CHAPTER 3
CONCURRENCY CONTROL USING FUNCTIONAL BACKPROPAGATION ALGORITHM (FUBPA)

3.1 INTRODUCTION
This chapter discusses the proposed FUBPA for training the ANN. The functioning of FUBPA and its convergence behavior are presented.

3.2 ARTIFICIAL NEURAL NETWORK
A neural network is constructed by highly interconnected processing units (nodes or neurons) which perform simple mathematical operations Fortuna et al,1992. Neural networks are characterized by their topologies, weight vectors, and activation function which are used in the hidden layers and output layer Lippmann 1987. The topology refers to the number of hidden layers and connection between nodes in the hidden layers. The activation functions that can be used are sigmoid, hyperbolic tangent and sine Purushothaman et al, 1994. The network models can be static or dynamic. Static networks include single layer perceptrons and multilayer perceptrons. A perceptron or adaptive linear element (ADALINE) Bernard Widrow, 1990 refers to a computing unit. This forms the basic building block for neural networks. The input to a perceptron is the summation of input pattern vectors by weight vectors. In figure 3.1, the basic function of a single layer perceptron is shown.

In figure 3.2, a multilayer perceptron is shown schematically. Information flows in a feed-forward manner from input layer to the output layer through hidden layers. The number of nodes in the input layer and output layer is fixed. It depends upon the number of input variables and the number of output variables in a pattern. In this research work, there are 1
input variable and three output variable. The number of nodes in a hidden layer and the number of hidden layers are variable. Depending upon the type of application, the network parameters such as the number of nodes in the hidden layers and the number of hidden layers are found by trial and error Hirose et al, 1991.

Fig. 3.1 Single layer perceptron
3.2.1 Disadvantages of Steepest-Descent Method

The number of cycles required for the error value $E$ to reach the desired minimum is very large. The error value $E$ does not reach the desired minimum due to some local minima whose domains of attraction are as large as that for the global minimum. The algorithm converges to one of those local minima and hence learning stops prematurely or the value diverges. The updating of weights will not stop unless every input is outside the significant update region. The significant update region is from 0.1 to 0.9. Due to this, the output of the network will be approaching either 0.0 or 1.0. Thus a large number of iterations are required for the convergence of the algorithm.
3.3 FUNCTIONAL UPDATE BACK PROPAGATION ALGORITHM

The concept of functional update method indicates that there will be weight updating of the connection between layers only if at least one node in the output layer is misclassified. The meaning of misclassification is valid when the difference between the target and the actual value of node in the output layer is more than 0.5.

By converting the analog patterns into binary patterns, the convergence and classification behavior change. When binary patterns are presented in BPA with modification in the convergence condition, the algorithms is called FUBPA.

The flowchart for BPA with functional update is given in figure 3.3. The number of layers and the number of nodes in the hidden layers are decided. The weights between layers, for hidden and output layers are initialized. A training pattern is presented to the input layer of the network and the difference between the network’s output and the target output is calculated for each node in the output layer. If the data is analog, the number of nodes in the output layer is 1. If the data is binary, the number of nodes in the output layer is more depending upon binary conversion. If the difference obtained for each node is greater than the value of functional criteria, a counter is incremented and the weights and thresholds are updated. If the difference of not even one node is greater than 0.5 then the weights and thresholds are not updated. The MSE of the network for each pattern is calculated, when at least one node in the output of the network is misclassified. Remaining training patterns are presented to the network. Training of the network is stopped, when a performance index of the network is reached.
The algorithm for the functional update is as follows:

**Step 1:** The weights of the network are initialized.

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

**Step 3:** The output of each node in the successive layers is calculated by

\[ X_i = \frac{1}{1 + \exp \left( \sum W_{ij} X_i \right)} \]  \hspace{1cm} \text{(3.1)}

Where \( i = 1 \) to 3

\( X_i \) represents no. of nodes in the o/p layer

**Step 4:** The number of nodes in the output layer, which are misclassified, are denoted by ‘nm’. A node is misclassified, if it does not satisfy the equation.

\[ 1 - \varepsilon > D \geq 0.5 \]  \hspace{1cm} \text{(3.2)}

Where

\( \varepsilon \) is the value fixed by the programmer, and

\( D = | \text{Desired output - Network output} | \)

If ‘nm’ is empty, i.e. Not even one node in the outer layer satisfy Equation (3.3), go to step 2.

