CHAPTER 5
IMPLEMENTATION OF ALGORITHMS FOR FINGERPRINT RECOGNITION

5.1 INTRODUCTION

This chapter presents the implementation of ANN and Fuzzy logic algorithms for identifying Fingerprint. Each algorithm is introduced and the implementation procedures for recognition of Fingerprint is presented.

5.2 ARTIFICIAL NEURAL NETWORK

An Artificial Neural Network (ANN) is an abstract simulation of a real nervous system that contains a collection of neuron units, communicating with each other via axon connections. Such a model bears a strong resemblance to axons and dendrites in a nervous system. Due to this self-organizing and adaptive nature, the model offers potentially a new parallel processing paradigm. This model could be more robust and user-friendly than the traditional approaches. ANN can be viewed as computing elements, simulating the structure and function of the biological neural network. These networks are expected to solve the problems, in a manner, which is different from conventional mapping. Neural networks are used to mimic the operational details of the human brain in a computer. Neural networks are made of artificial ‘neurons’, which are actually simplified versions of the natural neurons that occur in the human brain. It is hoped, that it would be possible to replicate some of the desirable features of the human brain by constructing networks that consist of a large number of neurons. A neural architecture comprises massively parallel adaptive elements with interconnection networks, which are structured hierarchically.

Artificial neural networks are computing elements, which are based on the structure and function of the biological neurons. These networks have nodes or neurons, which are described by difference or differential equations.
The nodes are interconnected layer-wise or intra-connected among themselves. Each node in the successive layer receives the inner product of synaptic weights with the outputs of the nodes in the previous layer. The inner product is called the activation value. The activation value is passed through a non-linear function.

The function of a neuron is shown in Figure 5.1. When the vectors are binary or bipolar, hard-limiting non-linearity is used. When the vectors are analog, a squashed function is used. Some of the squashed functions are sigmoid (0 to 1), tanh (-1 to +1), Gaussian, logarithmic and exponential. A network with two states of a neuron (0 or 1, and -1 or 1) is called ‘discrete’, and the same with a continuous output is called ‘analog’. If, in a discrete network at a particular time ‘t’, the state of every neuron is updated, the network is said to be synchronous. If the state of only one neuron is updated, the network is said to be asynchronous. A network is feed forward, if there is no closed chain of dependence among neural states. The same network is feed backward, if there is such a closed chain. When the output of the network depends upon the current input, the network is static (no memory). If the output of the network depends upon past inputs or outputs, the network is dynamic (recurrent). If the interconnection among neurons changes with time, the network is adaptive; it is called non-adaptive. The synaptic weight updation of the networks can be carried out by supervised methods, or by unsupervised methods, or by fixed weight association networks methods. In the case of the supervised methods, inputs and outputs are used; in the unsupervised methods, only the inputs are used; and in the fixed weight association networks methods, inputs and outputs are used along with pre-computed and pre-stored weights.

Some of the supervised learning algorithms are the perceptrons, decision-based neural networks, adaptive linear element (ADALINE), multilayer perceptron, temporal dynamic models and hidden Markov analysis.
The various unsupervised learning algorithms are neo-cognition, self-organizing feature map, competitive learning, adaptive resonance theory (ART) and the principal component analysis. The fixed weight networks are Hamming net, Hopfield net and the combinatorial optimization. The total pattern recognition system constitutes instantiation space, feature extraction, training the network, and the testing the network.

5.3 BACK PROPAGATION ALGORITHM

Back propagation algorithm is a supervised method which maps inputs with target outputs. The BPA learns the statistical feature values obtained from equation 4.1-4.5, using the steepest decent concept. The topology of the artificial neural network is as follows: Input layer = 5 Nodes, Hidden layer = 6 nodes, and Output layer = 1 Node.

During training the ANN, normal fingerprint image details are used for training the ANN. Information flows from input layer to output layer through hidden layer. Learning is done by updating weights from output layer to input.
layer through hidden layer. The training is stopped once the desired mean squared error (MSE) is reached. During forward propagation, the weights of the network are initialized for the first pattern. The inputs and outputs of a pattern are presented to the network. The output of each node in the successive layers is calculated in the forward propagation. In the reverse process, weights are updated between layers.

The above process completes one weight updation. Second pattern is presented and the above sequences are followed for the second weight updation. When all the training patterns are presented, a cycle of iteration or epoch is completed. If the MSE is less than a specified value, training is stopped and the final weights are stored.

5.3.1 STEPS INVOLVED IN TRAINING BPA

FORWARD PROPAGATION:

Step 1: The weights of the ANN are initialized.

