10 Associative Memory Networks

10.1 Introductory concepts

Consider the way we are able to retrieve a **pattern** from a partial **key** as in Figure 10–1.

Figure 10–1: A key (left) and a complete retrieved pattern (right)

Imagine a question “what is it” in relation to the right image.

- The hood of the Volkswagen is the **key** to our associative memory neural network and the stored representation of the whole Volkswagen can be thought of as an network **attractor** for all similar keys.
- The key starts a retrieval process which ends in an attractor which contained both the whole car and its name (maybe you go only for the name since the question is “what is it”)

- Storing a memory like the shape of a Volkswagen in an associative memory network and retrieving it, starting with a key, i.e. an incomplete version of the stored memory is the topic of this chapter.

There are **two fundamental types** of the associate memory networks:

- **Feedforward** associative memory networks in which retrieval of a stored memory is a **one-step procedure.**
- Recurrent associative memory networks in which retrieval of a stored memory is a **multi-step relaxation procedure.**

  Recurrent binary associative memory networks are often referred to as the Hopfield networks.

For simplicity we will be working mainly with binary patterns, each element of the pattern having values $\{-1, +1\}$.

Example of a simple binary pattern

$$\boldsymbol{\xi}_M = \quad = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad ; \quad \boldsymbol{\xi} = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ +1 \\ +1 \\ -1 \\ \vdots \\ -1 \\ +1 \\ \vdots \\ -1 \end{bmatrix}$$

10.1.1 Encoding and decoding single memories

The concept of creating a memory in a neural network, that is, memorizing a pattern in synaptic weights and its subsequent retrieval is based on the "read-out" property of the outer product of two vectors that we have studied earlier.

Assume that we have a **pair of column vectors**:

p-component vector	$\boldsymbol{\xi}$	representing the input pattern
m-component vector	$\mathbf{q}$	representing the desired output association with the input pattern

The pair $\{ \boldsymbol{\xi}, \mathbf{q} \}$ to be stored is called a **fundamental memory**.

Encoding a single memory

We **store or encode** this pair in a matrix $\boldsymbol{W}$ which is calculated as an outer product (column $\times$ row) of these two vectors

$$\boldsymbol{W} = \mathbf{q} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} \tag{10.1}$$

Decoding a single memory

The **retrieval or decoding** of the store pattern is based on application of the input pattern $\mathbf{x}$ to the weight matrix $\boldsymbol{W}$. The result can be calculated as follows:

$$\mathbf{y} = \boldsymbol{W} \cdot \boldsymbol{\xi} = \mathbf{q} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} \cdot \boldsymbol{\xi} = ||\boldsymbol{\xi}|| \cdot \mathbf{q} \tag{10.2}$$

The equation says that the decoded vector $\mathbf{y}$ for a given input pattern $\boldsymbol{\xi}$ is proportional to the encoded vector $\mathbf{q}$, the length of the input pattern $\boldsymbol{\xi}$ being the proportionality constant.

10.1.2 Feedforward Associative Memory

The above considerations give rise to a simple **feed-forward associative memory** known also as the **linear associator**.
It is a well-known single layer feed-forward network with m neurons each with p synapses as illustrated in Figure 10–2.

