Hebb Rule Method in Neural Network for Pattern Association

Hello Ali Hama

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Applied Mathematics and Computer Science

Eastern Mediterranean University
May 2014
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Applied Mathematics and Computer Science.

Prof. Dr. Nazim Mahmudov
Chair, Department of Mathematics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Applied Mathematics and Computer Science.

Prof. Dr. Rashad Aliyev
Supervisor

Examining Committee

1. Prof. Dr. Rashad Aliyev
2. Asst. Prof. Dr. Ersin Kuset Bodur
3. Asst. Prof. Dr. Müge Saadetoğlu

ABSTRACT

In the process of the development of intelligent systems the artificial neural network plays an important role as a paradigm for pattern recognition, pattern association, optimization, prediction, and decision making problems.

This master thesis focuses on analysis of Hebb rule for performing a pattern association task. The application of Hebb rule enables computing optimal weight matrix in heteroassociative feedforward neural network consisting of two layers: input layer and target output layer.

The Hebb algorithm is applied to both binary and bipolar data representations. The advantages of bipolar representation of training patterns compared to binary representation of training patterns are presented. Two different ways for calculating weight matrix are used: the results of application of the Hebb algorithm, and the outer products. New input vectors which can be similar and not similar to training input vectors are tested. A new input vector differing from the training input vector in fewer components should produce the reasonable response as the same output vector.

Keywords: Neural network, Hebb rule, pattern association, binary and bipolar vectors, outer products

ÖZ

Yapay sinir ağı akıllı sistemlerin oluşumu sürecinde önemli rol alır ve örüntü tanıma, örüntü ilişkilendirme, optimizasyon, öngörü, ve karar verme problemlerinde paradigma olarak kullanılır.

Bu tez Hebb kuralını kullanarak örüntü ilişkilendirme görevinin incelenmesine odaklanır. Hebb kuralını uygulamakla zit ilişkili ileri beslemeli sinir ağında optimal ağırlık matrisi hesaplanır. Bu ağ iki katmandan oluşmaktadır: giriş ve hedef çıkış katmanları.

Hebb algoritması ikili ve iki kutuplu veri representasyonu için uygulanır. Eğitim örüntülerin iki kutuplu representasyonunun ikili representasyona nazaran daha avantajlı olduğu gösterilir. Ağırlık matrisinin hesaplanması iki farklı yöntemle hayata geçirilir: Hebb algoritmasının uygulanmasından elde edilen sonuçlar, ve dış çarpımlar yöntemi. Eğitim giriş vektörlerine benzer olan ve benzer olmayan yeni giriş vektörleri test edilir. Eğitim giriş vektöründen daha az bileşenle farklanan yeni giriş vektörü uygun cevap olarak aynı çıkış vektörünü üretmelidir.

Anahtar Kelimeler: Sinir ağı, Hebb kuralı, örüntü ilişkilendirme, ikili ve iki kutuplu vektörler, dış çarpımlar

ACKNOWLEDGMENTS

I express my sincere gratitude and special appreciation to my supervisor Professor Dr. Rashad Aliyev for his support in my master study and research. His guidance helped me to finish my thesis.

I would like to thank all my instructors in the department of Mathematics at the Eastern Mediterranean University for their help and support for giving me a wide range of knowledge.

I would be honored to express my gratitude to all my family, my parents, my brothers Pers and Shallo, and his wife Nian, my sister Hallaz, my dear Ekhlas, and pari, naaima, and, my uncles Ismael, Rahman, Rafiq, and Sdiq. They always supported me.

A lot of thanks go to all my friends for their continuous support and encouragement throughout this period.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Neural network is one of the most powerful techniques of artificial intelligence which is also called artificial neural network (ANN). Neural networks are the systems that are able to acquire and to use human knowledge available from the experience.

The history of neural network begins from 1943 in USA by the physiologists McCulloch and Pitts. They showed the first sample of a neuron. In the very first paper in neural network published by above scientists the modeling of a neural network was performed on the base of electrical circuits.

In 1955 Allen Newell and Herbert A. Simon tried to simulate human thinking (making mind). At the same time B. Widrow worked on modeling the brain to design the electronic system simulating human brain.

In 1969 Marvin Minsky and Seymour Papert published the book “Perceptrons”, in which it was proven that the neural systems can use XOR logical function.

Another scientist Rosenblatt designed the first neural network called perceptron. In 1977 associative memory model was developed by Finnish scientist Kohonen. In 1982, Hopfield applied a simulation of the physical energy to a neural network.

In 1986 the multi layer perceptron was proposed. Since that time the neural systems achieved impressive results in many different fields such as shape identification, signal analysis, pattern recognition, image compression and many other civil and military applications.

The researchers in artificial neural networks have been creating algorithms and theory for the basics of the brain action.

Why is a neural network an important technique? What kind of advantages does a neural network provide? The following characteristics of a neural network show the importance of its implementation for different problems:

- Adaptability to learn performing some tasks on the base of data available from the experience;

- Ability of a self-organization while learning process;

- Capability of the realization of computations in a parallel form like human brain does it, i.e. the neural network is not programmable;

- Powerful capability for pattern recognition and pattern classification;

- Ability for approximation of a universal function;

- Handling the information with probabilistic and fuzzy natures;

- Ability to produce the output of the neural network for the classification problems if there is no information about how should this output look like;

- Possibility of the implementation of the neural network for the problems where the relationship between dependent and independent variables is non-linear. If non-linear data should be modeled, the classical linear network can't perform this task, and that is why an implementation of a non-linear neural network is more advantageous compared to classical one. Non-linear modeling of data for defining relationship between dependent and independent variables can provide a motivation for a neural network to be an excellent forecasting technique. A neural network is an appropriate technique for optimization problems.

The neural networks can be classified into the following types:

- Feedforward neural network.

The processing units in feedforward network will connect where the information is flowing in one direction, i.e. the information always goes forward but never backward. This type of neural network is well suited for forecasting problems. The feedforward neural network can be used for modeling the dynamic systems, but this

network doesn't have a dynamic memory, for example, the backpropagation neural network;

- Recurrent neural network.

In this type of a neural network the connection between the units are performed in a form of direction cycle (forward and backward directions).

There are three types of training techniques in neural networks: supervised, unsupervised, and hybrid learning.

Supervised learning encompasses technicality of supplying network with the inputs and outputs. In supervised learning the inputs and outputs are supplied as a training set, and a comparison between the output result and the direction output is performed by defining the mean-squared error between them, and a system propagates back the errors, after that the system regulates the weights. This process happens many times by learning function until the best result will be available. One of the popular supervised neural networks is backpropagation algorithm, and the different modifications of this algorithm are used to decrease the time needed for the training process in the neural network.

