Warsaw University of Technology

Neural Networks

Builds and Tests the Hopfield Associative Memory with the Hebb rule

Authors:
Abba Umar, Hassan Jaber

May 8, 2023

Contents

1 Project description **2**
1.1 Requirements 2

2 Theoretical description **2**
2.1 Associative Memory 2
2.2 Hopfield Network 3
2.3 Hebb Rule 5

3 Algorithms **6**
3.1 Variables 6
3.2 Hopfield Network Training Algorithm 6
3.2.1 Storage(Learning) 6
3.3 Testing Vector's Stability 8
3.4 Convergence 9

4 Application Manual **9**
4.1 Technologies Used 9
4.2 Procedure to run 9
4.3 User manual 11

5 Results **12**

6 Conclusion **14**

List of Figures

1 Associative memory. Tutorialspoint (2022) 3
2 Hopefield. wikiwand (2022) 4
3 User manual. 11
4 Input vector. 12
5 Weight Matrix. 12
6 Stability test. 13
7 Test vectors. 13
8 Convergence test. 14

1 Project description

This is an implementation of a program that builds and tests the Hopfield Associative Memory with the Hebb rule. **Note:** All program parts except for GUI must be written without using external libraries (in particular the NN-related libraries).

1.1 Requirements

1. INPUT in the form of an external file or provided in an interactive way by the user. BOTH POSSIBILITIES (external *.txt file and by means of GUI) NEED TO BE AVAILABLE IN THE PROGRAM
2. Calculation of the Hopfield Associative Memory
3. Iterative testing of any input vector: the user provides a test vector and the system outputs the HAM vector; next the user may input another vector – receive the output, etc.
4. The program must allow calculation (testing) of the output for ANY input vector of appropriate size NOT ONLY one of the base / stored vectors.
5. The program must be able to handle both UNIPOLAR and BIPOLAR vectors;
6. NOTE THAT the final outcome may not be available just after the first iteration.

NOTE that, sometimes you must iterate several times. You may end iterations ONLY in either one of the two following cases:

1. (a) In the current iteration the output vector is equal to the input one → then you print this vector as the network's response; OR
   (b) you enter a cycle of length two → in which case you print an information about entering a cycle and output both alternating outputs.

2 Theoretical description

2.1 Associative Memory

A repository of associated patterns in some way, associative memory stores these patterns. The related pattern pair appears at the output if a pattern triggers the repository. The input could be a complete or insufficient representation of a previously stored pattern.

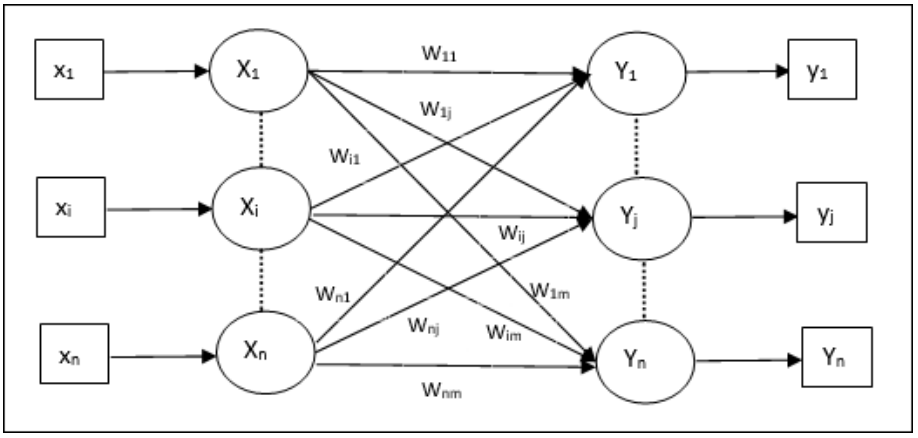


Figure 1: Associative memory. Tutorialspoint (2022)

The associated pattern is automatically recovered if the memory was created with an input pattern, such as may be input pattern x then the associated pattern y is recovered automatically.

