CHAPTER 6
PASS WORD AUTHENTICATION USING BRAIN -STATE -IN -A BOX (BSB) MODEL

6.1 Introduction:

This chapter introduces new technique to overcome the limitations of present password techniques. So before introducing this approach of authentication let us introduce some basics of Brain-State-in -Box (BSB).

6.1.1 Brain -State -In -A Box Model:

The "Brain-State-In-A-Box" [9, 51] (BSB) model is one of the earliest Dynamic Associative Memories (DAM) models. It is a discrete-time continuous-state parallel updated DAM. The BSB model extends the Linear Associator model and is similar to the Hopfield Model in that it is an Auto-associative model with its connection matrix computed using outer products in the usual way. The operation of both models is also very similar, with differences arising primarily in the way activations are computed in each iteration, and in the signal function used. The BSB model stands apart from other models in its use of the linear threshold signal function.

- **Activation Pattern:** \( x(t) = [x_1(t), ..., x_d(t)] \)
- **BSB Algorithm \((W,b_r)\):** \( x(t) \rightarrow x(t+1) \) where

\[
x_{r}(t+1) = \begin{cases} 1 & \text{if } u_r(t) > +1 \\ 0 & \text{if } |u_r(t)| \leq 1 \text{ and } u_r(t) = x_r(t) + \gamma \left( \sum_{j=1}^{d} w_{jr} x_j(t) + b_j \right) \\ -1 & \text{if } u_r(t) < -1 \\ \end{cases}
\]

- **Connection Matrix:**

\[
W = \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{d1} & \cdots & w_{dd} \end{bmatrix}
\]

\( w_{jr} = w_{jr} \) (symmetric \( W \))

![Figure 6.1 Working of BSB](image-url)
6.2 User Authentication Using BSB:

The architecture of the brain state in a box (BSB) consists of one layer of units that connect to themselves as illustrated in Figure 1. The connection weights between units are bidirectional and symmetric. The units may be fully connected, as illustrated in the figure, or only partially connected by randomly setting some of the weights to 0. Anderson and his colleagues have frequently used 50%, or less, connectivity. Partial connectivity does not qualitatively affect the network performance, but reduces computational time, and provides some increase in biological realism (Anderson, 1995).

![Figure 6.2. Architecture of BSB](image)

![Figure 6.3. Validating the User](image)
6.2.1 Process of Authentication using BSB:

The process of authentication uses any one among Textual password or Graphical password as password and the training will be given to the network so that it can authenticate users.

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 6.4. Converting Character into Binary Values](image)

By using above procedure we can convert all 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>SASTRY</td>
<td>011001010100000101100101000101010010010101001101</td>
</tr>
<tr>
<td>VAMSY</td>
<td>0011010101000001010110010110010101001101</td>
</tr>
<tr>
<td>SAI</td>
<td>011001010100000101001001010011010010001101</td>
</tr>
</tbody>
</table>

*Table 6.1 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 method can be enhanced for better authentication by using bipolar input instead of binary input. We can convert 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 \((2X-1)\).

\[
1 \rightarrow 1 \\
0 \rightarrow -1
\]

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>
</table>

*Table 6.2. Bipolar Values for the Given User Name*

**BSB Learning:**

Whenever new users are creating accounts 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 Brain State in a Box Model

First, the learning phase establishes the weights for each connection between units of the auto-associative memory. Using either standard Hebbian or Windrow-Hoff techniques, each representation of a multiplication number fact is associated with itself.

![Diagram of Brain State in a Box Model](image)

**Figure 6.5 Implementation details**

```c
int NoOfPatterns = 0;
int NoOfBitsPerPattern = 0;
int[ , ] Weight,Input;
```

In this method **NoOfPatterns** specifies the number of patterns to use in training, **NoOfBitsPerPattern** specifies number of bits to use for each pattern, **Weight** stores weight values of the network, **Input** Stores input vector and **Implementing Training**. Before training, application takes the training samples from the user and stores them in the corresponding variables.

In Training, the NxN connection matrix $A$ is modified as

$$\Delta A = lr \times (X - AX) \otimes X \quad (6.1)$$
\[ A = A + \Delta A \quad (6.2) \]

Where

- \( X \) is the normalized input training pattern;
- \( \text{lr} \) is the Learning rate;
- \( \otimes \) is the outer product of two vectors;

```csharp
private void Train()
{
    for (int i = 0; i < Input.GetLength(0); i++)
    {
        int[,] pattern = MatrixMath.GetRow(Input, i);
        int[,] temp1 = MatrixMath.Multiply(Weight, pattern);
        int[,] temp2 = MatrixMath.Subtract(pattern, temp1);
        int[,] dw = MatrixMath.Multiply(temp2, pattern);
        dw = MatrixMath.ScalarMultiply(lr, Weight);
        Weight = MatrixMath.Add(Weight, ContributionMatrix);
    }
}
```