In unsupervised learning the network has inputs but running out of any output, i.e. in unsupervised type of learning, the network is supplied with inputs but not with desirable outputs, and the desired output is unknown. In other words, the teacher signal is absent in unsupervised neural network. Unsupervised neural networks mostly consist of input and output layers, and the hidden layer is absent. This kind

of network has an ability to find the distinctive features for the inputs through the other outputs, and it works without any previous knowledge which means that a neural network has an ability to organize itself after the weights are correlated.

The hybrid learning uses the combination of both supervised and unsupervised learning techniques.

Tuevo Kohonen is one of the best researchers in unsupervised neural networks, who developed the self-organizing map also called Kohonen SOM. The competitive learning process is realized in this kind of networks. Some characteristics of input data should be known. This network can also recognize and classify the inputs which were not used whenever before. This classification is done for the values of input data apart from input space of data.

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.

Hebb's rule's implementation is easy and takes a few number of steps. Implementation of Hebb's rule considers at first the input values and expected output values, then the activation function is used, and finally the Hebb's algorithm is implemented.

The Hebbian algorithm is used in many areas, and especially in speech and image processing.

The question whether the neural networks can fully simulate the human brain in optimal form is still actual. Neural networks are very useful for applications in combination with other components such as genetic algorithms, fuzzy logic, expert systems etc. All the existing technologies of neural networks can be improved in integration with above components for pattern recognition, stock market analysis and prediction problems, medical diagnosis of patients, transformation of handwritten documents to other appropriate formats, and in other areas where people deal with a large quantity of data.

Chapter 2

REVIEW OF EXISTING LITERATURE ON APPLICATION OF THE HEBB RULE

In [1] the unsupervised Hebbian learning algorithm is discussed, and both the content addressability and saving capacity of the learned networks are considered. The observed dynamical results show that the learning process increases the dimension of the possibility attractors, but less chaoticity compare to supervised neural network is obtained.

The paper [2] describes the classical neuroscience model of Hebbian learning. It is hard to achieve the efficient associative memory storage using Hebbian synaptic learning. In the result it is proposed that associative learning by Hebbian synaptic learning should be accompanied by continuous remodeling of regulator processes in the brain.

In [3] the Generalized Hebbian Algorithm equivalent to Latent Semantic Analysis (LSA) is studied, and the possibility of application of Generalized Hebbian Algorithm for the tasks of LSA as well as for very large datasets is defined.

In [4] a learning rule for oscillators that prepare their frequency to the frequency of any periodic or pseudo-periodic input signal is shown. The important feature of this model is being comfortably generalizable for a big category of oscillators. The main

advantage of the learning rule is that the oscillators enable to prepare their frequency and don't need any signal processing.

In [5] the Generalized Hebbian Algorithm is proposed that allows the singular value decomposition of datasets to learn. This algorithm is very useful for large datasets in which data are normally intractable. The experimental results are discussed on the task of modeling some letter bigram pairs.

Using weighted Hebb rule in the presence of noise confirms that the structure of the minima of the free energy at finite temperatures is differed from the using of a usual Hebb rule [6]. If there is a single extra pattern stored with appropriate weight, then the temperature of the free energy is lower than the network without extra pattern. The convergence time can be significantly improved.

The error-driven rules are preferred to pure Hebb rule when the correct output of the neural network is specified [7]. In the unsupervised learning rules the Hebb rule is more appropriate in combination with normalization in order to stop increasing the number of synapses.

The idea of [8] is to develop the algorithms for the metadata extraction in a digital library. This matrix of association is used to recommend the documents which are more convenient to user's interest profile. The matrix is also used for calculation of document similarity values, and clustering the similar documents. The data are extracted to create a documentation system.

The paper [9] introduces the unsupervised learning algorithm based on weights update and connections of the Hebbian rule. The presented algorithm is applied for reliable classification of the handwritten digits.

In [10] the adaptive reasoning theory and Hebb rule are used to develop the neural network and pattern recognization with the aim of precise classification of network on unseen test data. The experiments show that the learning process can be improved by considering the accuracy of the outcome feedback.

Hebb rule based learning algorithm for studying the unspecific reinforcement is proposed in [11]. The asymptotic convergence for learning parameters may be sometimes as fast as Hebbian learning, but may be also slower. The initial conditions can define the regime of asymptotically perfect generalization or the state of poor generalization.

The application of the contrastive Hebbian algorithm to continuous Hopfield network is represented in [12]. The five different training regimes are analyzed. The implementation instructions of contrastive Hebbian learning in competition models are presented.

In [13] the back-propagation learning rule as supervised neural network, and the Hebbian learning rule as unsupervised learning are presented, and the filtering aspects of Hebbian network for the recognition of the anomalous patterns in data are discussed. The ability of storing information by Hebbian network is discussed. The

parallels between the emergence phenomena of Hebbian neural network and human intelligence are drawn.

In [14] the learning capability of Hebbian unsupervised rule is discussed that makes a probabilistic associative memory (PAM) a good functional model for hierarchical pattern recognition problem. The strength of the synapse is related with the outputs of the presynaptic and postsynaptic neurons; if the outputs of the neurons are identical, then the strength increases, and decreases otherwise.

The recursive auto-associative memory (RAAM) learning is used for training of auto-associative networks for representing structured information. The use of Hebbian learning rule to represent the structured information is given in [15]. This information is represented in terms of vector graphic.

In [16] the models having energy function from the Hebb rule is presented. The networks with static synaptic noise and synapses which are nonlinear functions are discussed.

In [17] the extension of some learning rules in a Principal Component Analysis network is performed for deriving optimality for the family of probability density functions. The probability density functions are used to perform Exploratory Projection Pursuit.

The application of Soft Computing techniques is important for modeling systems. In [18] the Fuzzy Cognitive map (FCM) consisting of neural network and fuzzy logic components is discussed. For the improvement of efficiency and robustness, and training FCM, the unsupervised learning method based on nonlinear Hebbian rule is practically applied. The process control is performed by defining the desired convergence regions for FCM.

Chapter 3

BINARY DATA REPRESENTATION OF A HETEROASSOCIATIVE NET

3.1 Pattern association

Learning is the operation of formalization in associations between the patterns. In associative neural network the weights are determined, and the network can store a set of patterns [19].

We will assume some simple single-layer neural networks with ability to learn a set of associations. The associative memory network can act in the form of a very simplified human brain.