These are the terms that are related to the Associative memory network:

Encoding or memorization: Encoding or memorization refers to building an associative memory. It implies constructing an association weight matrix w such that when an input pattern is given, the stored pattern connected with the input pattern is recovered.

$$(w_{ij}) = (p_i)_k(q_j)_k$$

Errors and noise: The input pattern may hold errors and noise or may contain an incomplete version of some previously encoded pattern. If a corrupted input pattern is presented, the network will recover the stored Pattern that is adjacent to the actual input pattern.

Performance Measures: The measures taken for associative memory performance to correct recovery are memory capacity and content addressability. Memory capacity can be defined as the maximum number of associated pattern pairs that can be stored and correctly recovered.

2.2 Hopfield Network

One specific variety of single-layered neuron networks is the Hopfield network. It was developed in 1982 by Dr. John J. Hopfield. These networks were introduced to

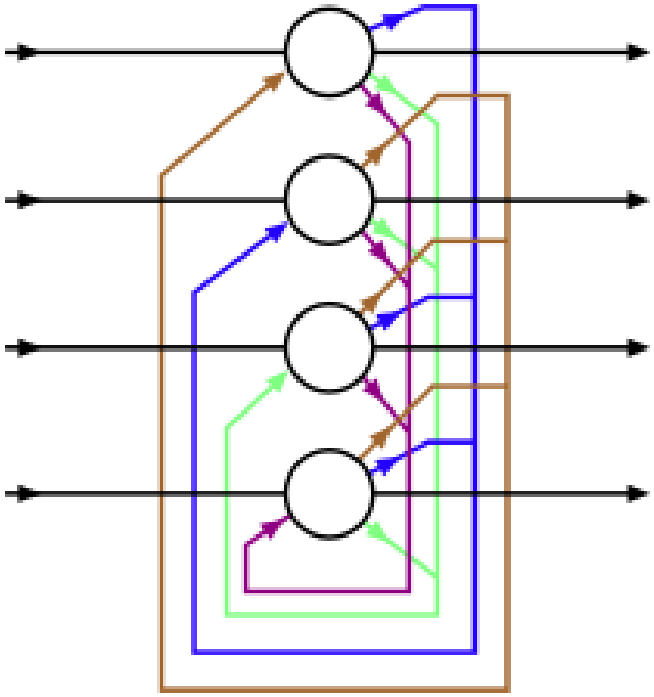


Figure 2: Hopefield. wikiwand (2022)

gather, retrieve, and store different patterns. These networks can also be used for job optimization and auto-association. The effectiveness of memory search is linked to the solution of the overfitting problem. It is proposed a strategy for breaking the training and input network vectors into pieces that require a fewer number of neurons to process. The results of a series of tests on Hopfield network models with various numbers of neurons trained with various numbers of vectors and operated under various noise circumstances are reported. The simplest associative memory is just a sum of outer products of the N patterns xiNi=1 that we want to store (Hebbian learning rule). In classical Hopfield Networks, these patterns are polar (binary), i.e. xi -1,1d $\in$, where d is the length of the patterns. The corresponding weight matrix W is:

$$W = \sum_{i}^{N} x_i x_i^T$$

The weight matrix W stores the patterns, which can be retrieved starting with a state pattern.

2.3 Hebb Rule

Hebb theory was introduced in 1949 by Donald Olding Hebb in his book "The Organization of Behavior". This theory is also called Hebb's postulate or Hebb's rule. This rule was intended to connect statistical methods to neurophysiological experiments on plasticity.