Step 2: The inputs and outputs of a Fingerprint training patterns are presented to the BPA network. The output of each node in the successive layers is calculated using equation (5.1)

\[
o_{\text{output of a node}} = \frac{1}{1+\exp(-\sum w_{ij} x_i)}
\]  

(5.1)

Step 3: The error of a pattern is calculated using equation (5.2).

\[
E(p) = \frac{1}{2} \sum (d(p) - o(p))^2
\]

(5.2)

REVERSE PROPAGATION (Weight updation)

Step 4: The error for the nodes in the output layer is calculated using equation (5.3).

\[
\delta_{\text{output layer}} = o(1-o)(d-o)
\]

(5.3)

Step 5: The weights between output layer and hidden layer are updated using equation (5.4).

\[
W_{(n+1)} = W_{(n)} + \eta \delta_{\text{output layer}} o_{\text{hidden layer}}
\]

(5.4)
Fig. 5.2 Flow chart of the back propagation algorithm
**Step 6:** The error for the nodes in the hidden layer is calculated using equation (5.5)

\[ \delta_{\text{Hidden layer}} = o(1-o) \sum \delta_{\text{output layer}} W_{\text{updated weights between hidden and output layer}} \]  

(5.5)

**Step 7:** The weights between hidden layer(s) and input layer are updated using equation (5.6).

\[ W_{(n+1)} = W(n) + \eta \delta_{\text{hidden layer}} o_{\text{input layer}} \]  

(5.6)

The above steps complete one weight updation. Second pattern is presented and the above steps are followed for the second weight updation. When all the training patterns are presented, a cycle of iteration or epoch is completed. The errors of all the training patterns are calculated using equation (5.7).

\[ E_{\text{MSE}} = \sum E(p) \]  

(5.7)

Figure 5.2 presents flow-chart describing the working of BPA.

**5.3.2 Convergence curve of BPA**

The weight updating algorithm incorporates various parameters to update the weights of the connection strength matrices between input and hidden layer, hidden and output layers. The various parameters used in the BPA algorithms are as follows:

\( \alpha \) is an accelerating factor (>0 and \( \leq 1 \))

\( \eta \) is a learning factor (>0 and \( \leq 1 \))
Fig. 5.3 Effect of number of nodes in the hidden layer for the ANN trained by using BPA

The initialization of the weights and the thresholds are in the range of 0.52 to 0.75. The iterations required by the network which are trained by using BPA for different number of nodes in the hidden layer to reach MSE of 0.01, are shown in Figure 5.3. Number of hidden layer=1 and the $\eta = 1$. X-axis presents number of nodes in one hidden layer.
The learning factor $\eta$ is supposed to guide the convergence rates of the network to the desired MSE with less number of iterations. Sometimes $\eta$ will make the network to converge to the desired MSE after an increased number of iterations. For 6 nodes in the hidden layer it requires 985 iterations for the network to reach MSE of 0.01 when $\eta$ is 1.0 and 2404 iterations for the network to reach MSE of 0.01 when $\eta$ is 0.05. The convergence rates of the network for various numbers of nodes in the hidden layer for different values of $\eta$ is shown in Figure 5.4.
To achieve faster convergence of the network an accelerating factor is used which is a parameter called momentum factor ($\alpha$). The network is trained with $\alpha$ and without $\alpha$. The value of $\alpha$ is from 0 to 1. For 6 nodes in the hidden layer, it requires 985 iterations for the network to reach MSE of 0.01 without $\alpha$, and 380 iterations for the network to reach MSE of 0.01 with $\alpha$. The value of $\alpha$ used is 0.8. For other values of $\alpha$, the network requires very large number of iterations to reach MSE of 0.01. The convergence rates of the network trained with $\alpha$ and without $\alpha$ are shown on Figure 5.5. The network trained with $\alpha$ requires less number of iterations to reach the desired MSE.

**Fig.5.5 Effect of $\alpha$ in the network trained by using Back propagation algorithm**
Fig. 5.6 Mean Squared Error of the ANN trained by using BPA for Fingerprint recognition

Figure 5.6 shows the convergence curve for the topology of 3, 6 and 9 nodes in the hidden layer. The network was trained to reach MSE of 0.01853. It converged in 405 iterations. If the number of patterns is further increased from 100 then the convergence iterations will increase. In addition the convergence depends on the orthogonality of data presented for training data set. If the subsequent patterns are not orthogonal, the convergence would take a long time or it may not converge.
5.4 RADIAL BASIS FUNCTION (RBF)

An RBF neural network consists of an input and output layer of nodes and a single hidden layer. Each node in the hidden layer implements a basis function $G(x_i)$ and the number of hidden nodes is equal to the number of data points in the training database.

![Radial Basis Function Neural Network](image)

**Fig.5.7 Radial basis function neural network**
The RBF approximates the unknown function that maps the input to the output in terms of a basis function expansion, with the functions, \( G(x_i) \), as the basis functions. The input-output relation for the RBF is given by equation (5.8)

\[
y_i = \sum_{j=1}^{N} w_{ij} G(x, x_j), i=1,2, \ldots, M.
\]  

(5.8)

where \( N \) is the number of basis functions used, \( y=(y_1, y_2, \ldots y_m)^T \), is the output of the RBF, \( X \) is the test input, \( X_j \) is the center of the basis function and \( w_{ij} \) are the expansion coefficients or weights associated with each basis function.