The purpose of using an associative neural network is to find the appropriate output vector corresponding to an input vector which can be one of the stored patterns or a new pattern [19].

Every association is a pair of vectors of an input-output A, B, where A is the input vector, and B is the target vector. If both the vectors A and B are the same, then we call it an autoassociative memory neural network. If the vectors A and B are different, then it is called a heteroassociative memory neural network. In each of

these cases, the learning is not only for particular pattern pairs that were used for training, but likewise has ability to recall response pattern.

3.2 Training algorithm for pattern association. Hebb algorithm

Hebb rule is common and simple method for specifying the weights for an associative memory neural network. The Hebb rule can be used with either binary or bipolar patterns. The algorithm will iterate for input and target (output) training vectors, and in order to find the weights, the outer product is used as the general procedure. We consider the training pair of vectors A, B, and afterwards the testing input vector x is considered.

The Hebb algorithm can be represented in the following form:

- Set all the initial weights equal to 0:

$$w_{ij} = 0; (i = 1, \dots, n\ ; j = 1, \dots, m);$$

- For each input-output training case set the following activations: for input units to the current input

$$A_i = x_i\ , (i = 1, \dots, n),$$

and for output units to the current target output:

$$B_j = y_j, (j = 1, \dots, m);$$

- Adjust the weights using the following formula:

$$w_{ij}(new) = \ \ w_{ij}(old) + x_i\, y_j, (i = 1, \dots, n; j = 1, \dots, m);$$

- Set the activation of the output units:

$$y_j = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad \text{(for binary targets)}$$

$$y_j = \begin{cases} 1, & if\ y\_in_j > 0 \\ 0, & if\ y\_in_j = 0 \\ -1, & if\ y\_in_j < 0 \end{cases} \quad \text{(for bipolar targets)}$$

where

$$y\_in_j = \sum_i x_i w_{ij}$$

is a computation of the net input to the output unit.

3.3 Outer products

Sometimes the weights of a net are computed from the training set by using the outer product instead of iterative updating of weights used in algorithm for Hebb rule. We write the training vector of the matrix product as a column represented as $n \times 1$ matrix, and the target vector represented as a row vector with $1 \times m$ matrix. If $A = (a_1, \dots, a_i, \dots, a_n)$, and $B = (b_1, \dots, b_j, \dots b_m)$, we have:

$$AB = \begin{bmatrix} a_1 \\ \vdots \\ a_i \\ \vdots \\ a_n \end{bmatrix} [b_1 \dots \quad b_j \dots \quad b_m] = \begin{bmatrix} a_1 b_1 \dots & a_1 b_j \dots & a_1 b_m \\ \vdots & \vdots & \vdots \\ a_i b_1 \dots & a_i b_j \dots & a_i b_m \\ \vdots & \vdots & \vdots \\ a_n b_1 \dots & a_n b_j \dots & a_n b_m \end{bmatrix}$$

Using the Hebb rule we can find the weight matrix to store the association A, B.

We store association set $A(p), p = 1, \dots, P$, where

$$A(p) = (a_1(p), \dots, a_i(p), \dots, a_n(p))$$

and

$$B(p) = \left(b_1(p), \dots, b_j(p), \dots, b_m(p)\right)$$

The weight matrix $W = \{w_{ij}\}$ is given by

$$w_{ij} = \sum_{p=1}^{P} a_i(p)\, b_j(p)$$

which means the summation of the outer product matrices.

The general preceding formula (vector-matrix form) is

$$W = \sum_{p=1}^{P} A^T(p)B(p)$$

The appropriateness of the Hebb rule in different problems is up to the correlation between the input training vectors. In case the input vectors are orthogonal (if their dot product is equal to 0), the correct weights will be produced using Hebb rule. Afterwards in the process of testing one of the given training vectors, the response of the network will be perfect recall of the target associated with given input vector. Otherwise, the response will contain a portion of target values, if the input vectors are not orthogonal, and this is called cross talk.

We show the case when cross talk occurs, and two vectors $A(k)$ and $A(p)$ $(k \neq p)$ are considered to be orthogonal, if their dot product is 0.

In other words, we can write it as:

$$A(k)A^T(p) = 0,$$

or

$$\sum_{i=1}^{n} a_i(k)a_i(p) = 0$$

The weight matrix is W, and the response of the net is $y = xW$. If $x = a(k)$, the input is the *k*th training input vector, then the net response is

$$a(k)W = \sum_{p=1}^{p} a(k)a^T(p)b(p)$$

$$= a(k)a^T(k)b(k) + \sum_{p \neq k} a(k)a^T(p)b(p)$$

If $A(k)$ is not orthogonal to the other A - vectors, it is necessary to know which $A(k)$ is not orthogonal. In case the input vectors are not orthogonal, but they are related to the same target values, the cross talk between these input vectors does not cause any problem to accomplish the training process of the network.

3.4 Heteroassociative and autoassociative memory neural networks

Neural networks in which weights can store a set of associations of P pattern are called associative memory neural networks. A pair of vectors $A(p)$, $B(p)$, with $p = 1,2, \dots, P$ is an association. Every vector $A(p)$ is n-tuples (n components), and every vector $B(p)$ is m-tuples (m components). By using Hebb rule the weights can be calculated. The Hebb rule is used in examples in this section. An appropriate output vector that identifies an input vector x can be either one of the stored patterns $A(p)$ or a new pattern which is found by the network.

The architecture of the heteroassociative memory neural network is shown in Figure 1.