Hebbian learning is implemented by loading the m-selected n-dimensional stable states x1, x2,..., xm on the network and by updating the network's weights (initially set to zero) after each presentation according to the rule.

$$w_{ij} \leftarrow w_{ij} + k_i^k k_j^k$$

where i,j = 1,.....,n and $i \neq j$

The symbols x_i^k and x_j^k denote the i-th and j-th components respectively of the vector x^k. The only difference from an auto-associative memory is the requirement of a zero diagonal. After the presentation of the first vector x_1 the the weight matrix is given by the expression

$$W_1 = x_1^T x_1 - I$$

where I denote the n $\times$ n identity matrix. Subtraction of the identity matrix guarantees that the diagonal of W becomes zero, since for any bipolar vector x_i holds that $x_k^i x_k^i = 1$. Obviously, W_1 is a symmetric matrix

$$E(x) = -\frac{1}{2} x W_1 x^T = -\frac{1}{2} x x_1^T x_1 x^T - x x^T$$

and xx^T = n for bipolar vectors. This means that the function

$$E(x) = -\frac{1}{2} \parallel x x_1^T \parallel^2 + \frac{n}{2}$$

has a local minimum at x = x_1. In this case, it holds that

$$E(x) = -\frac{n^2}{2} + \frac{n}{2}$$
$$sgn(e) = sgn(x_1)$$

The same argumentation can be used for any of the other vectors. The best results are achieved with Hebbian learning when the vectors x1, x2,..., xm are orthogonal or close to orthogonal, just as in the case of any other associative memory.

3 Algorithms

3.1 Variables

In this section, we describe all variables used in this implementation.

Note: We use convert every vector 0 to vector -1

1. (a) V_i: represents a single neuron. It is binary, with 1 representing firing and -1 not firing.

   (b) V=[V_1,V_2,..,${V_n}^T$]: represents the entire neural network, in vector form.

   (c) X: Number of input vectors V_i.

   (d) W_{ij}: represents the weight of a connection between neurons V_i and V_j. It is determined during Hebbian Learning.

   (e) W: the weights put together in matrix form. Note that $w_{ii} = 0$ and this matrix is symmetric.

   (f) N_i=[V_i,..,${V_n}^T$]: a new training input pattern where vi is the state of neuron i of the new pattern and v_i is either 1 or -1.

   (g) U_i: The fixed threshold or bias determines neuron state. Unless otherwise stated, $U_i = 0$.

3.2 Hopfield Network Training Algorithm

During the training of the Hopfield network, weights will be updated. As we know that we can have binary input vectors as well as bipolar input vectors. Hence, in both cases, weight updates can be done with the following relation.

3.2.1 Storage(Learning)

In the learning step of the Hopfield network, we need to find the weight matrix for X of patterns fundamental memories:[V_1,V_2,..,V_x] stored in the synaptic weights of

the network according to the equation:

$$W_{ij} = \begin{cases} \sum_{x=1}^{X} v_{x,i} v_{x,j} & i \neq j \\ 0 & i = j \end{cases} \tag{1}$$

$v_{x,i}$ and $v_{x,j}$, i_th and j_th elements of the fundamental memories V_x in matrix form:

$$\mathrm{W} = \begin{bmatrix} 0 & w_{12} & ... & w_{1i} & ... & w_{1n} \\ w_{21} & 0 & ... & w_{2i} & ... & w_{2n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ w_{i1} & (w_{i2} & ... & 0 & ... & w_{in} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ w_{n1} & w_{n2} & ... & w_{ni} & ... & 0 \end{bmatrix}$$

Where $w_{ij} = w_{ji}$, The weight matrix W remains fixed.

To implement the equation 1, we adopted the following algorithm:

Algorithm 1 Iterative Weight update Algorithm

Require: $input - vectors \neq 0$

Initilize $x = 1$ to X
for each *vector* in $input - vectors$ **do**
  for $i = 0$ to $Xnumberofcolumninvector$ **do**
    for $j = 0$ to $Xnumberofcolumninvector$ **do**
      if $w_{in} = w_{ni}$ **then**
        $W[i, j] = 0$
      else
        $W[i, j] = v[x][i] + v[x][j]$
      end if
    end for
  end for
end for

In the above figure, to create the initial weight matrix w of size i * j for each vector V we have used i * j number of iterations for each vector iteration. When the last vector is reached, the matrix W with all, final, and correct W_{ij} is finalized.