Each training data sample is selected as the center of a basis function. Basis functions \( G(X_i, X_j) \) that are radially symmetric are called radial basis functions. Commonly used radial basis functions include the Gaussian and inverse multi quadrics.

The Figure 5.7 shows ANN topology trained by RBF algorithm.

**5.4.1 Training RBF is done as follows,**

**Step 1:** Finding distance among patterns and centers.

**Step 2:** Creating an RBF matrix whose size will be (np X cp), where np= number of Fingerprint training patterns (100) and cp is number of centers to form rbf nodes. The number of centers chosen should make the RBF network learn the maximum number of Fingerprint training patterns under consideration.

**Step 3:** Final weights are calculated which are inverse of RBF matrix multiplied with Target (labelling) values.

**Step 4:** During testing the RBF network, statistical test pattern of a Fingerprint are presented in the input layer of the RBF network. The pattern is processed with the final weights obtained during training. Based on the result obtained, the pattern is identified as Fingerprint of a person.

Figure 5.8 gives the flow-chart for RBF implementation. Table 5.1 gives the
steps involved in training an RBF. Table 5.2 gives the steps for testing the RBF.

![Radial basis function flow-chart](image)

**Fig. 5.8 Radial basis function flow-chart**
### Table 5.1 Training RBF

<table>
<thead>
<tr>
<th>Step 1: Apply Radial Basis Function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Input = 5</td>
</tr>
<tr>
<td>No. of Patterns = 100</td>
</tr>
<tr>
<td>No. of Centre = 100</td>
</tr>
<tr>
<td>Calculate RBF as</td>
</tr>
<tr>
<td>RBF = ( \exp(-X) )</td>
</tr>
<tr>
<td>Calculate Matrix as</td>
</tr>
<tr>
<td>( G = RBF )</td>
</tr>
<tr>
<td>( A = G^T \ast G )</td>
</tr>
<tr>
<td>Calculate</td>
</tr>
<tr>
<td>( B = A^{-1} )</td>
</tr>
<tr>
<td>Calculate</td>
</tr>
<tr>
<td>( E = B \ast G^T )</td>
</tr>
<tr>
<td><strong>Step 2:</strong> Calculate the Final Weight.</td>
</tr>
<tr>
<td>( F = E \ast D )</td>
</tr>
<tr>
<td><strong>Step 3:</strong> Store the Final Weights in a File</td>
</tr>
</tbody>
</table>

### Table 5.2 Testing RBF

<table>
<thead>
<tr>
<th>Step 1: Read the Input pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2: Read the final weights</td>
</tr>
<tr>
<td>Step 3 Calculate.</td>
</tr>
<tr>
<td>Numerals = ( F \ast E )</td>
</tr>
<tr>
<td><strong>Step 4:</strong> Check the output with the templates.</td>
</tr>
</tbody>
</table>
Figure 5.9 presents Fingerprint recognition graphs for different number of centers. The graph is plotted by considering 10 Fingerprint images for testing the performance of RBF.

5.5 FUZZY LOGIC FOR FINGERPRINT RECOGNITION

Fuzzy Logic (FL) is a multi-valued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low. Fuzzy systems are an alternative to traditional notions of set membership and logic.

The training and testing fuzzy logic is to map the input pattern with target output data. For this, the inbuilt function has to prepare membership table and finally a set of number is stored. During testing, the membership function is used to test the pattern.
Training Fuzzy logic for identifying fingerprints

Step 1: Read the statistical features of the wavelet coefficients and its target value.

Step 2: Create Fuzzy membership function.


Step 4: Process with target values.

Step 5: Obtain final weights.

Testing Fuzzy logic for identifying fingerprints

Step 1: Input a pattern (statistical features of the wavelet coefficients).

Step 2: Process with Fuzzy membership function.

Step 5: Find the cluster to which the pattern belongs.

Step 4: Obtain estimated target values.

Step 5: Identify the fingerprint.

RADII specifies the range of influence of the cluster center for each input and output dimension, assuming the data falls within a unit hyperbox (range [0 1]). Specifying a smaller cluster radius will usually yield more, smaller clusters in the data, and hence more rules. When RADII is a scalar it is applied to all input and output dimensions.

![Graph showing the performance of Fuzzy logic](attachment:image.png)

**Fig.5.10 Performance of Fuzzy logic**
Figure 5.10 presents number of persons’ fingerprints and Fuzzy logic estimation. In all the 10 fingerprints, the recognition is 100%.

5.6 SUMMARY

This chapter has presented the implementation of BPA, RBF and Fuzzy logic by using statistical features of Fingerprints for identifying the Fingerprints. Chapter 6 presents discussions on the performance of BPA, RBF and Fuzzy logic in identifying Fingerprints.