CHAPTER 4
ARTIFICIAL NEURAL NETWORK (ANN)

4.1 INTRODUCTION

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 4.1. 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).
Fig. 4.1 Operation of a neuron
If the interconnection among neurons changes with time, the network is adaptive; it is called non-adaptive. The synaptic weight updating 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.

The development of artificial neural networks was first reported in the early forties by McCullah et al., 1943. In this model, a neuron fires if the sum of its excitatory inputs exceeds its threshold. This happens, as long as it receives no inhibitory input. Using this model, it is possible to construct a network that can compute any logical function. Rosenblatt, 1961, found that the McCulloh-Pitts
model was unbiological. In order to overcome the deficiencies in the McCulloh-Pitts model, he found out a new model, namely, the perceptron model, which could be utilized to learn and generalize. Further, he investigated several mathematical models, which included competitive learning or self-organization, and forced learning which is somewhat similar to reinforcement learning.

In addition to the above two types of learning, the concept of supervised learning was developed and incorporated in the adaptive linear element model (ADALINE). The ADALINE was found by Widrow et al., 1960.

The ADALINE is a single neuron, which uses a method to descend the gradient of the error, by using the supervised learning. The ADALINE is a linear neuron, and it is helpful to discriminate the patterns, which are linearly separable. The concept of multi-layer ADALINEs or multi-layer network was developed for patterns which were not linearly separable. The training of the multi-layer network was first explained by Werbos, 1994, as back-propagation algorithm (BPA) in his Ph.D. dissertation. His work did not become popular. Rumelhart, 1996, and his group published the parallel processing, a two-volume collection of studies on a broad variety of neural network configurations. Through these books, the concept of back-propagation algorithm became popular for training a multi-layer network. Lippmann, 1977, briefed the concept of different algorithms in his tutorial paper, and he still made neural networks more popular.
Much work has been carried out, with respect to the number of hidden layers, the number of hidden nodes in the hidden layer, methods of representing the patterns, training the network with initial random weights at different ranges, types of error criteria used and selection of patterns. Even though the training procedure for the neural network is unique and problem-oriented, it is sufficient to have one hidden layer for most of the problems solved by supervised training. Sietsma, 1991, have analyzed the various training strategies with more than one hidden layer and finally claimed that one hidden layer was sufficient.

Chester, 1990, claimed better performance for the network with two hidden layers. The number of nodes in a hidden layer should be, neither too many, nor too few. Too many nodes in the hidden layer will result in the oscillation of the mean squared error (MSE) around a particular value without any convergence; or sometimes the network converges to one of the local minima. Similarly, too few a number of nodes in the hidden layer will sometimes be just suitable, only to learn the training patterns, but generalization of the network is not possible. Therefore, it is necessary that there should be a way to find out the optimum number of nodes in a hidden layer. Hirose et al., 1991, adapted a different approach, by using an algorithm based on MSE to estimate the same. To overcome the difficulty of analyzing the number of nodes in the hidden layer, Weymaere et al., 1991, used
Gaussian function in the hidden nodes and sigmoid function in the output nodes. Fujita, 1992, analyzed hidden unit function.

In most of the supervised training methods, the patterns are presented in a pre-determined sequence in a cycle. Normally, the order of presentation of the patterns is maintained in all the cycles. Ridgway, 1962, found in his thesis that cyclic presentation of patterns could lead to cyclic adaptation. These cycles would cause the weights of the entire network to cycle, by preventing convergence. Various error criteria were been tried by Zaki, 1964 and Walach et al., 1984, for better convergence of the network. Quantization of the weights and training BPA has been analyzed by shoemaker, Carlin and Shimabukuro, 1991. Analysis of BPA with respect to mean weight behavior was done by Bershad et al., 1993.

In reality, most of the patterns are not linearly separable. Non-linear classifiers are used for pattern classification, in order to achieve good separability. The multiplayer network is a non-linear classifier, since it uses hidden layer. In addition to multiplayer network, polynomial discriminant function (PDF) is also a non-linear classifier. In the PDF, the input vector is pre-processed similar to the suggestions by Specht, 1990.