3.3 Testing Vector's Stability

Given an input memory or pattern, called V_{in} , Then, using the equation below to iterate through all vectors input for updating.

$$W_{ij} = \begin{cases} 1 & \text{If} \sum_{x \neq 1} W_{xi} v_j > U_i \\ -1 & otherwise \end{cases} \tag{2}$$

where $U_i = 0$ unless otherwise stated. This process continues, with the synchronous update of neurons, until V_{in} reaches a learned memory or pattern, which will be a stable point. At this point, it can be considered the output memory which the Hopfield The network has been identified and recovered.

To implement the above equation, we utilized the following algorithm:

Algorithm 2 Algorithm for testing the stability of the vectors

```
1:  procedure STABILITY(vector, x)
2:      for j = 0 to sizeofvector do
3:          a = new List < int > ();
4:          for k = 0 to numcolumnofweightmatrix do
5:              if main diagonal then
6:                  a.Add(0)
7:              else
8:                  a.Add(vector.ElementAt(k) * weight[i,j])
9:              end if
10:             b.SetValue(a.Sum(), j);
11:         end for
12:     end for
13:     sgn(b)
14:     if matchVec(b, vector) = True then
15:         vector is stable
16:     else
17:         if itr > 2&&matchVec(ref, temp − arr, b) then
18:             Infinite iterative loop detected for this vector
19:         else
20:             vector is unstable
21:             Stability(b, x)
22:         end if
23:     end if
24: end procedure
```

3.4 Convergence

Present an unknown n-dimensional vector, N, a corrupted or incomplete version of a pattern from fundamental memories to the network and retrieve a stored association (stable state). To get the convergence vector for a specific test vector, we utilized the same algorithm 2.

4 Application Manual

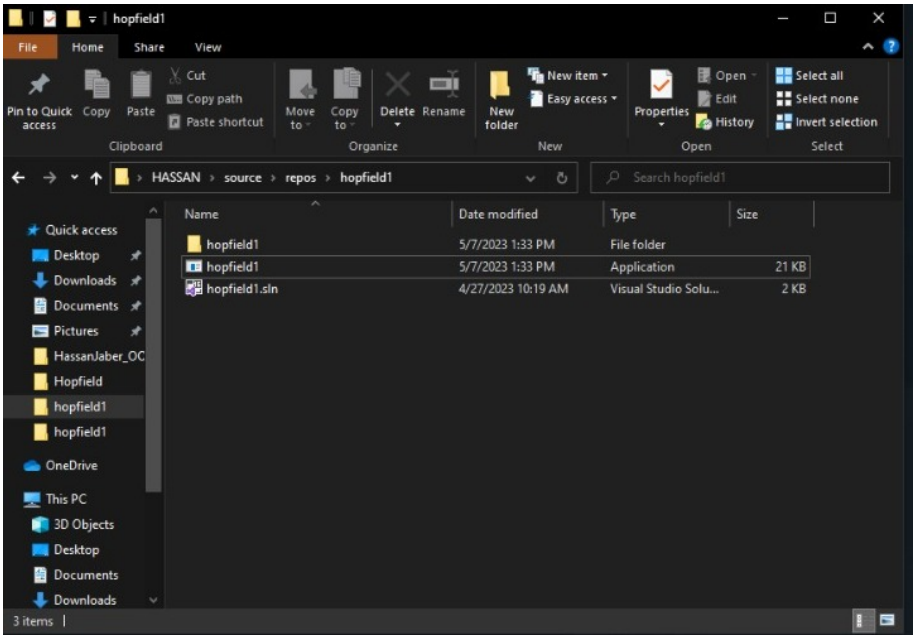
The followings are the technologies and procedures needed to run the application.

