CHAPTER 5
PASS WORD AUTHENTICATION USING BIDIRECTIONAL ASSOCIATIVE MEMORY

5.1 Introduction:

The password authentication using HPNN takes the input and the output pattern together to the network. There is a chance of wrong user validation. In order to eliminate the limitations in PAS using HPNN in this chapter introduces new technique i.e. password authentication using bidirectional associative memory.

5.2 Bidirectional Associative Memory (BAM):

BKosko (1988) extended the Hopfield model by incorporating an additional layer to perform recurrent auto-associations as well as hetero-associations on the stored memories. The network structure of the Bi-directional Associative Memory model[9,50] is similar to that of the linear associator, but the connections are bidirectional, i.e., \( w_{ij} = w_{ji} \), for \( i = 1, 2, ..., m \) and \( j = 1, 2, ..., n \). Also, the units in both layers serve as both input and output units depending on the direction of propagation. Propagating signals from the X layer to the Y layer makes the units in the X layer act as input units while the units in the Y layer act as output units. The same is true for the other direction, i.e., propagating from the Y layer to the X layer makes the units in the Y layer act as input units while the units in the X layer act as output units. Below is an illustration of the BAM architecture[8].

Just like the linear associator and Hopfield model, encoding in BAM can be carried out by using: to store a single associated pattern pair and
In BAM, decoding involves reverberating distributed information between the two layers until the network becomes stable. In decoding, an input pattern can be applied either on the X layer or on the Y layer. When given an input pattern, the network will propagate the input pattern to the other layer allowing the units in the other layer to compute their output values. The pattern that was produced by the other layer is then propagated back to the original layer and let the units in the original layer compute their output values. The new pattern that was produced by the original layer is again propagated to the other layer. This process is repeated until further propagations and computations do not result in a change in the states of the units in both layers where the final pattern pair is one of the stored associated pattern pairs. The final pattern pair that will be produced by the network depends on the initial pattern pair and the connection weight matrix.

\[
W_k = X_k^T Y_k 
\]

\[
W = \alpha \sum_{k=1}^{p} W_k 
\]

**Figure 5.1. BAM Architecture**
Several modes can also be used to update the states of the units in both layers namely synchronous, asynchronous, and a combination of the two. In synchronous updating scheme, the states of the units in a layer are updated as a group prior to propagating the output to the other layer. In asynchronous updating, units in both layers are updated in some order and output is propagated to the other layer after each unit update. Lastly, in synchronous-asynchronous updating, there can be subgroups of units in each layer that are updated synchronously while units in each subgroup are updated asynchronously.

Since the BAM also uses the traditional Hebb's learning rule to build the connection weight matrix to store the associated pattern pairs, it too has a severely low memory capacity. The BAM storage capacity for reliable recall was given by Kosko (1988) to be less than minimum \((m, n)\), i.e., the minimum of the dimensions of the pattern spaces. A more recent study by Tanaka et al (2000) on the relative capacity of the BAM using statistical physics reveals that for a system having \(n\) units in each of the two layers, the capacity is around 0.1998\(n\).

### 5.3 Authentication using Bidirectional Associative Memory:

The Bidirectional associative memory (BAM) is **heteroassociative**, **content-addressable** memory. A BAM consists of neurons arranged in two layers say A and B. The neurons are bipolar binary. The neurons in one layer are fully interconnected to the neurons in the second layer. There is no interconnection among neurons in the same layer. The weight from layer A to layer B is same as the weights from layer B to layer A. Dynamics involves two layers of interaction. Because the memory process information in time and involves Bidirectional data flow, it contradicts in principle from a linear association, although both networks are used to
store association pairs. It also differs from the recurrent auto associative memory in its update mode.

5.3.1 Authentication Process:

This method can use any one of the password among Text password or Graphical passwords and can train network, so that it can authenticate the authorized users.

Figure 5.2 User Validation Using BAM
**Text password:**

First this method converts the username and password into binary values and the uses those values as training samples, which can be performed by the following steps

- Convert each character into a unique number (for example ASCII value)
- Convert the unique number into binary value

![Figure 5.3 Converting Character in to Binary Values](image)

This procedure converts all the characters in the username and password into binary values.