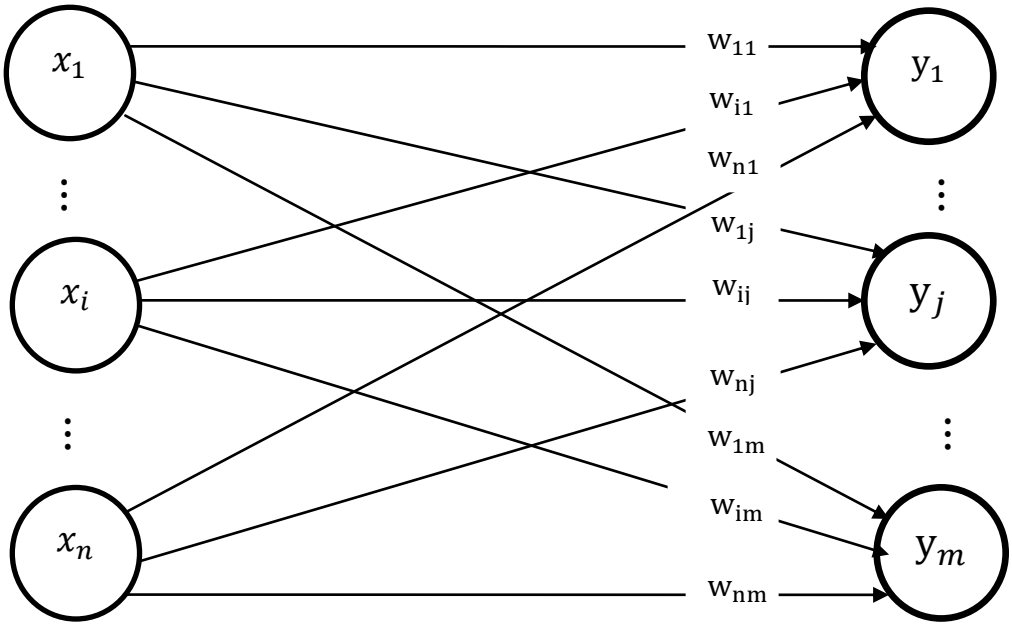

Figure 1: Architecture of the heteroassociative neural network

The feedforward autoassociative network is a private case of heteroassociative neural network. In autoassociative network the input (training) and output (target) are identical. The training process is referred to the vectors storing either binary or bipolar targets. If the input is enough similar to the stored vector, it may be restored from noisy (deformed or partial) input. The ability to copy a stored pattern from partial (noisy) input is the judgment of the performance of the net. In general the bipolar vector is better than the binary vector in different applications.

The architecture of the autoassociative memory neural network is shown in Figure 2.

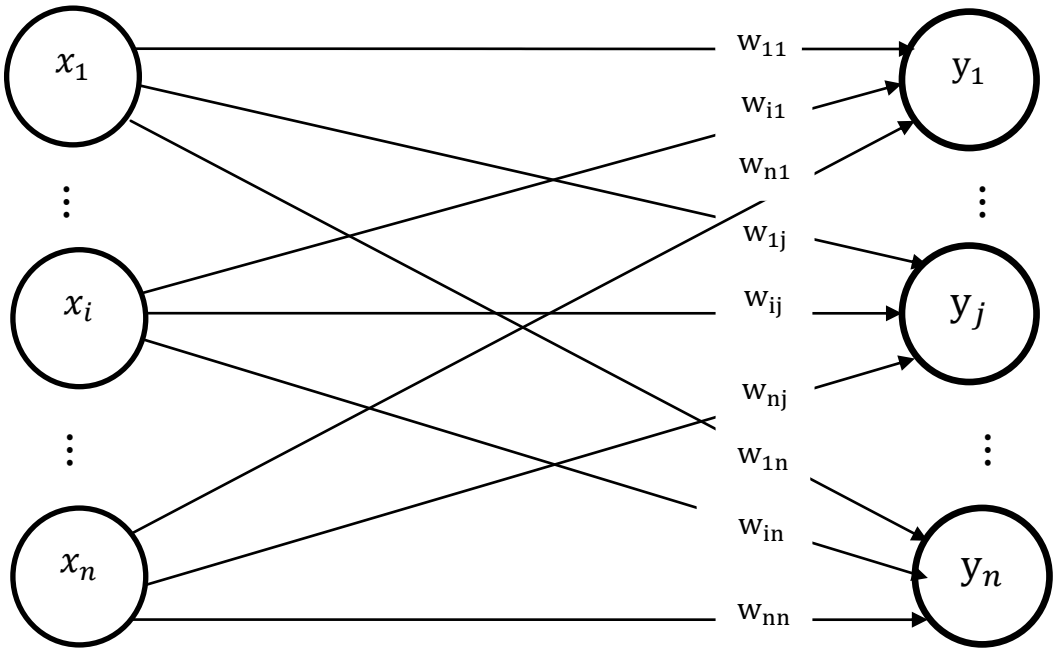

Figure 2: Architecture of the autoassociative memory neural network

A general form is used in the bidirectional associative memory (BAM), when a threshold is θ_j, then

$$y_j = \begin{cases} 1, & if\ y\_\ in_j > \theta_j \\ y_j, & if\ y\_\ in_j = \theta_j \\ -1, & if\ y\_in_j < \theta_j \end{cases}$$

Let's assume that the input vectors are $A = (a_1, a_2, a_3, a_4, a_5)$, and the output row vectors are $B = (b_1, b_2)$. In the following example we use the Hebb rule for training a heteroassociative neural network.

Suppose a neural network is trained to store the mapping from input row vectors with five components to output (target) row vectors with two components, and these vectors are used for pattern classification problem.

First input	$A = (1, 0, 1, 0, 0)$,	first output	$B = (1, 0)$
Second input	$A = (0, 1, 1, 0, 0)$,	second output	$B = (1, 0)$
Third input	$A = (0, 0, 0, 1, 1)$,	third output	$B = (0, 1)$
Fourth input	$A = (0, 0, 0, 1, 0)$,	fourth output	$B = (0, 1)$

We observe that the first and second input vectors, and also the third and fourth input vectors are not mutually orthogonal, and the cross talk between first and second input vectors, and also between third and fourth input vectors does not create any problem, since their target values are the same.

The figure 3 shows the architecture of the heteroassociative neural network consisting of input vector with five components, and output vector with two components. The Hebb rule is used to perform the training process using

$$w_{ij}(new) = w_{ij}(old) + a_i b_j; \; i.e., \Delta w_{ij} = a_i b_j$$

Training algorithm:

- Set all the initial weights equal to 0.
- For the first input-target pair $(1, 0, 1, 0, 0) : (1, 0)$:
- $x_1 = 1; \; x_2 = 0, \; x_3 = 1, \; x_4 = 0, x_5 = 0.$
- $y_1 = 1; \; y_2 = 0.$
- $w_{11}(new) = w_{11}(old) + x_1 y_1 = 0 + 1 * 1 = 1.$

$$w_{31}(new) = w_{31}(old) + x_3 y_1 = 0 + 1 * 1 = 1.$$

(All other weights remain unchanged).