4.1 Technologies Used

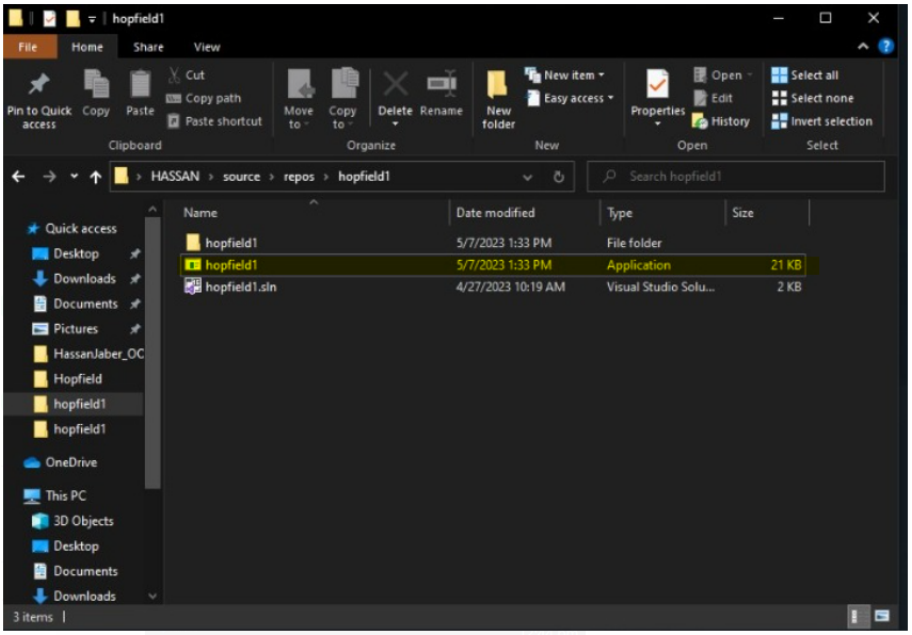
- C#
- Visual Studio 2019
- .Net Framework
- using a System library like Linq, Collections, Windows.form, etc

4.2 Procedure to run

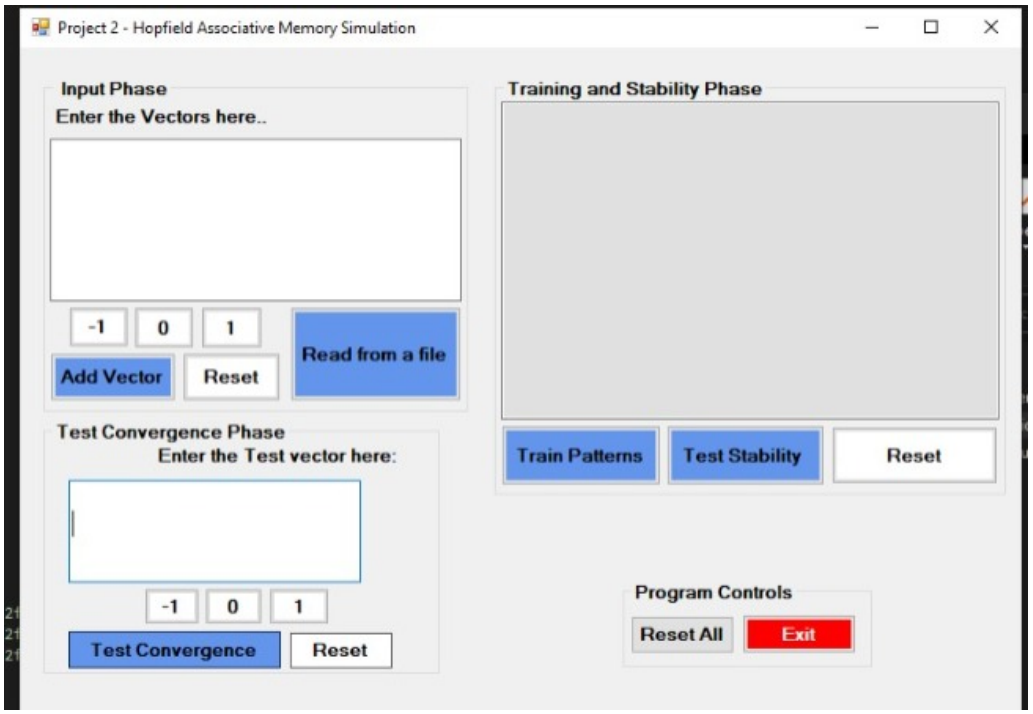
Step 1. Go to the folder "hopfield1".