<table>
<thead>
<tr>
<th>Username</th>
<th>Binary value representing username</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAJESH</td>
<td>001001010100000100101010101000101100101001100101</td>
</tr>
<tr>
<td>SWAMY</td>
<td>0110010101110101010000010101100101001101</td>
</tr>
<tr>
<td>KIRAN</td>
<td>0110100101001001001001010100000100111001</td>
</tr>
</tbody>
</table>

*Table 5.1 Binary Values for the Given User Name*

After converting username and password into binary equivalents the pairs can be used as training samples. Once the training has been completed very soon the network will be stored in each server. When the user wants to get service from a server he/she submits user name and password to the server, then server loads network and generates output by giving username as input. If the output matches with the password
submitted by the user then server provides service. The application can provide better authentication by using bipolar input instead of binary input. Application converts a binary number into bipolar number by using following formula or by simply replacing zeros with -1s. If \( X \) is a binary digit then corresponding bipolar value is \((2X-1)\).

\[
\begin{align*}
1 & \rightarrow 1 \\
0 & \rightarrow -1
\end{align*}
\]

The above procedure will reinforce in converting binary value in to bipolar value and can be used it as input to the network.

<table>
<thead>
<tr>
<th>Username</th>
<th>Binary value representing username</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAMY</td>
<td>-111-1-11-11-1111-11-11-1-1-11-1-11-1-11-1-11-1-11-1-11-1-11-1</td>
</tr>
</tbody>
</table>

*Table 5.2. Bipolar Values for the Given User Name*

### 5.3.2 Learning:

When a new user is fascinated to create an account, network has to adjust weights so that it can recognize all the users who are registered. This process of changing weights is called learning.

**Learning in Bidirectional Associative Memory:**

Suppose we wish to store the binary (bipolar) patterns \((A_1, B_1)\)… \((A_m, B_m)\) at or near local energy minima. How can these association pairs be encoded in some BAM n-by-p matrix M.

The association \((A_i, B_i)\) can be viewed as a meta-rule or set-level logical implication: IF \( A_i \) THEN \( B_i \). However, bidirectionality implies that \((A_i, B_i)\) also represents the converse meta-rule: IF \( B_i \) THEN \( A_i \). Hence the
logical relation between $A_i$ and $B_i$ is symmetric, namely, logical implication (set equivalence). The vector analogue of this symmetric biconditionality is correlation. The natural suggestion then is to memorize the association $(A_i, B_i)$ by forming the correlation matrix or vector outer product $A_i^T B_i$. The correlation matrix redundantly distributes the vector information in $(A_i, B_i)$ in a parallel storage medium, a matrix. The next suggestion is to superimpose the $m$ associations $(A_i, B_i)$ by simply adding up the correlation matrices point wise.

$$M = \sum_i A_i^T B_i$$  \hspace{1cm} (5.3)

With dual BAM memory $M^T$ given by

$$M^T = \sum_i (A_i^T B_i)^T = \sum_i B_i^T A_i$$  \hspace{1cm} (5.4)

5.3.3 BAM Implementation:

```c
int NoOfPatterns, NoOfBitsPerInput, NoOfBitsPerOutput;
int[,...] Weight, input, output, test;
```

Here $NoOfPatterns$ specifies the no of patterns we want to use in training, $NoOfBitsPerInput$ determines no of bits we want to use for each input, $NoOfBitsPerOutput$ specifies no of bits we want to use for each output, $Weight$ stores weight values of the network, $input$ stores input vector, $output$ stores output vector and $test$ stores pattern used for testing the network.

5.3.4 Implementing Training:

Before training, application will take training samples from the user and stores them in the corresponding variables.
The basic training procedure

Consider \( N \) training pairs \( \{(A_1, B_1), (A_2, B_2), \ldots, (A_N, B_N)\} \) where \( A_i = (a_{i1}, a_{i2}, \ldots, a_{im}) \) and \( B_i = (b_{i1}, b_{i2}, \ldots, b_{im}) \) and \( a_{ij}, b_{ij} \) are either in ON or OFF state. Where in binary mode, ON = 1 and OFF = 0 and in bipolar mode, ON = 1 and OFF = -1.