**Step 5:** If ‘nm’ is not empty, the objective function ‘j’ is computed by

\[ J = \frac{1}{2} \sum_{\text{X\in nm}} \sum D^2 \]  \hspace{1cm} \text{(3.3)}

**Step 6:** The weights and thresholds are updated.

**Step 7:** The steps (2-6) are repeated, until the total MSE of all the patterns is below a specified value.

The main advantage of FUBPA is that it will stop as soon as the misclassified set is empty.
Read number of layers, number of nodes, Initialize the weights

Read a pattern and calculate the outputs of nodes in the successive layers

\[ D_i = d_i - X_i \], If \( D_i > 0.5 \), Increment the counter ‘nm’

Is ‘nm’ empty?

Calculate \( E(p) \) of a pattern, Calculate \( \delta \) of a node in the output

Update weights between layer

For all layers?

Calculate \( \delta \) of a node in the hidden layer

Calculate MSE, \( E \) for all the patterns

All the patterns?

\( E < \) specified value?

Stop

Fig. 3.3 Flowchart for FUBPA
3.4 ADVANTAGES OF FUBPA OVER CONVENTIONAL BPA

When the network is trained with analog data by conventional update method the number of iterations is large for the objective function \((J)\), to reach the desired MSE. The objective function does not reach the desired MSE due to some local minima, whose domains of attractions are as large as that for the global minimum. The network converges to one of those local minima or the network diverges. The updating of the weights will not stop, unless every input is outside the significant update region (0.1 to 0.9) and the outputs of the network will be approaching either 0 or 1. This requires much iteration for the network to converge. To overcome these difficulties a functional criterion that results in faster convergence of the network is used.

3.5 TRAINING STRATEGIES FOR THE NETWORK

For the network to learn the patterns, different weight updating algorithms have been developed. They are called supervised methods and unsupervised methods. Since both the inputs and outputs are considered for locking strategies of concurrency control, supervised learning technique has been used. The present work involves modification of existing weight updation algorithm, combination of classical method with neural network, method of training the network.

3.6 IMPLEMENTATION OF INTELLIGENT LOCKING

Inbuilt library functions for the Fork base are available in standard CAD software, Brodie et al. (1984). In Figure 2.1, the Fork base and its entities are shown schematically. The Fork base drawing (Figure 2.1) is used to manufacture one variety of Fork base that has to be used to clamp two plates. This drawing file will be accessed by many designers who will choose their choice of designs and the designs are stored in the same location of the
server. Each designer can choose their option of changing the shape, dimensions of different components of the subassembly of the Fork base. When any modification is done for one subassembly, due to associative dimensioning concept, the dimensions of the entire Fork base shape and dimensions can change. Similarly the Bolted connection (Figure 2.3) and Bearing (Figure 2.5) have been considered for analysis of concurrency efficiency using back propagation algorithm. If it is associative dimensioning, then all the changes in shape and size of the Fork base system have to be updated for a small change in the dimension of the subassembly. At the same time, another designer would want to retain earlier version of the drawing. When more than one transaction adopts similar changes in the file, and when they commit at the same time, how to maintain the consistency. In Table 3.1, the variables used to represent locks for objects for training the ANN is given.

| Table 3.1 Variables considered for training and testing of ANN for lock management |
|---------------------------------|---------------------------------|
| **Object**                      | **Lock mode**                  |
| where                           |                                 |
| Object represents the entire Manufacturing related file or an entity in the file |
| Mode represents type of lock assigned to an object. |
| exclusive (X) mode. Data item can be both read as well as written. |
| shared (S) mode. Data item can only be read. |
| intention-shared (IS): indicates explicit locking at a lower level of the tree but only with shared locks. |
| intention-exclusive (IX): indicates explicit locking at a lower level with exclusive or shared locks |


shared and intention-exclusive (SIX): the sub tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.

Intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes.

In Table 3.2, column 1 represents the lock type. Column 2 represents the value to be used in the input layer of the ANN in module 1 and module 2. Column 3 gives binary representation of Lock type to be used in the output layer of module 1. The values are used as target outputs in the module 1 and module 2 during lock release on a data item. The modules of algorithm which work using FUBPA are given in Table 3.3. The modules given in Table 3.3 give their usage for learning and finding the lock states: OL (Object, Lock).