Step 2. Double-click on the "hopfield1.exe" application.



Step 3. Application graphic user interface will show.



4.3 User manual

The following steps are needed for using the application.

Step 1. Input Vector.

There are two ways to insert vectors into the application

1. User enters each vector manually.

2. Upload the vectors from a file.

Note : After using the manual method above you need to press the "Add Vector" button after each vector you type.

Step 2. After adding the vectors press the "Train Patterns" to calculate the weight matrix.

Step 3. To test the stability of the vectors press the "Test Stability" button.

Step 4. To test convergence enter the test vector and press the "Test convergence" button.

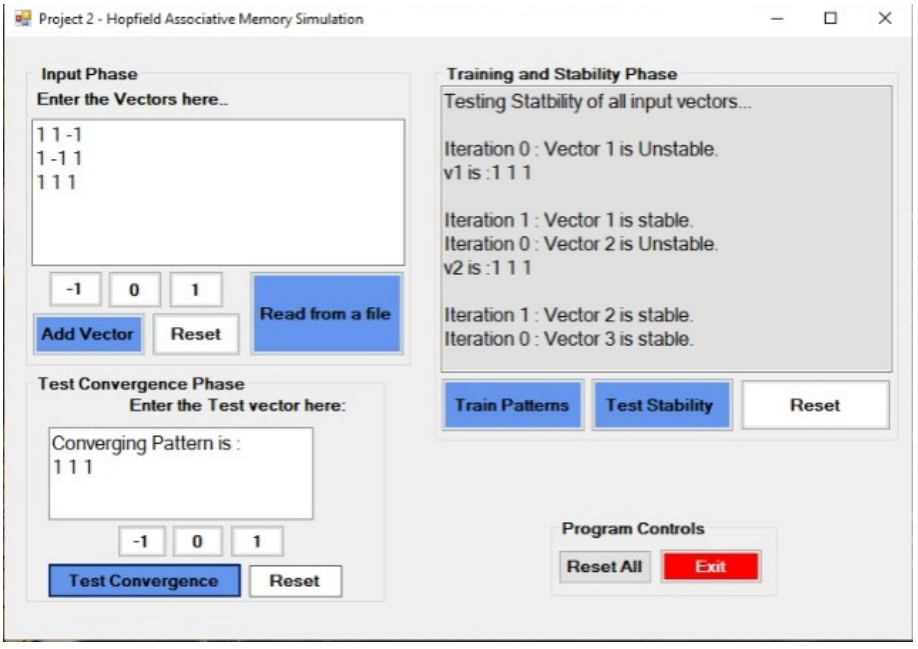


Figure 3: User manual.

5 Results

In this section, we show the result of the weight matrix, stability test, and convergence test for the vectors used after the implementation of the above-mentioned technique.

The input vectors to be used are shown below

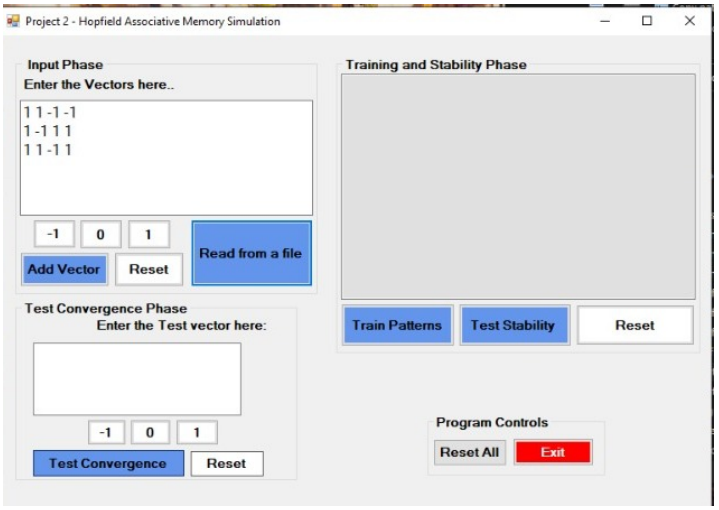


Figure 4: Input vector.