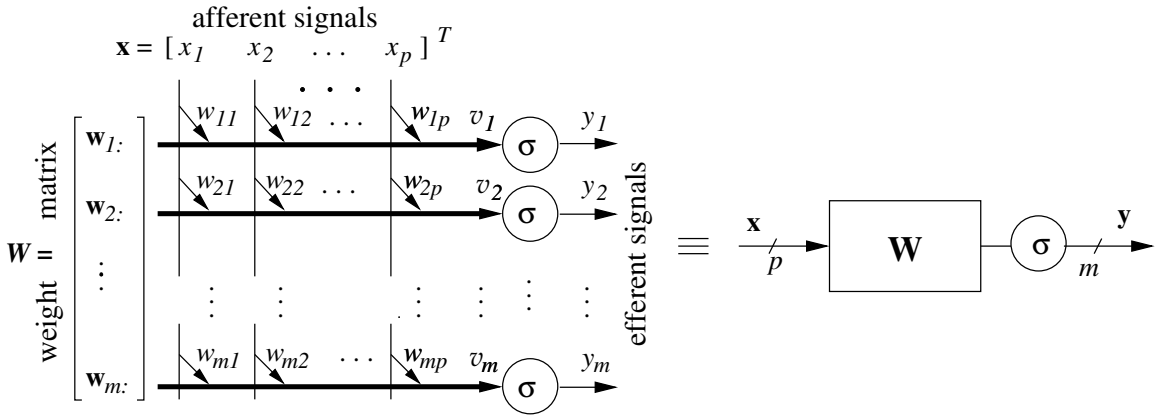


Figure 10–2: The structure of a feed-forward linear associator: $\mathbf{y} = \sigma(\boldsymbol{W} \cdot \mathbf{x})$

For such a simple network to work as an associative memory, the input/output signal are **binary signals** with

$$\{0, 1\} \text{ being mapped to } \{-1, +1\}$$

- During the **encoding phase** the fundamental memories are stored (being encoded) in the weight matrix $\boldsymbol{W}$
- During the **decoding or retrieval phase** for a given input vector $\mathbf{x}$ which is the key to the memory a specific output vector $\mathbf{y}$ is decoded.

10.1.3 Encoding multiple memories

Extending the introductory concepts let us assume that we would like to store/encode N **pairs** of column **vectors** (fundamental memories) arranged in the two matrices:

$\Xi = \xi(1) \ldots \xi(N)$	a matrix of p-component vectors representing the desired input patterns
$Q = \mathrm{q}(1) \ldots \mathrm{q}(N)$	a matrix of m-component vectors representing the desired output associations with the input patterns

In order to **encode** the $\{\Xi, Q\}$ patterns we **sum outer products** of all pattern pairs:

$$W = \frac{1}{N} \sum_{n=1}^{N} \mathrm{q}(n) \cdot \xi^T(n) = \frac{1}{N} Q \cdot \Xi^T \tag{10.3}$$

The sum of the outer products can be conveniently replaced by product of two matrices consisting of the pattern vectors. The resulting $m \times p$ matrix W encodes all the desired N pattern pairs $\mathrm{x}(n), \mathrm{q}(n)$.

Note that eqn (10.3) can be seen as an extension of the Hebb's learning law in which we multiply afferent and efferent signals to form the synaptic weights.

10.1.4 Decoding operation

Retrieval of a pattern is equally simple and involves acting with the weight matrix on the input pattern (the key)

$$\mathbf{y} = \boldsymbol{\sigma}(\boldsymbol{W} \cdot \mathbf{x}) \tag{10.4}$$

where the function $\boldsymbol{\sigma}$ is the sign function:

$$y_j = \sigma(v_j) = \begin{cases} +1 & \text{if } v_j \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{10.5}$$

It is expected that

1. $\mathbf{x} = \boldsymbol{\xi}$

   If the key (input vector) $\mathbf{x}$ is equal to one of the fundamental memory vectors $\boldsymbol{\xi}$, then the decoded pattern $\mathbf{y}$ will be equal to the stored/encoded pattern $\mathbf{q}$ for the related fundamental memory.

2. $\mathbf{x} = \boldsymbol{\xi} + \boldsymbol{n}$

   If the key (input vector) $\mathbf{x}$ can be considered as one of the fundamental memory vectors $\boldsymbol{\xi}$, corrupted by noise $\boldsymbol{n}$ then the decoded pattern $\mathbf{y}$ will be also equal to the stored/encoded pattern $\mathbf{q}$ for the related fundamental memory.

3. $\mathbf{x} \neq \boldsymbol{\xi} + \boldsymbol{n}$

   If the key (input vector) $\mathbf{x}$ is definitely different to any of the fundamental memory vectors $\boldsymbol{\xi}$, then the decoded pattern $\mathbf{y}$ is a spurious pattern.

- The above expectations are difficult to satisfy in a feedforward associative memory network if the number of stored patterns N is more than a fraction of m and p.
- It means that the **memory capacity** of the feedforward associative memory network is **low** relative to the dimension of the weight matrix W.

