPPT

Network Training:

A set of 25 training patterns was presented to the network, while another set of 9 points were randomly chosen as the testing data points which were not utilized for training. These training patterns were uniformly distributed to cover all the typical input space in order to get good performance where temperature ranges from -40 Degrees Celsius to fifty Degrees Celsius and solar irradiation ranged from 0 to 1000W/m². The implementation of Neuro-Fuzzy based MPPT for maximum power point tracking using Matlab is done.

Initial Memberships of the temperature T and Irradiation G are shown in Fig 6.29 while Fig 6.30 presents the final membership functions after ANFIS training. Fig 6.32 presents the ANFIS architecture obtained.

Training Data for Neuro-Fuzzy Controller:

1) 900 10 16.16
2) 830 35 8.75
3) 910 12 16.39
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4)</td>
<td>850</td>
<td>50</td>
<td>20.36</td>
</tr>
<tr>
<td>5)</td>
<td>920</td>
<td>13</td>
<td>16.57</td>
</tr>
<tr>
<td>6)</td>
<td>600</td>
<td>52</td>
<td>20</td>
</tr>
<tr>
<td>7)</td>
<td>770</td>
<td>47</td>
<td>19.9</td>
</tr>
<tr>
<td>8)</td>
<td>700</td>
<td>42</td>
<td>19.9</td>
</tr>
<tr>
<td>9)</td>
<td>950</td>
<td>23.75</td>
<td>17.7</td>
</tr>
<tr>
<td>10)</td>
<td>980</td>
<td>24.5</td>
<td>17.89</td>
</tr>
<tr>
<td>11)</td>
<td>985</td>
<td>24.62</td>
<td>17.8</td>
</tr>
<tr>
<td>12)</td>
<td>995</td>
<td>24.87</td>
<td>18</td>
</tr>
<tr>
<td>13)</td>
<td>180</td>
<td>30</td>
<td>13.51</td>
</tr>
<tr>
<td>14)</td>
<td>220</td>
<td>40</td>
<td>16.77</td>
</tr>
<tr>
<td>15)</td>
<td>825</td>
<td>20.6</td>
<td>17.1</td>
</tr>
<tr>
<td>16)</td>
<td>925</td>
<td>18</td>
<td>17.4</td>
</tr>
<tr>
<td>17)</td>
<td>940</td>
<td>20</td>
<td>17.68</td>
</tr>
<tr>
<td>18)</td>
<td>930</td>
<td>23</td>
<td>17.7</td>
</tr>
<tr>
<td>19)</td>
<td>650</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>20)</td>
<td>955</td>
<td>23.87</td>
<td>17.7</td>
</tr>
<tr>
<td>21)</td>
<td>960</td>
<td>24</td>
<td>17.8</td>
</tr>
<tr>
<td>22)</td>
<td>970</td>
<td>24.25</td>
<td>17.9</td>
</tr>
<tr>
<td>23)</td>
<td>975</td>
<td>24.37</td>
<td>17.8</td>
</tr>
<tr>
<td>24)</td>
<td>100</td>
<td>-40</td>
<td>7.8995</td>
</tr>
<tr>
<td>25)</td>
<td>100</td>
<td>0</td>
<td>11.74</td>
</tr>
</tbody>
</table>

**Root Mean Square Error:**

Root mean square error evaluates the magnitude of deviations between the measured values and actual predictions.
RMSE = \left[ \frac{1}{n} \sum_{i=1}^{n} (m_i - z_i)^2 \right]^{0.5} \quad \text{.... (6.45)}

Where

- \( m_i \) = measurements
- \( z_i \) = model outputs
- \( N \) = number of observations

Fig 6.31 presents the root mean square error obtained.

Validation of the Model:

The accuracy of the obtained model is validated using testing data. Fig 5.33 shows the variations of \( V_{pp}(V) \) for various insolation and temperature conditions. The obtained model designed and trained has shown a very fast convergence with little number of memberships. The accuracy of the model is tested using a set of data different from training and the ANFIS controller gives accurate results and is able to closely mimic the actual values.

![Diagram showing membership functions](image)

a) Original membership function for irradiation
b) Original membership function of temperature

Fig: 6.29 Original memberships of T and G before training

a) Final membership function of Irradiation

b) Final membership function for temperature

Fig: 6.30 Final memberships of T and G after training
Fig: 6.31 RMSE of the training data with respect to training epoch

Fig: 6.32 ANFIS architecture

Fig: 6.33 $V_{pp}$ plotted for various values of insolation and temperature
Table 6.7 summarizes the Neuro-Fuzzy controller output values for various insolation and temperature conditions.

Table 6.7: Output of Neuro-Fuzzy controller

<table>
<thead>
<tr>
<th>Insolation(W/m²)</th>
<th>Temperature(Degrees Celsius)</th>
<th>Maximum power tracked by Neuro–fuzzy controller (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>25</td>
<td>59.91</td>
</tr>
<tr>
<td>930</td>
<td>12</td>
<td>51.45</td>
</tr>
<tr>
<td>700</td>
<td>40</td>
<td>45.45</td>
</tr>
<tr>
<td>500</td>
<td>46</td>
<td>32.091</td>
</tr>
<tr>
<td>950</td>
<td>15</td>
<td>53.7</td>
</tr>
<tr>
<td>820</td>
<td>40</td>
<td>53.46</td>
</tr>
<tr>
<td>200</td>
<td>20</td>
<td>9.74</td>
</tr>
<tr>
<td>100</td>
<td>-10</td>
<td>3.48</td>
</tr>
</tbody>
</table>

6.4 CONCLUSIONS

For a 60 W module, the power output is given by fuzzy logic based method is 59.9 W, neural network based method is 59.8885 W and by neuro-fuzzy based method is 59.91 W. The neural network and neuro-fuzzy methods are advantageous due to auto training of the network.