The figure below shows the calculated weight matrix of the vectors

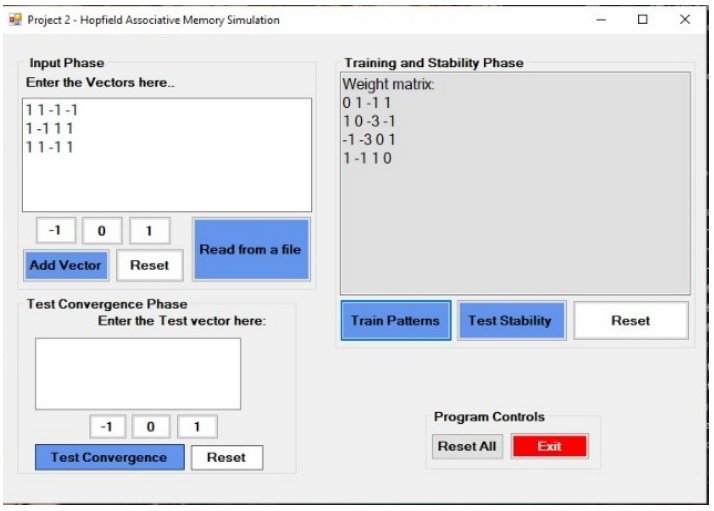


Figure 5: Weight Matrix.

The figure below shows the stability test result of the vectors after pressing the "Test stability" button. As we can see vector 1 is stable, and vector 2 is unstable but after the second iteration, it becomes stable. similarly, vector 3 is unstable after the second iteration it becomes stable

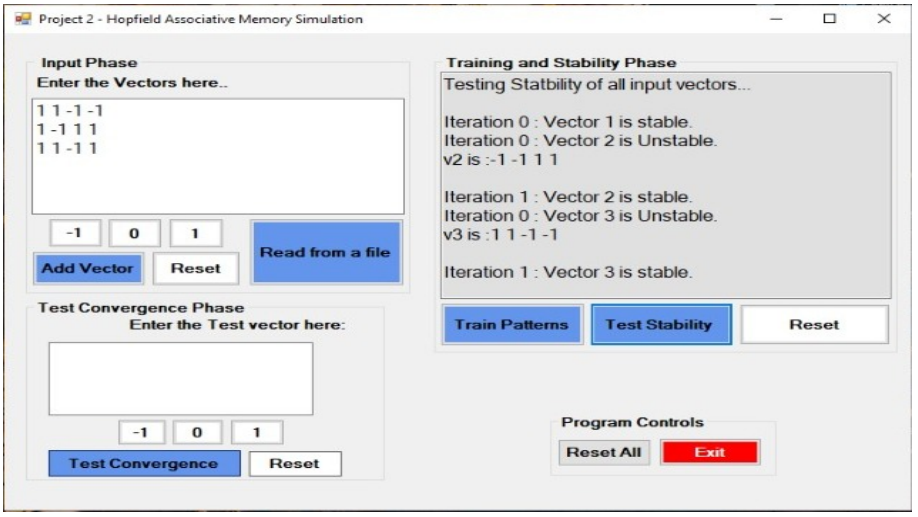


Figure 6: Stability test.