Normally, neural networks are used for classify patterns by learning from samples. Different neural networks paradigms employ different learning rules. In some way, all these paradigms determine different pattern statistics from a set of
training samples. Then, the network classifies new patterns on the basis of these statistics. The BPA uses steepest descent method, which is slow and linear in convergences. This algorithm may get stuck in local minima.

4.2 ADVANTAGES OF ANN OVER OTHER CLASSICAL METHODS

Unlike statistical estimators, ANN estimates a function without a mathematical model of how outputs depend functionally on the inputs; it learns from samples and offers robust and adaptive processing capabilities, by adopting adaptive learning and self-organization rules. It enhances the network approximation classification and has noise-immunity capabilities. It employs many processing units enhanced by extensive interconnectivity.

4.3 AREAS OF APPLICATIONS OF ANN

The application domains of ANN are in association / clustering / classification, pattern recognition, regression / generalization, and in optimization. The different areas of applications are in computer vision, signal/image processing, speech / character recognition, remote sensing and in controls.
4.4 NORMALIZATION OF THE PATTERNS

The patterns are normalized so that the values of the features are in the range of 0 to 1, and the computational complexity is reduced. The normalization of the patterns is done by:

\[ x_i = \frac{x_i}{x_{\text{max}}} \]  

(4.1)

Where

- \( x_i \) is the value of a feature, and
- \( x_{\text{max}} \) is the maximum value of the feature.

4.4.1 Selection of patterns for training

The numbers of classes, which are based on the classification range of the outputs, are decided. If only one output is considered, the range of classification is simple. If more than one output is considered, a combination criterion has to be considered. The total numbers of patterns are decided for each class. Out of these patterns, the number of patterns to be used for training the network is decided. The remaining patterns are used for testing the classification performance of the network. The patterns selected for training the network should be, such that they represent the entire population of the data.
The selection of patterns is done by:

\[ E_i^2 = \frac{\sum_{j=1}^{nf} (x_{ij} - \bar{x}_j)^2}{\sigma_i^2} \]  

(4.2)

Where

\( E_i^2 \) is the maximum variance of a pattern,

\( nf \) is the number of features, and

\[ \sigma_i^2 = \frac{\sum_{j=1}^{nf} (x_{ij} - \bar{x}_j)^2}{L} \]  

(4.3)

\( \bar{x}_j \) is the mean for each feature, and

\( L \) is the number of patterns

The value of \( E_i^2 \) is found for each pattern. Patterns with maximum \( E_i^2 \) are chosen from each class for training the network.
4.4.2 Training strategies for the network

For the network to learn the patterns, different weight updating algorithms has been developed. They are called supervised methods and unsupervised methods. Since both the inputs and outputs are considered for emotional facial expression classification, supervised learning technique has been used.

The network functions on a supervised learning strategy. The inputs of a pattern are presented. The output of the network obtained in the output layer is compared with the desired output of the pattern. The difference between the calculated output of the network and the desired output is called error (MSE) of the network for the pattern presented. This error is propagated backwards, such that the weights connecting the different layers are updated. By this process, the MSE of the network for the pattern presented is minimized. This procedure is summed up. After presenting the last training pattern, the network is considered to have learnt all the training patterns through iterations, but the MSE is large. To minimize MSE, the network has to be presented with all the training patterns many times. There is no guarantee that the network will reach the global minimum; instead, it will reach one of the local minima. The MSE may increase, which means, divergence rather than convergence. Sometimes, there may be oscillations between convergence and divergence.
The training of the network can be stopped, either by considering MSE or by considering classification performance as the criterion. When classification performance is considered as the criterion, test patterns are presented at the end of each epoch (iteration). Once the desired performance is obtained, training of the network is stopped. When MSE is considered as the criterion, one may not know the exact MSE, to which the network has to be trained. If the network is trained till it reaches a very low MSE, over fitting of the network occurs. Over fitting represents the loss of generality of the network. That is, the network classifies only the patterns, which are used during training, and not the test patterns.