The original weight matrix of the BAM is \( X_i \)

\[
M_0 = \sum_{i=0}^{N} [X_i]^T [Y_i] \quad (5.5)
\]

where \( X_i = (X_{i1}, X_{i2}, \ldots, X_{im}) \) and \( Y_i = (Y_{i1}, Y_{i2}, \ldots, Y_{ip}) \) and \( X(Y_{ij}) \) is the bipolar form of \( a_{ij}(b_{ij}) \)

```csharp
private void Train()
{
    Weight = new int[input.GetLength(1), output.GetLength(1)];
    for (int i = 0; i < input.GetLength(0); i++)
    {
        for (int j = 0; j < input.GetLength(1); j++)
        {
            for (int k = 0; k < output.GetLength(1); k++)
                Weight[j, k] += input[i, j] * output[i, k];
        }
    }
}
```

**Note:** Here instead of calculating the transpose we multiply the required elements, which will be multiplied when we calculate transpose.
5.3.5 Recognizing the Pattern using BAM:

The pattern which we want to use for testing the network will be supplied as input to the application and then application stores the pattern in the corresponding variable.

The methods and the equations for retrieve are:

Start with an initial condition which can be any given pattern pair \((\alpha , \beta)\). Determine a finite sequence of pattern pairs \((\alpha^1 , \beta^1)\), \((\alpha^{11} , \beta^{11})\), until an equilibrium point \((\alpha_f , \beta_f)\) is reached, where

\[
B = \nabla (A M) \quad \text{and} \quad A^1 = \nabla (B^1 M^T) \tag{5.6}
\]

\[
B^{11} = \nabla (A^1 M) \quad \text{and} \quad A^{11} = \nabla (B^{11} M^T) \tag{5.7}
\]

\[
\nabla (F) = G = g_1, g_2, …, g_r \tag{5.8}
\]

\[
F = (f_1, f_2, …, f_r) \tag{5.9}
\]

\(M\) is correlation matrix

\[
g_i = \begin{cases} 
1 & \text{if} \quad f_i > 0 \\
0 \text{ (binary)} & , \quad f_i < 0 \\
-1 \text{ (bipolar)} & 
\end{cases} \tag{5.10}
\]

Previous \(g_i\), \(f_i = 0\)
private int[,] Recognize()
{
    // pa --> Previous alpha
    int[,] a = null, pb = null, b = null;
    a = test;
    do{
        pb = Phi(MatrixMul(a, Weight));
        a = Phi(MatrixMul(pb, Transpose(Weight)));
        b = Phi(MatrixMul(a, Weight));
    }while(!areEqual(pb,b));
    ShowMatrix(pb);
    return pb;
}
5.4 Results:

BAM for Textual Passwords:

![Figure 5.4 Screen showing how to setup network](image)

In the figure 5.4 *No of Patterns* enumerates the number of patterns we need to use in training, *No Of Bits Per Input* specifies number of bits desired to use for each input and *No Of Bits Per Output* stipulates number of bits to use for each output.

Once the required information has given, *OK* button has to be pressed in the application which provides enough fields to enter input and output pairs.
Training the Network:

Once the required training set has given to the above process, to make the application to undergo training process press **Train** button.
After the training is completed it will be manifested weight matrix.

**Figure 5.7 Screen showing Completion of BAM Training**

**User Authentication Using BAM**

**Figure 5.8 Screen showing Completion of BAM Authentication**

Output of Network (Password)

User Name

Password
Here a comparison can be done to the output of the network with the password given by the user and, if both are the same then the user is a valid user which gives him an opportunity to get serviced.

**BAM For Graphical Passwords:**

There is no chance of giving an image as input to the network. So we have to alter image into text.

![Image Conversion Demo](image_conversion.gif)

*Figure 5.9 Screen showing image conversion of BAM*

In figure 5.9  **Path** indicates path of the image that has to convert,**Selected Image** shows the selected image, **Resolution** specifies resolution of the selected image ,**O/P Matrix Size** specifies size of the output matrix.

After an image has been picked out ,it will be displayed in the **Selected Image** box and conversion buttons will be enabled next  the conversion takes place as mentioned in the Chapter 4,section 4.5.
5.5 Conclusion:

In this chapter password authentication using BAM is introduced to overcome the limitations of *Password Authentication Scheme* using HPNN. In password authentication using BAM we need not give input and output together to train the network. Since it is bidirectional in nature we can further implement this method to give username if we give password. But this may have some limitation that different usernames may have the same password. In order to solve this we can take password + any unique image as input for identifying the username, if the same combination was given as input while training network.