The figure below shows the vectors test we wish to use with the weight matrix to test for convergence. The test vector can be corrupted or incomplete

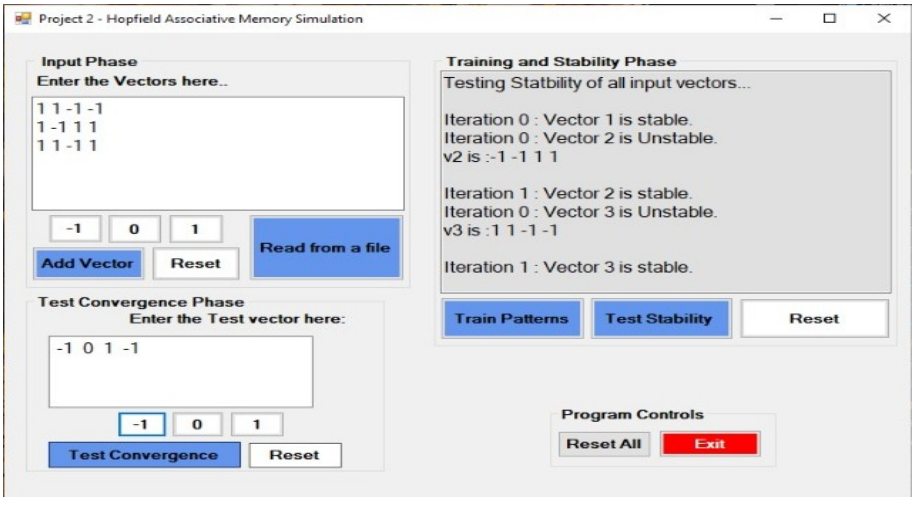


Figure 7: Test vectors.

The figure below shows the convergence test result of the test vectors

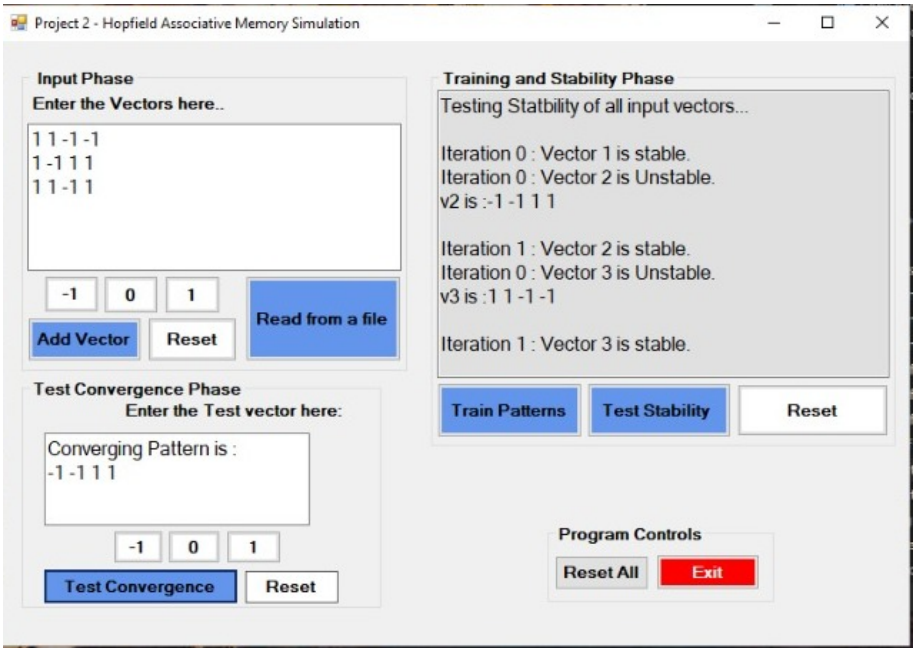


Figure 8: Convergence test.

6 Conclusion

The Hopfield-hebb rule approach to associative learning in which simultaneous activation of neuron cells leads to pronounced increases in synaptic strength between those cells has solved the optimization problem.

References

Tutorialspoint (2022), "Neural network: Hopfield networks." URL `https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_hopfield.htm`.

wikiwand (2022), "Hopfield networks." URL `https://www.wikiwand.com/en/Hopfield_network`.