4.5 BACK-PROPAGATION ALGORITHM (BPA)

The BPA uses the steepest-descent method to reach a global minimum. The flow chart for the BPA is given in Figure 4.2. The number of layers and number of nodes in the hidden layers are decided. The connections between nodes are initialized with random weights. A pattern from the training set is presented in the input layer of the network and the error at the output layer is calculated. The error is propagated backwards towards the input layer and the weights are updated. This procedure is repeated for all the training patterns.
Fig. 4.2 Flow chart of the back propagation algorithm
Fig.4.3 Topology of BPA

Figure 4.3 presents the topology of BPA for learning the FEE. The image is segmented. FLD features are obtained and presented for training the BPA. Target output is presented in the output layer. The network is stopped training, once the final weights are obtained. The weights are stored in a database for testing purpose.
At the end of the each iteration, test patterns are presented to ANN, and the classification performance of ANN is evaluated. Further training of ANN is continued till the desired classification performance is reached.

**STEPS INVOLVED.**

**FORWARD PROPAGATION**

1. The weights and thresholds of the network are initialized.

2. The inputs and outputs of a pattern are presented to the network.

3. The output of each node in the successive layers is calculated.

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

4. The error of a pattern is calculated

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

**REVERSE PROPAGATION**

1. The error for the nodes in the output layer is calculated

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

2. The weights between output layer and hidden layer are updated

   \[
   W(n+1) = W(n) + \eta \delta_{\text{output layer}} o_{\text{hidden layer}}
   \]
3. The error for the nodes in the hidden layer is calculated

\[ \delta_{\text{hidden layer}} = o(1-o) \sum \delta_{\text{output layer}} \ W_{\text{updated weights between hidden and output layer}} \] (4.9)

4. The weights between hidden and input layer are updated.

\[ W(n+1) = W(n) + \eta \delta_{\text{hidden layer}} o_{\text{input layer}} \] (4.10)

The above steps complete one weight updation.

5. Second pattern is presented and the above steps are followed for the second weight updation.

6. When all the training patterns are presented, a cycle of iteration or epoch is completed.

7. The errors of all the training patterns are calculated and displayed on the monitor as the mean squared error (MSE).

\[ E(\text{MSE}) = \sum E(p) \] (4.11)

4.5.1 Effect of number of hidden layers in BPA convergence

The network is trained with two hidden layers. The total number of nodes used in both the hidden layers is 14. The different combinations of number of nodes in the first hidden layer and in the second hidden layer are given in Table 4.1. The convergence rates of the network with two hidden layers are more when compared to that of convergence rates of the network with one hidden layer. When there is
only one hidden layer and the number of hidden nodes is 10, less number of epochs is required for the network to reach lowest MSE. When there are two hidden layers with 6 nodes in the first hidden layer and 8 nodes in the second hidden layer, it requires more epochs for the network to reach required MSE. Since it requires more number of iterations for the network with more than one hidden layer, it is sufficient to have only 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. It so happens, that sometimes \( \eta \) will make the network to converge to the desired MSE after an increased number of iterations. For 10 nodes in the hidden layer, it requires less iterations for the network to reach MSE of 0.01 when \( \eta \) is 0.05.

The network is trained with threshold (\( \theta \)) and without \( \theta \) is used, updation of \( \theta \) is done similar to weight updation. The parameter \( \theta \) is used in all the layers except in the input layer. For 10 nodes in the hidden layer, it requires less iterations for the network to reach MSE of 0.01 without \( \theta \), and more iterations for the network to reach MSE of 0.01 with \( \theta \).
Table 4.1 Presents number of epochs required to reach 0.01 with two hidden layers in the BPA

<table>
<thead>
<tr>
<th>No. of nodes in layer 1</th>
<th>No. of nodes in layer 2</th>
<th>No. of iterations to reach MSE of 0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10</td>
<td>102</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>74</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>85</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>95</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>124</td>
</tr>
</tbody>
</table>
4.5.2 Fixing the number of nodes in the hidden layer

The number of nodes in the hidden layer of BPA is fixed based on the maximum number of emotion images recognized. Simulation of the number of nodes in the hidden layer is changed from 2 nodes to 20 nodes. More than 20 nodes is not simulated in the BPA training.

Figure 4.3a presents the convergence curve for BPA with different number of nodes used in the hidden layer for training the FLD feature obtained from the angry images. The hidden layer with 20 nodes results in more convergence of the error. Other convergence curve indicates less convergence and hence more epochs taken to reach the desired error value.