**Recognizing the Pattern using BSB**

Here the pattern which is used for testing the network will be supplied as input to the application and then application stores the pattern in the corresponding variable. The equation 6.3 is used to calculate the output of the network.

\[ X_{[n+1]} = f (\gamma X_n + \eta WX_{[n]} + \delta X) \]
Where $f(x)$ is defined as follows.

\[
    f(x) = \begin{cases} 
    -1, & \text{for } x < -1 \\
    x, & \text{for } -1 \leq x \leq 1 \\
    +1, & \text{for } x > +1 
    \end{cases} \quad (6.3)
\]

This is implemented as follows

```csharp
private void Recognize()
{
    try {
        int[,] pattern = CreateMatrix(tableLayoutPanel2);
        int[,] temp1 = MatrixMath.ScalarMultiply(gama, pattern);
        int[,] temp2 = MatrixMath.ScalarMultiply(lr, Weight);
        int[,] temp3 = MatrixMath.ScalarMultiply(delta, pattern);
        result = MatrixMath.Add(temp1 + temp2 + temp3);
        result = f(result);
        MessageBox.Show(MatrixMath.GetString(result));
    }
    catch (Exception ee) { MessageBox.Show(ee.Message); }
}
```
BSB for Graphical password

Image to Bipolar Conversion

- By using above procedure first convert the image into matrix representing binary values.
- Now convert the binary values into bipolar values by replacing 0 with -1 and represent in the form of a matrix.

```
1 0 0 0 1 1 1 1 0 0 1 1 1 0 1 1 0 1 0 1 1
```

The above procedure converts the matrix consisting of binary values into a matrix consisting of bipolar values representing all the pixels of the image.

```
1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 1 1 1 -1 1 1 1 -1 -1 -1
```

**Figure 6.6 Conversion of Image to Bipolar**

After converting the image into bipolar values the same procedure which is used for textual password can be applied.
6.3 Results for BSB

BSB For Textual Passwords

Here *No of Patterns*, specifies the number of patterns application used in the BSB training, *No of Bits Per Pattern* specifies number of bits to be used for each pattern.

![Figure 6.7 Screen showing how to setup network](image)

*Figure 6.7 Screen showing how to setup network*

Once the required information has given and *OK* button is pressed, the application will provide enough fields to enter input and output pairs.

![Input fields](image)

*Figure 6.8 Screen showing Introduction of BSB*
Training in BSB:

The required training set has given for the application as shown in the figure 6.9. When the Train button is pressed the application will perform training internally.

![Training Set](image)

*Figure 6.9 Screen showing Training of BSB*

Once the training has been completed it will display a message shown in figure 6.10.

![Completion Message](image)

*Figure 6.10 Screen showing Completion of Training*
Checking User authorization:

The application compares the output of given network with the password given by the user, if both are same then the user is valid and he will get services form the server.

Figure 6.11 Screen showing User Validation
BSB for Graphical Passwords:

This application cannot take an image directly as an input to the network. Before train the network, the required image should be converted in to text.

Figure 6.12. Welcome screen for Image Authentication

In the screen shown in figure 6.12, Path indicates path of the image, Selected Image shows the selected image, Resolution specifies resolution of the selected image and O/P Matrix Size specifies size of the output matrix.
Once the image has been selected, the "Convert To RGB" button is used to convert the image into a matrix consisting of RGB values as shown in figure 6.14.
After selecting an image if “Convert To Binary“ pressed in the application then output matrix is displayed in the output box as shown in the figure 6.15

![Figure 6.15. Screen showing how to convert an image into a binary matrix](image)

All the RGB values are converted into binary values and they will be displayed in a matrix format. This values can be used as input for password authentication using BSB method and it satisfies the condition of taking probabilistic values as input for this method.
After specifying the path the required image will be retrieved and given as input for this method. When “Convert To Bipolar” is pressed the image will be converted in to bipolar values and it will be displayed as a matrix as in figure 6.16.

![Figure 6.16 Screen showing how to convert an image into a bipolar matrix](image)

Once the image has been converted in to text, it can be used as normal textual password for giving it as input the BSB network.
6.4 Conclusion:

This chapter introduced a password authentication using BSB. In this chapter an algorithm for constructing the interconnection matrix $W$ and vector $b$ is proposed and implemented. This chapter also provides a heuristic explanation for yielding an interconnection matrix with desired properties.

The desired properties include the asymmetry of $W$. The algorithm ensures that the negatives of the desired patterns are not automatically stored as asymptotically stable equilibrium points of the network, and it has provisions to minimize the number of spurious states. Digital computer simulations verified that our design algorithm yielded a network which stored all of the desired patterns as asymptotically stable equilibrium points with very few spurious states. The network has one main shortcoming: the network is not guaranteed to be globally stable.