In general, associative memories also known as **content-addressable memories** (CAM) are divided in two groups:

Auto-associative: In this case the desired patterns Ξ are identical to the input patterns X, that is, $Q = \Xi$. Also $p = m$.

Eqn (10.3) describing encoding of the fundamental memories can be now written as:

$$W = \frac{1}{N} \sum_{n=1}^{N} \xi(n) \cdot \xi^T(n) = \frac{1}{N} \Xi \cdot \Xi^T \tag{10.6}$$

Such a matrix W is also known as the **auto-correlation matrix**.

Hetero-associative: In this case the input Ξ and stored patterns Q and are different.

10.1.5 Numerical examples

Assume that a fundamental memory (a pattern to be encoded) is

$$\boldsymbol{\xi} = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 \end{bmatrix}^{\boldsymbol{T}}$$

The weight matrix that encodes the memory is:

$$\boldsymbol{W} = \boldsymbol{\xi} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

Let us use the following two keys to retrieve the stored pattern:

$$\boldsymbol{W} \cdot \boldsymbol{X} = \boldsymbol{W} \cdot [\mathbf{x}(1)\ \mathbf{x}(2)] = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \\ 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 2 \\ -6 & -2 \\ 6 & 2 \\ 6 & 2 \\ 6 & 2 \\ -6 & -2 \end{bmatrix}$$

$$\boldsymbol{Y} = [\mathbf{y}(1)\ \mathbf{y}(2)] = \boldsymbol{\sigma}(\boldsymbol{W} \cdot \boldsymbol{X}) = \boldsymbol{\sigma}(\begin{bmatrix} 6 & 2 \\ -6 & -2 \\ 6 & 2 \\ 6 & 2 \\ 6 & 2 \\ -6 & -2 \end{bmatrix}) = \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\xi} & \boldsymbol{\xi} \end{bmatrix}$$

- The first key, $\mathbf{x}(1)$, is identical to the encoded fundamental memory, but the other, $\mathbf{x}(2)$, is different from $\boldsymbol{\xi}$ in two positions.
- However, in both cases the retrieved vectors $\mathbf{y}(1), \mathbf{y}(2)$ are equal to $\boldsymbol{\xi}$

Recurrent associative memory

- The capacity of the feedforward associative memory is relatively low, a fraction of the number of neurons.
- When we encode many patterns often the retrieval results in a corrupted version of the fundamental memory.
- However, if we use again the corrupted pattern as a key, the next retrieved pattern is usually closer to the fundamental memory.
- This feature is exploited in the recurrent associative memory networks.

10.2 Recurrent Associative Memory — Discrete Hopfield networks

10.2.1 Structure

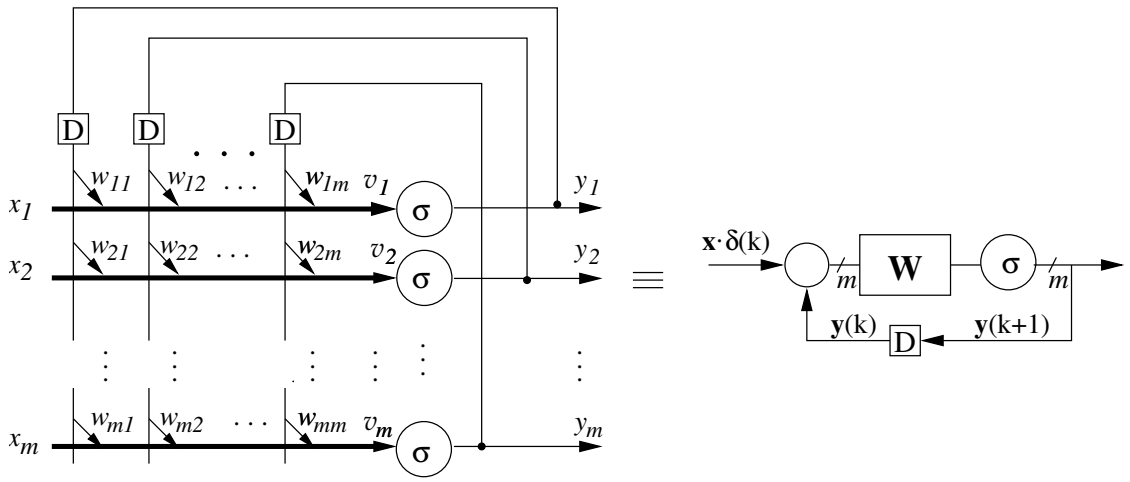


Figure 10–3: A dendritic and block diagram of a recurrent associative memory

- A recurrent network is built in such a way that the output signals are fed back to become the network inputs at the next time step, $\boldsymbol{k}$
- The working of the network is described by the following expressions:

$$\mathbf{y}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{\sigma}\left(\boldsymbol{W}\cdot(\mathbf{x}\cdot\boldsymbol{\delta}(\boldsymbol{k})+\mathbf{y}(\boldsymbol{k})\right) = \begin{cases} \boldsymbol{\sigma}\left(\boldsymbol{W}\cdot\mathbf{x}\right) & \text{for } \boldsymbol{k}=\mathbf{0} \\ \boldsymbol{\sigma}\left(\boldsymbol{W}\cdot\mathbf{y}(\boldsymbol{k})\right) & \text{for } \boldsymbol{k}=\mathbf{1},\mathbf{2},\ldots \end{cases} \tag{10.7}$$

- The function $\boldsymbol{\delta}(\boldsymbol{k})$ is called the Kronecker delta and is equal to one for $\boldsymbol{k}=\mathbf{0}$, and zero otherwise. It is a convenient way of describing the initial conditions, in this case, the initial values of the input signals are equal to $\mathbf{x}(\mathbf{0})$.

- A discrete Hopfield network is a model of an associative memory which works with binary patterns coded with $\{-1, +1\}$

  Note that if $v \in \{0, 1\}$ then $u = 2v - 1 \in \{-1, +1\}$

- The feedback signals $\mathbf{y}$ are often called the state signals.

- During the **storage (encoding) phase** the set of N m-dimensional fundamental memories:

$$\Xi = [\xi(1), \xi(2), \ldots, \xi(N)]$$

  is stored in a matrix $\boldsymbol{W}$ in a way similar to the feedforward auto-associative memory networks, namely:

$$\boldsymbol{W} = \frac{1}{m} \sum_{n=1}^{N} \xi(n) \cdot \xi(n)^T - N \cdot \boldsymbol{I}_m = \frac{1}{m} \Xi \cdot \Xi^T - N \cdot \boldsymbol{I}_m \tag{10.8}$$

  By subtracting the appropriately scaled identity matrix $\boldsymbol{I}_m$ the diagonal terms of the weight matrix are made equal to zero, $(w_{jj} = 0)$. This is required for a stable behaviour of the Hopfield network.

- During the **retrieval (decoding) phase** the key vector $\mathbf{x}$ is imposed on the network as an initial state of the network

$$\mathbf{y}(0) = \mathbf{x}$$

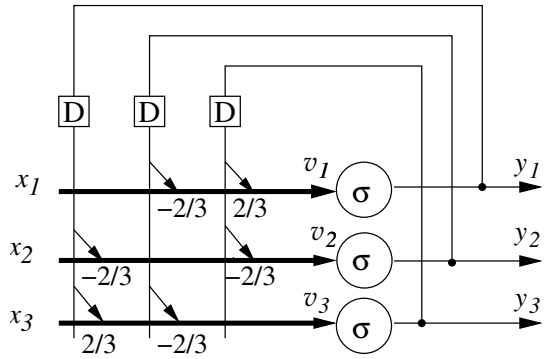
  The network then evolves towards a stable state (also called a fixed point), such that,

$$\mathbf{y}(k + 1) = \mathbf{y}(k) = \mathbf{y}_s$$

  It is expected that the $\mathbf{y}_s$ will be equal to the fundamental memory ξ closest to the key $\mathbf{x}$