![Fig.4.3a Epochs versus mean squared error](image-url)
4.5.3 FEE classification by BPA

Figure 4.3b shows the improvement in correct recognition of total number of Facial emotion expression considering all test images as the number of iterations increase to reach specified convergence criteria. The recognition performance of BPA for different number of nodes in the hidden layer is shown in Figure 4.3b.
Figure 4.3c shows target value for 6 categories of facial emotion expression. X-axis represents test image patterns and y-axis shows 6 FEE categories. In the testing process, the outputs of BPA do not show rounded numbers instead only fractional numbers. These fractional values are within the range of classification.
4.6 RADIAL BASIS FUNCTION NEURAL NETWORK

Radial basis function is a supervised artificial neural network. It is one of the best approximation functions which maps input and output data with less error. Due to its approximation capability, the RBF is proposed for estimation of emotional facial expression classification. RBF learns the feature patterns extracted by FLD. RBF require one iteration of training and it does not require more than one iteration of repeated training. Hence, there is no difficulty in the convergence of RBF in mapping input and output data. A bias value of 1 is appended in the hidden layer of RBF to process the RBF matrix in order to obtain final weights. These final weights are used for emotion facial expression classification.

For the association of input and output pattern values, the concept of distance measure is used. Radial Basis Functions (RBFs) are able of producing approximations to an unknown function ‘f’ from a set of input data abscissa. By passing an input point through a set of basis functions, the approximation is formed. Each of which contains one of the RBF centers. Before summing them linearly, multiply the result of each function by a coefficient.
Figure 4.4 shows a RBF network. The frame is segmented and presented to FLD. Two features are obtained using FLD. These features are presented to the input layer of the network and the target values are presented to the output layer of the network. A bias values is appended in the hidden layer to enable processing of RBF matrix. Training of the network is stopped when all the training patterns are presented to the network.
4.6.1 Training RBF

Training is a process of making ANN with RBF to learn the patterns of a data. During this process, initialization of the mathematical system is done by using a predefined procedure or by using random numbers. Repeated presentation of patterns to the system makes the system learn which is known through a set of updated connection strength called weights. These weights are further used for the process of testing the system or direct implementation for the required application.

The training procedure and the intermediate calculations are elaborated as follows:

**Input to hidden layer**

The features of each training pattern are input to the input layer of the RBF. The summation of the distance values are the inputs to the hidden layer of RBF. The values are passed over an activation function to obtain RBF values. These are the outputs of the hidden layer of RBF network.

**Hidden to output layer**

Similarly, if there are 100 training patterns, then 100 rows of RBF values are obtained.

These rows form an RBF matrix. Each column of the matrix represents a node in the hidden layer. The RBF matrix is further preceded with the target values of the training patterns to obtain final weight matrix.
4.6.2 Algorithm for Training RBF

Step 1: Apply Radial Basis Function.

No. of Inputs = 2
No. of Patterns = 100
No. of Centres = 100

Calculate RBF as

$$RBF = \exp(-X)$$

Calculate Matrix as

$$G = RBF$$
$$A = G^T \cdot G$$

Calculate

$$B = A^{-1}$$

Calculate

$$E = B \cdot G^T$$

Step 2: Calculate the Final Weight.

$$F = E \cdot D$$

Step 3: Store the Final Weights in a File.
4.6.3 Testing the RBF for FEE classification

In the testing of RBF, information flows from input layer to the hidden layer by finding the summation of the square of the distance between input pattern and the center patterns. The distance obtained before the hidden layer is passed over an exponential activation function to get outputs of nodes in the hidden layer. A bias is appended as the \((n+1)^{th}\) in the hidden layer.

The already obtained final weights between hidden layer and output layer are multiplied with the outputs of the hidden layer to obtain an output in the node of the output layer. This output is in the range of 0 and 1. The output is compared with a threshold value to decide the category of emotion.

4.6.4 Algorithm for Testing RBF

The algorithms used for testing the RBF for classify the emotion type is given as follows:

**Step 1:** Read patterns from FLD and final weights

**Step 2:** Calculate distance vector for the pattern and the centers to create input for the hidden layer of RBF.

**Step 3:** Pass the values of step 3 over exponential activation function. These values will be the outputs of the hidden layer. Append a bias value of 1.