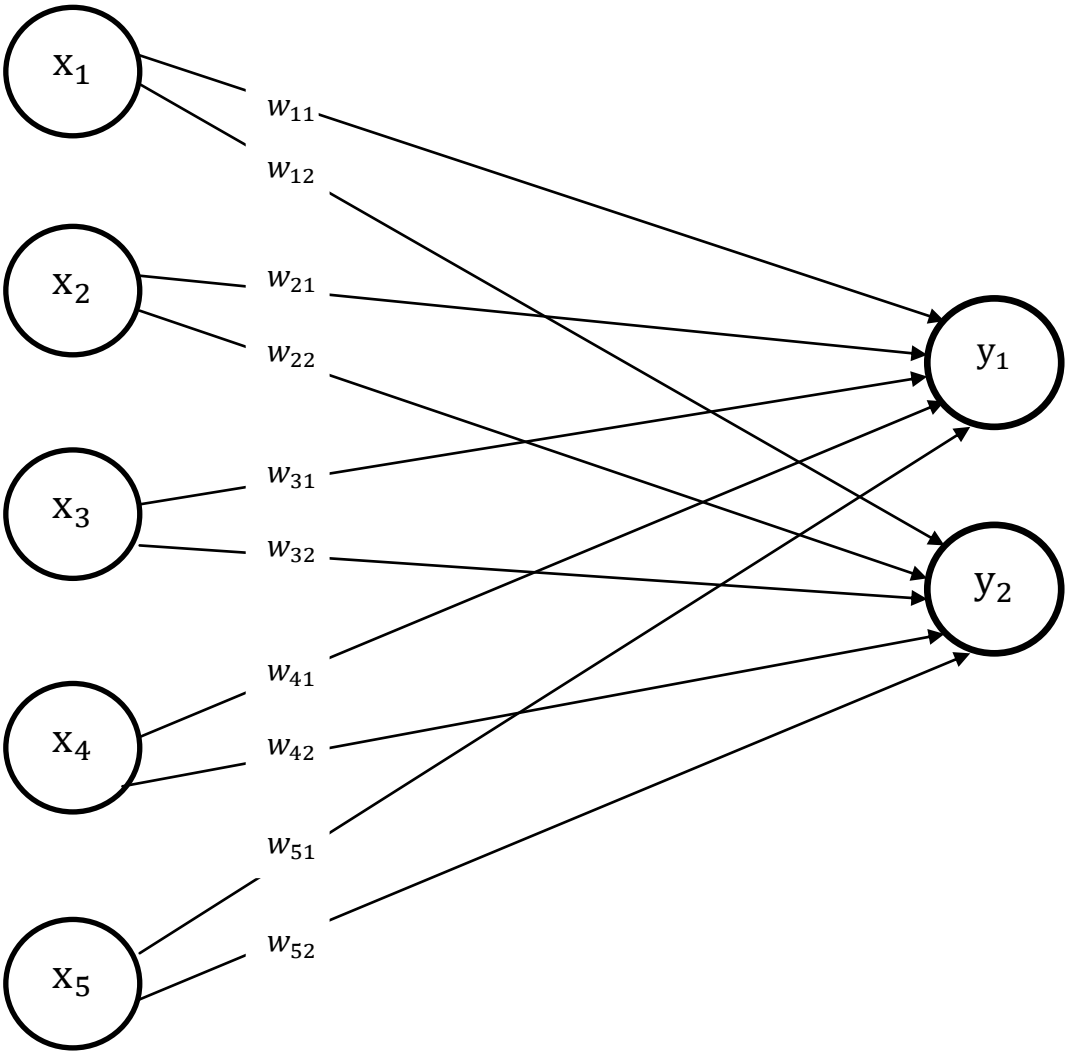

Figure 3: Architecture of the heteroassociative neural network consisting of input vector with five components and output vector with two components

- For the second input-target pair $(0,\ 1,\ 1,\ 0, 0) : (1, 0)$:
- $x_1 = 0,\ x_2 = 1,\ x_3 = 1,\ x_4 = 0,\ x_5 = 0.$
- $y_1 = 1,\ y_2 = 0.$
- $w_{21}(new) = \ w_{21}(old) + \ x_2 y_1 \ = 0 + 1 * 1 = 1.$

$$w_{31}(new) = \ w_{31}(old) + \ x_3 y_1 = 1 + 1 * 1 = 2.$$

(All other weights remain unchanged).

- For the third input-target pair $(0,0,0,1,1) : (0,1)$:
- $x_1 = 0,\ x_2 = 0,\ x_3\ 0,\ x_4 = 1,\ x_5 = 1.$
- $y_1 = 0,\ y_2 = 1.$
- $w_{42}(new) = \ w_{42}(old) + x_4 y_2 = 0 + 1 * 1 = 1.$

$$w_{52}(new) = \ w_{52}(old) + \ x_5 y_2 = 0 + 1 * 1 = 1.$$

(All other weights remain unchanged).

- For the last fourth input-target pair $(0,0,0,1,0) : (0,1)$:
- $x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 1,\ x_5 = 0.$
- $\mathrm{y}_1 = 0,\ \mathrm{y}_2 = 1.$
- $w_{42}(\mathrm{new}) = w_{42}(old) + x_4 y_2 = 1 + 1 * 1 = 2.$

(All other weights remain unchanged).

The final weight matrix is

$$\mathrm{W} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$$

Let's now use outer products instead of the algorithm of the Hebb rule. Using outer product leads to the same weights which were found by the application of the Hebb rule. For the first input-target pair

$$(1,0,1,0,0) : (1,0)$$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} [1 \quad 0] = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

We do the same process to store the second pair:

$$(0,1,1,0,0) : (1,0)$$

$$\begin{bmatrix}0\\1\\1\\0\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} = \begin{bmatrix}0 & 0\\1 & 0\\1 & 0\\0 & 0\\0 & 0\end{bmatrix}$$

For the third pair we have:

$$(0,0,0,1,1) : (0,1)$$

$$\begin{bmatrix}0\\0\\0\\1\\1\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\\0 & 1\end{bmatrix}$$

For the last fourth pair we define:

$$(0,0,0,1,0) : (0,1)$$

$$\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\\0 & 0\end{bmatrix}$$

The sum of all the weight matrices to store all pattern pairs is:

$$W = \begin{bmatrix}1 & 0\\0 & 0\\1 & 0\\0 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\1 & 0\\1 & 0\\0 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\\0 & 1\end{bmatrix} + \begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 1\\0 & 0\end{bmatrix} = \begin{bmatrix}1 & 0\\1 & 0\\2 & 0\\0 & 2\\0 & 1\end{bmatrix}$$

Now we use the training input to test the heteroassociative network.