10.2.2 Example of the Hopfield network behaviour for $m = 3$

(based on Haykin, *Neural Networks*)
Consider a discrete Hopfield network with three neurons as in Figure 10–4



$$\boldsymbol{W} = \frac{1}{3}\begin{bmatrix} \mathbf{0} & \mathbf{-2} & \mathbf{+2} \\ \mathbf{-2} & \mathbf{0} & \mathbf{-2} \\ \mathbf{+2} & \mathbf{-2} & \mathbf{0} \end{bmatrix}$$

Figure 10–4: Example of a discrete Hopfield network with $\boldsymbol{m = 3}$ neurons: its structure and the weight matrix

- With $\boldsymbol{m = 3}$ neurons, the network can be only in $\mathbf{2^3 = 8}$ different states.
- It can be shown that out of 8 states only two states are stable, namely: $\mathbf{(1, -1, 1)}$ and $\mathbf{(-1, 1, -1)}$. In other words the network stores two fundamental memories
- Starting the retrieval with any of the eight possible states, the successive states are as depicted in Figure 10–5.

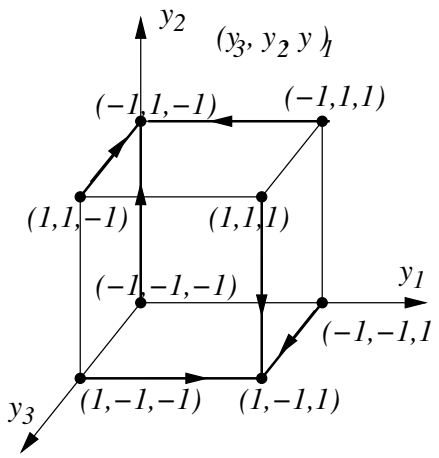


Figure 10–5: Evolution of states for two stable states

Let us calculate the network state for all possible initial states

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{y_3} \\ \boldsymbol{y_2} \\ \boldsymbol{y_1} \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}$$

(The following MATLAB command does the trick: `X = 2*(dec2bin(0:7)-'0')'-1`)

$$\boldsymbol{Y} = \boldsymbol{\sigma}(\boldsymbol{W} \cdot \boldsymbol{X}) = \frac{\mathbf{1}}{\mathbf{3}} \begin{bmatrix} 0 & -2 & +2 \\ -2 & 0 & -2 \\ +2 & -2 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}$$

$$= \boldsymbol{\sigma}\left(\frac{\mathbf{1}}{\mathbf{3}} \begin{bmatrix} 0 & 4 & -4 & 0 & 0 & 4 & -4 & 0 \\ 4 & 0 & 4 & 0 & 0 & -4 & 0 & -4 \\ 0 & 0 & -4 & -4 & 4 & 4 & 0 & 0 \end{bmatrix} \right) = \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

It is expected that after a number of relaxation steps

$$\boldsymbol{Y} = \boldsymbol{W} \cdot \boldsymbol{Y}$$

all patterns converge to one of two fundamental memories as in Figure 10–5.
We will test such examples in our practical work.

Retrieval of numerical patterns stored in a Recurrent Associative Memory (Hopfield network)

(from Haykin, pages 697, 698)