**Step 4:** Multiply the outputs of the hidden layer with the target weights to obtain
a single value in the output layer of the RBF network.

**Step 5:** Compare the output obtained with a threshold value and decide the emotion expression type.

![EFFECT OF NUMBER OF CENTERS](image)

**Fig.4.5 Effect of number of centers in FEE classification**

Figure 4.5 shows output of RBF in identifying the FEE from the test images. The number of test images is shown in x-axis. The output categorization is shown in y-axis. The target values for each group of expression are shown through horizontal and vertical continuous line. The classification output is shown through waveline.
4.7 ECHO STATE NEURAL NETWORK FOR EMOTION FACIAL EXPRESSION CLASSIFICATION

A recurrent neural network has been proposed for emotion facial expression classification. ESNN (Jaeger et al, Atiya et al) is a recurrent neural network. It uses the concept of state vector for each training pattern and tanh activation function to squash the values in the range of -1 to +1. The inputs to ESNN are the features obtained from FLD.

The ESNN architecture shown in figure 4.6, possesses a highly interconnected and recurrent topology of nonlinear PEs that constitutes a reservoir of rich dynamics and contains information about the history of input and output patterns. The outputs of this internal PEs (echo states) are fed to a memory less but adaptive readout network (generally linear) that produces the network output. The interesting property of ESNN is that only the memory less readout is trained, whereas the recurrent topology has fixed connection weights. This reduces the complexity of RNN training to simple linear regression while preserving a recurrent topology, but obviously places important constraints in the overall architecture that have not yet been fully studied.

The echo state condition is defined in terms of the spectral radius (the largest among the absolute values of the eigenvalues of a matrix, denoted by (|| ||) of the reservoir’s weight matrix (|| W ||<1). This condition states that the dynamics of the
ESNN is uniquely controlled by the input, and the effect of the initial states vanishes. The current design of ESNN parameters relies on the selection of spectral radius. There are many possible weight matrices with the same spectral radius, and unfortunately they do not perform at the same level of mean square error (MSE) for functional approximation.

Fig. 4.6 An Echo State Neural Network with CC

The recurrent network is a reservoir of highly interconnected dynamical components, states of which are called Echo states. The memory less linear readout is trained to produce the output. The topology of ESNN consists of M input units, N internal PEs, and L output units. The value of the input unit at time n is \( u(n) = [u_1(n), u_2(n), \ldots, u_M(n)]^T \),
The internal units are \( x(n) = [x_1(n), x_2(n), \ldots, x_N(n)]^T \), and

Output units are \( y(n) = [y_1(n), y_2(n), \ldots, y_L(n)]^T \).

The connection weights are given as follows:

An \((N \times M)\) weight matrix \( W_{\text{back}} = W_{ij}^{\text{back}} \) for connections between the input and the internal PEs,

An \(N \times N\) matrix \( W_{\text{in}} = W_{ij}^{\text{in}} \) for connections between the internal PEs

An \(L \times N\) matrix \( W_{\text{out}} = W_{ij}^{\text{out}} \) for connections from PEs to the output units and

An \(N \times L\) matrix \( W_{\text{back}} = W_{ij}^{\text{back}} \) for the connections, that project back from the output to the internal PEs.

The activation of the internal PEs (echo state) is updated according to

\[
x(n + 1) = f(W_{\text{in}} u(n + 1) + Wx(n) + W_{\text{back}} y(n)), \tag{4.12}
\]

Where \( f = (f_1, f_2, \ldots, f_N) \) are the internal PEs’ activation functions.

Here, all \( f_i \)'s are hyperbolic tangent functions \( \frac{e^x - e^{-x}}{e^x + e^{-x}} \). The output from the network is computed according to

\[
y(n + 1) = f_{\text{out}}(W_{\text{out}} x(n + 1)), \tag{4.13}
\]

Where

\( f_{\text{out}} = (f_1^{\text{out}}, f_2^{\text{out}}, \ldots, f_L^{\text{out}}) \) are the output unit’s nonlinear functions (Purushothaman et al, 2008).
4.7.1 Working Principle of ESNN