To produce the correct output for all the training inputs we test the ability of the net by using the activation function

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

The weight matrix found is

$$W = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$$

For the first input pattern $(1, 0, 1, 0, 0)$:

$$y\_in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} + x_5w_{51} =$$

$$= 1(1) + 0(1) + 1(2) + 0(0) + 0(0) = 3\ ;$$

$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} + x_5w_{52} =$$

$$= 1(0) + 0(0) + 1(0) + 0(2) + 0(1) = 0.$$

$$y_1 = f(y\_in_1) = f(3) = 1; \qquad y_2 = f(y\_in_1) = f(0) = 0.$$

This is the correct response.

For the second input pattern $x = (0\ ,1\ ,1\ ,0\ ,0)$:

$$y\_\ in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} + x_5w_{51}$$

$$= 0(1) + 1(1) + 1(2) + 0(0) + 0(0) = 3\ ;$$

$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} + x_5w_{52}$$

$$= 0(0) + 1(0) + 1(0) + 0(2) + 0(1) = 0.$$

$$y_1 = f(y\_in_1) = f(3) = 1; \qquad y_2 = f(y\_in_1) = f(0) = 0.$$

This is the correct response.

For the third input pattern $(0, 0, 0, 1, 1)$:

$$y\_\ in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} + x_5w_{51}$$

$$= 0(1) + 0(1) + 0(2) + 1(0) + 1(0) = 0\ ;$$

$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} + x_5w_{52}$$

$$= 0(0) + 0(0) + 0(0) + 1(2) + 1(1) = 3.$$

$$y_1 = f(y\_in_1) = f(0) = 0; \qquad y_2 = f(y\_in_1) = f(3) = 1.$$

This is the correct response.

For the last fourth input pattern $(0, 0, 0, 1, 0)$:

$$y\_\ in_1 = x_1w_{11} + x_2w_{21} + x_3w_{31} + x_4w_{41} + x_5w_{51}$$

$$= 0(1) + 0(1) + 0(2) + 1(0) + 0(0) = 0\ ;$$

$$y\_in_2 = x_1w_{12} + x_2w_{22} + x_3w_{32} + x_4w_{42} + x_5w_{52}$$

$$= 0(0) + 0(0) + 0(0) + 1(2) + 0(1) = 2.$$

$$y_1 = f(y\_in_1) = f(0) = 0; \qquad y_2 = f(y\_in_1) = f(2) = 1.$$

This is also the correct response. So the weights obtained to store input-output pairs were found correctly.

The vector-matrix notation can better represent the above process. We will use the application procedure for the input vectors.

- The weight matrix is:

$$W = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$$

- $XW = (y\_in_1, y\_in_2)$, and for the first input vector $x = (1\ ,0\ ,1\ ,0\ ,0)$ we have:

$$(1\ ,0\ ,1\ ,0\ ,0) * W = (1\ ,0\ ,1\ ,0\ ,0) \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} = (3\ ,0)$$

Since $f(3) = 1;\ \ f(0) = 0,$ we have $y = (1, 0)$.

For the other input vectors we can apply the same algorithm. For the second input vector we have:

$$(0\ ,1\ ,1\ ,0\ ,0) * W = (0\ ,1\ ,1\ ,0\ ,0) \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} = (3\ ,0)$$

Since $f(3) = 1;\ \ f(0) = 0,$ we have $y = (1, 0)$.

For the third input vector we have:

$$(0\ ,0\ ,0\ ,1\ ,1) * W = (0\ ,0\ ,0\ ,1\ ,1) \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} = (0\ ,3)$$

Since $f(0) = \ 0;\ \ f(3) = 1,$ we have $y = (0, 1)$.

For the last fourth input vector we have:

$$(0\,,0\,,0\,,1\,,0) * W = (0\,,0\,,0\,,1\,,0)\begin{bmatrix}1 & 0\\ 1 & 0\\ 2 & 0\\ 0 & 2\\ 0 & 1\end{bmatrix} = (0\,,2)$$

Since $f(0) = 0;\ \ f(2) = 1,$ we have $y = (0,1)$.

3.5 Testing new input vectors with binary representation

Let's test the heteroassociative network with a new input vector (unseen input vector). Firstly we consider the new input vector $(0\,,0\,,1\,,0,0)$. It is important to know whether a reasonable response is available. For this purpose the Hamming distance is used. The Hamming distance can be used for both binary and bipolar representations. The Hamming distance obtains in how many positions two strings of same length mismatch (disagree). A new input vector is compared with all the four training input vectors:

	$(0\,,0\,,1\,,0\,,0)$
$X(1) = (1\,,0\,,1\,,0\,,0)$	Hamming distance = 1
$X(2) = (0\,,1\,,1\,,0\,,0)$	Hamming distance = 1
$X(3) = (0\,,0\,,0\,,1\,,1)$	Hamming distance = 3
$X(4) = (0\,,0\,,0\,,1\,,0)$	Hamming distance = 2

The new input vector $(0\,,0\,,1\,,0\,,0)$ is closer to the first and second training input vectors (since the Hamming distance is minimum), and is differed from them in just one component, and we check which output is produced after the calculation of xW:

$$(0\,,0\,,1\,,0\,,0)\begin{bmatrix}1 & 0\\ 1 & 0\\ 2 & 0\\ 0 & 2\\ 0 & 1\end{bmatrix} = (2\,,0)$$

Since $f(2) = 1;\ \ f(0) = 0,$ we have the same output of the first and second training pairs $(1, 0)$, which is a desirable result.

Let's test another new input vector $(0\,,0\,,0\,,0\,,1)$, and consider the Hamming distance.

$(0\,,0\,,0\,,0\,,1)$

$X(1) = (1\,,0\,,1\,,0\,,0)$ Hamming distance = 3

$X(2) = (0\,,1\,,1\,,0\,,0)$ Hamming distance = 3

$X(3) = (0\,,0\,,0\,,1\,,1)$ Hamming distance = 1

$X(4) = (0\,,0\,,0\,,1\,,0)$ Hamming distance = 2

The output of the new input vector should be same with the output of the third training pair (minimum Hamming distance). Let's check it:

$$(0\,,0\,,0\,,0\,,1)\begin{bmatrix}1 & 0\\ 1 & 0\\ 2 & 0\\ 0 & 2\\ 0 & 1\end{bmatrix} = (0\,,1)$$

Since $f(0) = 0;\ \ f(1) = 1,$ the output is desirable.

For the testing the last new input vector $(1\,,1\,,1\,,1\,,1)$, we apply the similar procedure.