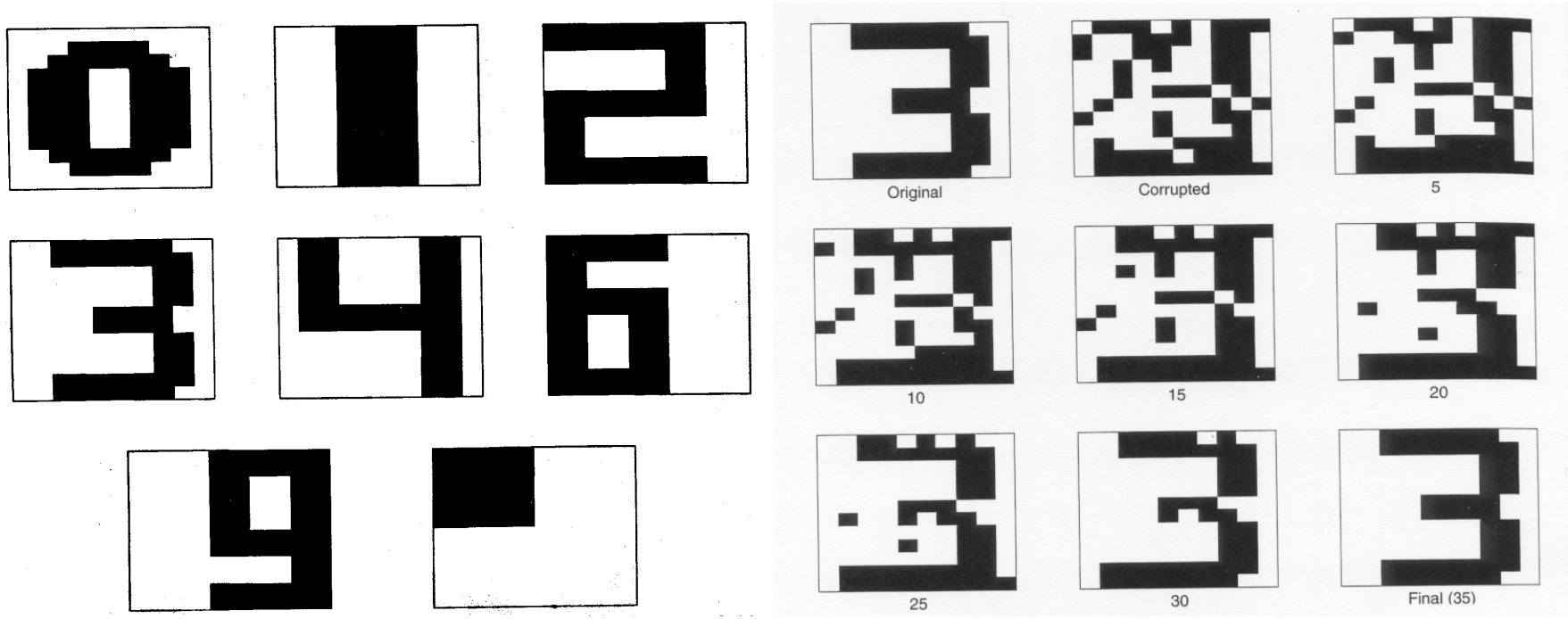


Figure 10–6: Retrieval of numerical patterns in by a Hopfield network

Retrieval from a corrupted pattern (key) succeeds because it was within the basin of attraction of the correct attractor, that is, a stored pattern, or fundamental memory.