4.7.1.1 Training the ESNN

The training patterns obtained from FLD are used to train the ANN topology with ESNN algorithm. In order to estimate the type of emotion, FLD features are presented in the input layer and the corresponding target outputs (1/2/3/4/5/6) are presented in the output layer of the ESNN. A state vector is initialized with zero. The length of the state vector is equivalent to number of nodes or reservoirs in the hidden layer of the ESNN. The number of reservoirs is decided based on the minimum error obtained in the ESNN in estimation of the expression. The summation of (input pattern multiplied with initial weights between input and hidden layers, multiplication of initial state vector with the initial weights of the reservoir and multiplication of target value with the initial weights between hidden layer and output layer) is obtained. A new state vector is obtained by passing the summed value over an activation function which is the tanh function. Hence, 100 state vectors are obtained if there are 100 training patterns. An ESNN matrix is obtained whose size is number of training patterns (100) X number of reservoirs (21). This matrix is a rectangular matrix and hence a pseudo inverse of the ESNN matrix is found and multiplied with the target values to obtain final weights.
Training patterns are presented to the ESNN. Initialize random weights between input layer (IL) and hidden layer (hL). Initialize weights between output layer (oL) and hidden layer (hL). Initialize weights in the reservoirs. A state vector is obtained by multiplying 2 FLD features with weight matrices and target value. Many state vectors are obtained corresponding many patterns. The pseudo inverse of the state vector matrix is multiplied with target values of the training pattern to obtain final weights. The weights are stored for testing purpose.

**Phase 1: Steps for Training ESNN**

**Step 1:** Read emotion image.

**Step 2:** Decide the number of reservoirs = 21 or 22.

**Step 3:** Decide the number of nodes in the input layer = 2.

**Step 4:** Decide the number of nodes in the output layer = number of target values = 1.

**Step 5:** Initialize state vector (number of reservoirs) = 0.

**Step 6:** Initialize random weights between input layer (IL) and hidden layer (hL).

Initialize weights between output layer (oL) and hidden layer (hL). Initialize weights in the reservoirs.

**Step 7:** Calculate state_vector\_next = tanh ((ILhL)\_weights * Input\_pattern + (hL)\_weights * state vector\_present + (hLoL)\_weights * Target\_pattern).
Step 8: Calculate, \( a = \text{Pseudo inverse (State vector all patterns)} \).

Step 9: Calculate, \( W_{\text{out}} = a \times T \) and store \( W_{\text{out}} \) for emotion facial expression classification.

4.7.1.2 Testing the ESNN for estimating the emotional facial expression classification

A pattern with two FLD features obtained is presented to the input layer of the ESNN. The summation of (input pattern multiplied with final weights between input and hidden layers + multiplication of final state vector with the final weights of the reservoir + the final weights between hidden layer and output layer) is obtained. The tanh (summation) is obtained and added with the already obtained value during training (pseudo inverse (state matrix) X target of all the patterns). The final value of the output of ESNN is compared with a threshold of (1/2/3/4/5/6/) to decide the type of emotional facial expression classification.

A state_vector is obtained by multiplying 2 FLD features with final weight matrices obtained during training. The obtained value is passed over tanh function. The resultant value is the output in the output layer.
Testing ESNN

**Step 1:** Adopt step 1 and step 2 mentioned in Training.

**Step 2:** Calculate state vector = \( \tanh ((I_{L}h_{L})_{weights} \times Input_{pattern} + (h_{L})_{weights} \times state \) \vector_{present} + (h_{L}o_{L})_{weights} \times Target_{pattern}). \)

**Step 3:** Estimated output = state vector \( \times W_{out} \).

**Step 4:** Based on output in step 4, decide the type of expression.

In order to obtain best estimation from ESNN, optimum values for different parameters of ESNN obtained.

Deciding the number of reservoirs, range of initial weights in reservoir matrix and range of initial weights between reservoir and output layer gives a good emotional facial expression classification.
4.7.1.3 Fixing the optimum number of reservoirs

![Graph showing error between estimated and actual outputs for different number of reservoirs.](image)

**Fig.4.7 Error plot for different number of reservoirs**

The nodes or reservoirs are present in the hidden layer of ESNN topology versus the error between the estimation values of ESNN and target value of ESNN is presented. The plot shows less error for different number of reservoirs (15 or 21 reservoir). The Figure 4.6 shows variations in the error for different reservoirs in the hidden layer.