$(1\,,1\,,1\,,1\,,1)$

$X(1) = (1\,,0\,,1\,,0\,,0)$ Hamming distance = 3

$X(2) = (0\,,1\,,1\,,0\,,0)$ Hamming distance = 3

$X(3) = (0\,,\ 0\,,0\,,1\,,1)$ Hamming distance = 3

$X(4) = (0\,,0\,,0\,,1\,,0)$ Hamming distance = 4

$$(1\,,1\,,1\,,1\,,1)\begin{bmatrix}1 & 0\\ 1 & 0\\ 2 & 0\\ 0 & 2\\ 0 & 1\end{bmatrix} = (4\,,3)$$

But $f(4) = 1;\ \ f(3) = 1,$ and the output is $(1,1)$ which is not similar with the output of any training input vector. So the output is considered as unknown.

Chapter 4

BIPOLAR DATA REPRESENTATION OF A HETEROASSOCIATIVE NET

4.1 Advantages of bipolar vectors compared to binary vectors in data representation

In a training process of a neural network, both binary and bipolar representations of training patterns can be used. Some advantages of a bipolar representation of patterns compared to a binary representation of patterns are given below:

- Bipolar representation is more robust in the presence of noise [19];

- Bipolar representation is better in terms of strength and sign of correction coefficients [20];

- Less training time is required for bipolar vectors in pattern association and classification, i.e. learning process in a neural network using binary vectors takes longer time compare to bipolar vectors;

- In comparison with the binary vectors, the bipolar vectors have greater probability of being orthogonal. If bipolar vectors are selected randomly, the probability to be pairwise orthogonal is higher compared to that randomly selected binary vectors can

be pairwise orthogonal. So not many bipolar vectors should be selected to be sure that pairwise orthogonality exists;

- The bipolar vectors perform greater accuracy than binary vectors while using outer product encoding;

- In the computation of the weight change in Hebbian learning rule, if the training input vector or the target vector is binary, then in updating the weight the result is 0. In other words, it is impossible to distinguish which of the following conditions are met by the input-target pair, if:

- input is 1, and target is 0;

- both input and target are 0.

A pair of bipolar vectors $X(p), Y(p), p = 1, \dots, P$ for a heteroassociative net is stored, where

$$X(p) = (x_1(p), \dots, x_i(p), \dots, x_n(p))$$

and

$$Y(p) = \left(y_1(p), \dots, y_j\,(p), \dots, y_m(p)\right)$$

The weight matrix $W = \{w_{ij}\}$ is calculated as [19]:

$$w_{ij} = \sum_p X_i(p) Y_j(p)$$

4.2 Using bipolar vectors in a heteroassociative network

We should represent binary input and target output vectors given in chapter 3 in the form of bipolar input – output pairs. After the mapping binary vectors onto bipolar vectors, we get the following input-output pairs:

$$X(1) = (1, -1, 1, -1, -1), \quad Y(1) = (1, -1)$$

$$X(2) = (-1, 1, 1, -1, -1), \quad Y(2) = (1, -1)$$

$$X(3) = (-1, -1, -1, 1, 1), \quad Y(3) = (-1, 1)$$

$$X(4) = (-1, -1, -1, 1, -1), \quad Y(4) = (-1, 1)$$

The outer products are used to find the weights in this example.

By using the outer product the first pattern pair for the vectors is stored, and the weight matrix is obtained as follows:

$$X(1) = (1, -1, 1, -1, -1), \quad Y(1) = (1, -1)$$

$$\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} [1 \quad -1] = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}.$$

To store the second pattern pair for the vectors by using the same process, we have the following weight matrix:

$$X(2) = (-1, 1, 1, -1, -1), \quad Y(2) = (1, -1)$$

$$\begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} [1 \quad -1] = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}$$

The weight matrix for the third pattern pair for the vectors is stored using the same procedure, and the weight matrix is obtained as

$$X(3) = (-1\,, -1\,, -1\,, 1\,, 1), \qquad Y(3) = (-1\,, 1)$$

$$\begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}$$

The similar process is used for the last fourth pattern pair for the vectors to be stored, and the weight matrix is obtained as

$$X(4) = (-1\,, -1\,, -1\,, 1\,, -1), \qquad Y(4) = (-1\,, 1)$$

$$\begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix}$$

To store all four pattern pairs, the final weight matrix is obtained by the sum of the weight matrices obtained above:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix}$$

So the final weight matrix is

$$\mathrm{W} = \begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix}$$

Let's test the network using training inputs. For the first training input vector $(1\,, -1\,, 1\,, -1\,, -1)$, we have the following result for xW:

$$(1\,, -1\,, 1\,, -1\,, -1) \begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = (10\,, -10) \rightarrow (1, -1)$$

For the second training input vector $(-1\,,1\,,1\,,-1\,,-1)$, we have the following result for xW:

$$(-1\,,1\,,1\,,-1\,,-1)\begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = (10\,,-10) \rightarrow (1,-1)$$

For the third training input vector $(-1\,,-1\,,-1\,,1\,,1)$, we have the following result for xW:

$$(-1\,,-1\,,-1\,,1\,,1)\begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = (-14\,,14) \rightarrow (-1,1)$$

For the last fourth training input vector $(-1\,,-1\,,-1\,,1\,,-1)$, we have the following result for xW:

$$(-1\,,-1\,,-1\,,1\,,-1)\begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = (-10\,,10) \rightarrow (-1,1)$$

So a net always has correct responses (desirable outputs) for the given input vectors.

4.3 Testing new input vectors with bipolar representation

Firstly we consider the new input vector $(1\,,1\,,1\,,-1\,,-1)$, and measure the Hamming distance:

	$(1\,,1\,,1\,,-1\,,-1)$
$X(1) = (1\,,-1\,,1\,,-1\,,-1)$	Hamming distance = 1
$X(2) = (-1\,,1\,,1\,,-1\,,-1)$	Hamming distance = 1
$X(3) = (-1\,,-1\,,-1\,,1\,,1)$	Hamming distance = 5
$X(4) = (-1\,,-1\,,-1\,,1\,,-1)$	Hamming distance = 4

The new input vector $(1\,,1\,,1\,,-1\,,-1)$ is closer to the first and second training input vectors (since the Hamming distance is minimum), and is differed from them in just one component, and we check which output is produced after the calculation of xW:

$$(1\,,1\,,1\,,-1\,,-1)\begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = (14\,,-14) \rightarrow (1,-1)$$

One can see that the output of the new input vector $(1\,,1\,,1\,,-1\,,-1)$ is same with the output (target) of the first and second training pairs which is (1, -1), and this is a desirable result.