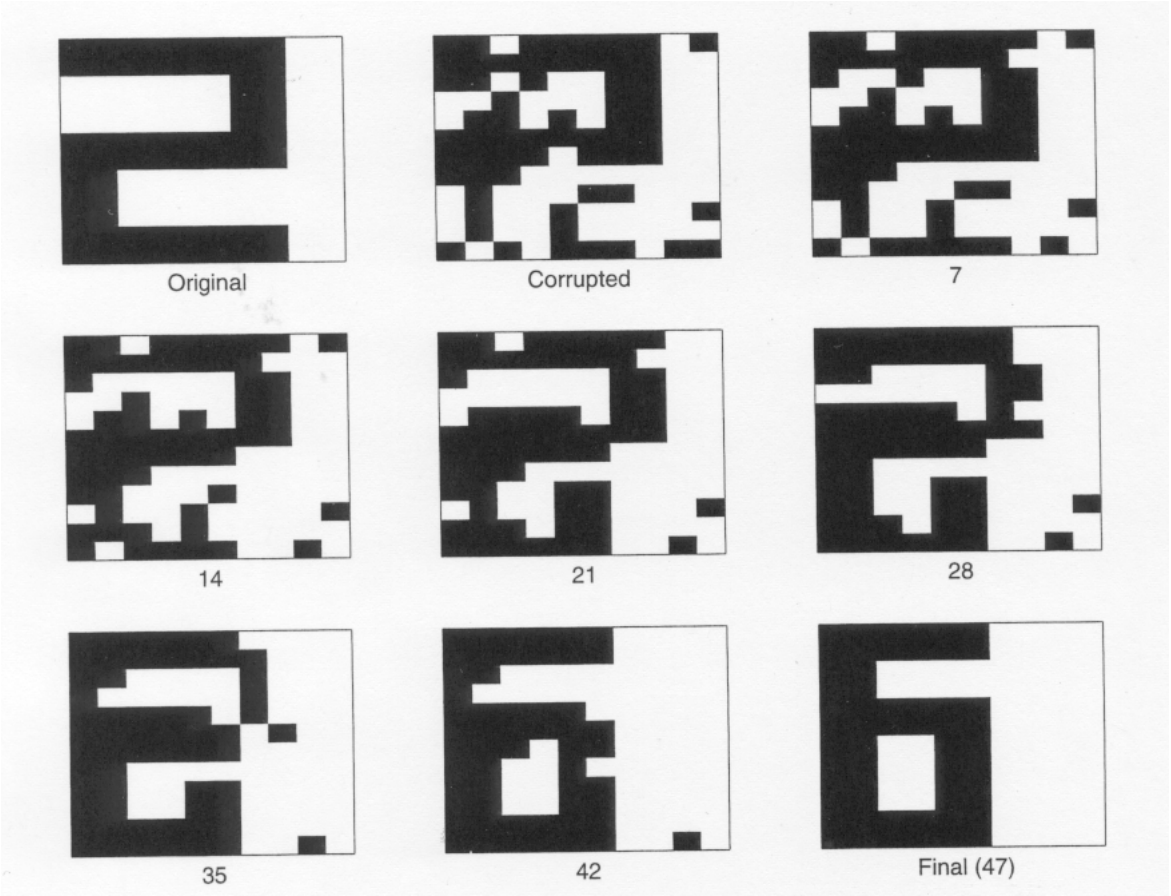
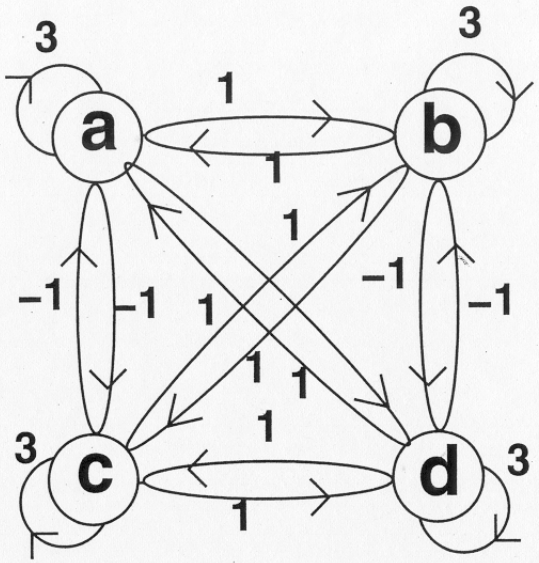


Figure 10–7: Unsuccessful retrieval of a numerical pattern in by a Hopfield network

This time retrieval from a corrupted pattern does not succeed because it was within the basin of attraction of an incorrect attractor, or stored pattern. This is not surprising.

A SIGNAL-FLOW and DENDRITIC REPRESENTATIONS of A DISCRETE HOPFIELD (RECURRENT) NETWORK BINARY



$$W = \begin{bmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & 1 \\ 1 & -1 & 1 & 3 \end{bmatrix}$$

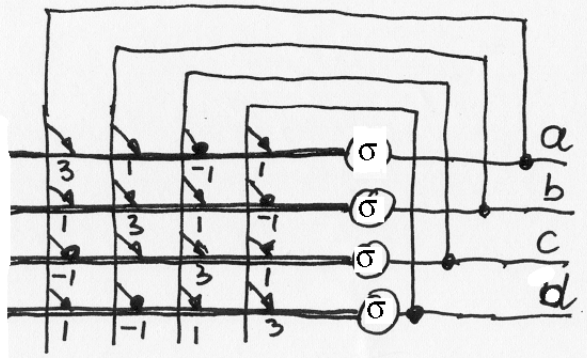


Fig. 10.1: Stick and ball diagram of a four-unit neural network from the summed matrix above. In its weights, this network carries the bittersweet memories of three orthogonal vectors.

(from LYTTON, W.W. From Computer to Brain)

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}(k+1) = \sigma\left(W \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}(k)\right)$$