The x-axis represents different number of reservoirs. The y-axis represents corresponding difference between target and the estimation by ESNN. The quality of ESNN estimation of type of expression depends upon the number of reservoirs.
4.7.1.4 Fixing the range of weights between reservoir and output layer

![Graph showing error plot for different weight ranges between reservoirs and nodes in output layer.]

**Fig. 4.8 Error plot for different weight ranges between reservoirs and nodes in output layer**

In Figure 4.8, the change of weight values and their impact in estimation of ESNN is presented when the weight normalization is done only between output layer and hidden layer (reservoirs). The error increases and decreases. Hence lesser weight range has to be used to obtain good estimation of type of emotional facial expression classification.
Fig. 4.9 Error plot for different connection weights between input layer and reservoirs

In Figure 4.9, the change of weight values and their impact in estimation of ESNN is presented when the weight normalization is done only between input layer and hidden layer (reservoirs). The error increases and decreases continuously. The x-axis represents the change in the weight values in input and hidden layer. The y-axis represents corresponding difference between target and the estimation by ESNN. The weight should be in the range of 0.5-0.6 for increased accuracy of estimation of type of emotional facial expression classification.
Fig. 4.10 Error plot for weight ranges in lateral connections in the reservoir

In Figure 4.10, the change of weight values and their impact in estimation of ESNN is presented when the weight normalization is done only in reservoirs. The error increases and decreases continuously. The x-axis represents the change in the weight values in hidden layer. The y-axis represents corresponding difference between target and the estimation by ESNN. The weight values should be less than 0.3 for increased accuracy of estimation of type of expression.
Figure 4.11 shows output of ESNN in identifying the FEE from the test images. The number of test images is shown in x-axis. The output categorization is shown in y-axis. The target values for each group of expression are shown through horizontal and vertical continuous line. The classification output is shown through waviline.
4.8 COUNTER PROPAGATION NETWORK FOR FACIAL EXPRESSION ESTIMATION

Counter-propagation neural network was developed by Robert Hecht-Nielsen as a means to combine an unsupervised Kohonen layer with a teachable output layer known as Grossberg layer, by James et al. (1991). The operation of this network type is very similar to that of the learning vector Quantization (LVQ) network in that the middle (Kohonen) layer acts as an adaptive look-up table.

The Figure 4.12 gives the flowchart of the CPN. The figure shows counter-propagation network with three layers: an input layer that reads input patterns from the training set and forwards them to the network, a hidden layer that works in a competitive fashion and associates each input pattern with one of the hidden units, and the output layer which is trained via a teaching algorithm that tries to minimize the mean square error (MSE) between the actual network output and the desired output associated with the current input vector. In some cases a fourth layer is used to normalize the input vectors but this normalization can be easily performed by the application before these vectors are sent to the Kohonen layer.

Regarding the training process of the counter-propagation network, it can be described as a two-stage procedure: in the first stage, the process updates the weights of the synapses between the input and the Kohonen layer, while in the
second stage the weights of the synapses between the Kohonen and the Grossberg layer are updated.

![Flow chart for Counter propagation network](chart.png)
4.8.1 Training the weights from the input to the hidden nodes

**Step 1:** The synaptic weights of the network between the input and the Kohonen layer are set to small random values in the interval [0, 1].

**Step 2:** A vector pair \((x, y)\) of the training set, is selected in random.

**Step 3:** The input vector \(x\) of the selected training pattern is normalized.

**Step 4:** The normalized input vector is sent to the network.

**Step 5:** In the hidden competitive layer the distance between the weight vector and the current input vector is calculated for each hidden neuron \(j\) according to the equation

\[
D_j = \sqrt{\sum_{i=1}^{K} (x_j - w_{ij})^2} 
\]  

(4.14)

where

- \(K\) is the number of the hidden neurons and
- \(w_{ij}\) is the weight of the synapse that joins the \(i^{th}\) neuron of the input layer with the \(j^{th}\) neuron of the Kohonen layer.

**Step 6:** The winner neuron \(W\) of the Kohonen layer is identified as the neuron with the minimum distance value \(D_j\).