<table>
<thead>
<tr>
<th>Lock type</th>
<th>(Input layer representation numerical value).</th>
<th>Binary representation in target layer of the ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Not locked</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>IS</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>IX</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>SIX</td>
<td>5</td>
<td>101</td>
</tr>
</tbody>
</table>
Table 3.3. Modules used for learning the lock status of an object

<table>
<thead>
<tr>
<th>Module</th>
<th>Name</th>
<th>Training / Testing</th>
<th>ANN Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OL</td>
<td>Training</td>
<td>1{object id} x {no. of nodes in hidden layer} x 3{Lock value}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{Figure 3.4 }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{Forward and Reverse}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Testing</td>
<td>1{object id} x {no. of nodes in hidden layer} x 3{Lock value}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{Figure 3.5 }</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{Reverse}</td>
<td></td>
</tr>
</tbody>
</table>

In the fourth column of this table, 3 values are given in the order, no. of nodes in the input layer, no. of nodes in the hidden layer which can be anything and no. of nodes in the output layer which is 3(fixed).
Fig. 3.4 OL Training

Fig. 3.5 OL Testing
3.7 SEQUENCE OF MODULES EXECUTED WHEN A TRANSACTION REQUESTS LOCK (FUBPA TRAINING) OR RELEASES LOCK (FUBPA TESTING)

The step 1 to step 4 for case 1, case 2 and case 3 are executed to produce performance metrics.

1. Initialize randomly the weights of OL training (Figure 3.4)
2. A transaction Ti requests lock on objects (O1, O2, … On)
3. OL testing (Figure 3.5) is executed with objects (O1, O2, … On) in step 2 to obtain binary value. If ‘000’ is obtained in the output layer, then the object(s) can be locked. If (001, 010, 011, 100) is obtained in output layer of OL testing, then the object(s) is under use.
4. In any case, if transaction Ti is requested on object Oi, then OL training update weights inclusive of new patterns using the back propagation algorithm (forward and backward steps).
5. In any case, if the object is under any lock mode other than shared or no lock, then the transactions are kept under queue.

3.8. RESULTS AND DISCUSSIONS

The proposed functional update back propagation algorithm for lock state learning and lock state finding have been implemented using Matlab 7. OL training is trained until a Mean Square Error (MSE) value of 0.001 is reached. The time for convergence to reach 0.001 is at an average of 3sec (since Matlab is used. If any compiler like turbo c has been used, then the time would be almost 1/3 of the time mentioned). The Figure 3.6 shows number of iterations versus mean squared error for OL training using BPA.
Figure 3.7 to Figure 3.18 show the performances of CC during locking and releasing for objects. The performances have been evaluated using FUBPA under controlled simulation environment. The releasing time, locking time for various transactions of a drawing depends on the processor capabilities, speed of the RAM and memory occupation.

![Graph of Mean Squared Error (MSE) convergence](image)

**Fig. 3.6 Mean squared error convergence**
3.8.1 FORK BASE DRAWING

**Fig. 3.7** Releasing time for each object in Fork base

**Fig. 3.8** Locking time for each object in Fork base
Fig. 3.9 Total Locking time for each transaction group-in Fork base

Fig. 3.10 Total Releasing time for each transaction group-in Fork base
3.8.2 BOLTED CONNECTION DRAWING

**Fig. 3.11** Releasing time for each object in Bolted connection

**Fig. 3.12** Locking time for each object in Bolted connection
Fig. 3.13 Total Locking time for each transaction group in Bolted connection

Fig. 3.14 Total Releasing time for each transaction group in Bolted connection
3.8.3 BEARING DRAWING

**Fig. 3.15 Releasing time for each object in Bearing**

**Fig. 3.16 Locking time for each object in Bearing**
Fig. 3.17 Total Locking time for each transaction group in Bearing

Fig. 3.18 Total Releasing time for each transaction group in Bearing
3.9 SUMMARY

This chapter has presented the different types of locks used for a transaction. The sequence in which FUBPA modules used for learning the object, transaction, lock type and the method of finding out if lock has been assigned for a transaction through testing of FUBPA are given. Performance metrics are presented. If the time gap between transactions are longer, then the time taken for training will not affect locking and unlocking. If the time between transaction is less, then FUBPA is not suitable. Chapter 4 discusses the implementation of Locally Weighted Projection Regression (LWPR) for providing locking and unlocking of transactions for various drawing objects in Fork base, Bolted connection and Bearing.