Another new input vector $(-1\,,-1\,,-1\,,-1\,,1)$ to be considered for testing disagrees with the third input vector in just one component. It is found out in comparisons of Hamming distances given below:

$(-1\,,-1\,,-1\,,-1\,,1)$

$X(1) = (1\,,-1\,,1\,,-1\,,-1)$ Hamming distance = 3

$X(2) = (-1\,,1\,,1\,,-1\,,-1)$ Hamming distance = 3

$X(3) = (-1\,,-1\,,-1\,,1\,,1)$ Hamming distance = 1

$X(4) = (-1\,,-1\,,-1\,,1\,,-1)$ Hamming distance = 2

So the desirable output for the new input vector $(-1\,,-1\,,-1\,,-1\,,1)$ should be same with the output of the third training pair (since the Hamming distance is minimum). The calculation carried out is

$$(-1\,,-1\,,-1\,,-1\,,1)\begin{bmatrix} 2 & -2 \\ 2 & -2 \\ 4 & -4 \\ -4 & 4 \\ -2 & 2 \end{bmatrix} = (-6\,,6) \rightarrow (-1,1)$$

We again get the desirable output, since the new input vector $(-1\,,-1\,,-\,1\,,-1\,,1)$ and the third training input vector produce the same output.

Chapter 5

CONCLUSION

Neural network is a branch of computer science, in particular artificial intelligence, and used for modeling human brain. In neural networks the computation of components are performed in a parallel form. A performance improvement of a neural network is achieved through learning process.

One of the important advantages of an artificial neural network is its implementation for accomplishing a task of a pattern association in which an association between input and target output vectors is learned. There exist different learning algorithms for a pattern association.

This master thesis investigates the Hebb rule for a feedforward heteroassociative neural network by determining the optimal weight matrix. The weights are updated after training of each input-output pair by using Hebb algorithm. The updating of weights can be also performed by the outer product learning rule which is simple and useful scheme. The sum of outer products of all the input-output training pairs defines the final weight matrix of a heteroassociative neural network.

A neural network uses both binary and bipolar data representations. Compare to a binary representation of patterns, a bipolar representation provides better

performance. The advantages of bipolar vectors in data representation are also discussed in this thesis.

After testing new unseen input vectors it is concluded that a neural network correctly responses to each training pattern by producing desirable output vector when the Hamming distance between testing and stored input vectors is minimum. The Hamming distance is the number of “mistake” components of two vectors of same length.

REFERENCES

[1] Colin Molter, Utku Salihoglu and Hugues Bersini. (2005). Introduction of a Hebbian unsupervised learning algorithm to boost the encoding capacity of Hopfield networks. *IEEE International Joint Conference on Neural Networks, IJCNN '05. Proceedings, Volume 3*, pp. 1552-1557.

[2] Gal Chechik, Isaac Meilijson, Eytan Ruppin. (2001). Effective Neuronal Learning with Ineffective Hebbian Learning Rules. *Neural Computation, Volume 13, Issue 4*, pp. 817-840.

[3] Genevieve Gorrell and Brandyn Webb. (2005). Generalized Hebbian Algorithm for Incremental Latent Semantic Analysis. *INTERSPEECH, ISCA*, pp.1325-1328.

[4] Ludovic Righetti, Jonas Buchli, Auke Jan Ijspeert. (2006). Dynamic Hebbian learning in adaptive frequency oscillators. *Physica D 216*, pp. 269–281.

[5] Genevieve Gorrell. (2006). Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing. *Proceedings of Interspeech*, pp. 97-104.

[6] Marzban, C., Viswanathan R. (1993). Stochastic neural networks and the weighted Hebb rule. *Proceedings of 1993 International Joint Conference on Neural Networks, IJCNN '93 (Volume: 3)*, pp. 2658 – 2661.

[7] Geoffrey E. Hinton (2003). The Ups and Downs of Hebb Synapses. *Canadian Psychology, Vol. 44, No. 1,* pp. 10-13.

[8] Heylighen, F., Bollen, J. (2002). Hebbian Algorithms for a Digital Library Recommendation System. *International Conference on Parallel Processing Workshops, 2002, Proceedings,* pp. 439-446.

[9] Behnke, S. (1999). Hebbian learning and competition in the neural abstraction pyramid. *International Joint Conference on Neural Networks, IJCNN '99 (Volume: 2),* pp. 1356-1361.

[10] Jitendra Singh Sengar, Niresh Sharma. (2011). Design a Neural Network Based on Hebbian Learning and ART. *International Journal of Computer Science & Technology, Vol. 2, Issue 4,* pp. 157-160.

[11] Reimer Kühn and Ion-Olimpiu Stamatescu. (1999). A two-step algorithm for learning from unspecific reinforcement. *Journal of Physics A: Mathematical and General, Volume 32, Issue 31,* pp. 5749-5762.

[12] Javier R. Movellan. (1990). Contrastive Hebbian Learning in the Continuous Hopfield Model. *Connectionist Models: Proceedings of the Summer School held in San Diego, California,* pp. 10-17.

[13] Cory Stephenson. (2010). Hebbian neural networks and the emergence of minds.

[14] James Ting-Ho Lo. (2010). Unsupervised Hebbian learning by recurrent multilayer neural networks for temporal hierarchical pattern recognition. *44th Annual Conference on Information Sciences and Systems (CISS),* pp. 1-6.

[15] M. Schaefer, W. Dilger. (2003).Training and Holistic Computation of Vector Graphics with Hebbian Bases in Contrast to RAAM Networks. *Proceedings of the International Joint Conference on Neural Networks, Volume 3,* pp. 1667-1672.

[16] H. Sompolinsky. (1987). The theory of neural networks: The Hebb rule and beyond. *Heidelberg Colloquium on Glassy Dynamics Lecture Notes in Physics, Volume 275,* pp 485-527.

[17] Colin Fyfe, Emilio Corchado (2002). Maximum Likelihood Hebbian Rules. *In proceeding of ESANN'2002, 10th European Symposium on Artificial Neural Networks,* pp. 143-148.

[18] Elpiniki Papageorgiou, Chrysostomos Stylios, and Peter Groumpos. (2003). Fuzzy Cognitive Map Learning Based on Nonlinear Hebbian Rule. *AI 2003: Advances in Artificial Intelligence. Lecture Notes in Computer Science, Volume 2903,* pp. 256-268.

[19] Lauren Fausett. (1994). Fundamentals of Neural Networks. Architectures, Algorithms, and Applications. *Prentice-Hall,* p. 461.

[20] Bart Kosko. (1988). Biderectional associative memories. *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 18, No. 1,* pp. 49-60.