**Step 7:** The synaptic weights between the winner neuron \(W\) and all \(M\) neurons of the input layer are adjusted according to the equation

\[
w_{wi}(t+1) = w_{wi}(t) + \alpha(t)(x_i - w_{wi}(t))
\]  

(4.15)
In the above equation the coefficient is known as the Kohonen learning rate.

The training process starts with an initial learning rate value 0.0 that is gradually decreased during training according to the equation

$$\alpha(t) = \alpha_o (1 - \frac{t}{T})$$  \hspace{1cm} (4.16)

Where

$T$ is the maximum iteration number of the stage A of the algorithm.

A typical initial value for the Kohonen learning rate is a value of 0.7.

**Step 8:** The steps 1 to 6 are repeated until all training patterns have been processed once. For each training pattern $p$ the distance $D_p$ of the winning neuron is stored for further processing. The storage of this distance is performed before the weight update operation.

**Step 9:** At the end of each epoch the training set mean error is calculated according to the equation

$$E_i = \frac{1}{P} \sum_{k=1}^{P} D_k$$  \hspace{1cm} (4.17)

Where

$P$ is the number of pairs in the training set,

$D_k$ is the distance of the winning neuron for the pattern $k$ and $i$ is the current training epoch.
The network converges when the error measure falls below a user supplied tolerance value. The network also stops training in the case where the specified number of iterations has been performed, but the error value has not converged to a specific value.

4.8.2 Training the weights from the hidden to the output nodes

Step 1: The synaptic weights of the network between the Kohonen and the Grossberg layer are set to small random values in the interval [0, 1].

Step 2: A vector pair (x, y) of the training set, is selected in random

Step 3: The input vector x of the selected training pattern is normalized.

Step 4: The normalized input vector is sent to the network

Step 5: In the hidden competitive layer the distance between the weight vector and the current input vector is calculated for each hidden neuron j according to the equation

\[ D_j = \sqrt{\sum_{i=1}^{K} (x_j - w_{ij})^2} \]  \hspace{1cm} (4.18)

Where

- K is the number of the hidden neurons and
- \( W_{ij} \) is the weight of the synapse that joins the ith neuron of the input layer with the jth neuron of the Kohonen layer.
Step 5: The winner neuron $W$ of the Kohonen layer is identified as the neuron with the minimum distance value $D_j$. The output of this node is set to unity while the outputs of the other hidden nodes are assigned to zero values.

Step 6: The connection weights between the winning neuron of the hidden layer and all $N$ neurons of the output layer are adjusted according to the equation

$$V_{jw}(t+1) = V_{jw}(t) + \beta (y_j - V_{jw}(t)) \quad (4.19)$$

In the above equation the $\beta$ coefficient is known as the Grossberg learning rate.

Step 7: The above procedure is performed for each training pattern. In this case the error measure is computed as the mean Euclidean distance between the winner node's output weights and the desired output, that is

$$E = \frac{1}{P} \sum_{j=1}^{N} D_j = \frac{1}{P} \sum_{j=1}^{P} \sum_{k=1}^{N} \sqrt{(y_k - w_{kj})^2} \quad (4.20)$$
Figure 4.13 shows flow of information from input to hidden to output layers. The frame is segmented and presented to FLD. Two features are obtained using FLD. These features are presented to the input layer of the network and the target values are presented to the output layer of the network. Training of the network is stopped when all the training patterns are presented to the network.
4.9 EFFECT OF NUMBER OF NODES IN HIDDEN LAYER OF CPN NETWORK

Figure 4.14 presents bar chart showing the outputs of CPN for different number of nodes in the hidden layer of CPN topology.

Fig.4.14 Number of nodes in the hidden layer of CPN
Figure 4.15 shows the outputs CPN for classification of FEE. The red color shows the target value used for training the network. The outputs during testing are shown in green color.
Figure 4.16 shows output of CPN in identifying the FEE from the test images. The number of test images is shown in x-axis. The output categorization is shown in y-axis. The target values for each group of expression are shown through horizontal and vertical continuous line. The classification output is shown through waveline.
4.10 SUMMARY

This chapter has presented implementation of BPA/RBF/ESNN/CPN for recognizing FEE. Chapter 5 presents results and discussion about the performance of the proposed algorithms.