7 Associative Memory Networks

Related to/based on Lytton's Chapter 10.

7.1 Introductory concepts

Consider the way we are able to retrieve a **pattern** from a partial **key** as in Figure 7–1.

Figure 7–1: A key (left image) and a complete retrieved pattern (right image)

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 (an image) 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:

$$\boldsymbol{\xi}_M = \quad = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{\xi} = 2\boldsymbol{\xi}_M(:) - 1 = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ +1 \\ +1 \\ -1 \\ \vdots \\ -1 \\ +1 \\ \vdots \\ -1 \end{bmatrix}$$

The pattern $\boldsymbol{\xi}$ is a column-scan of the matrix $\boldsymbol{\xi}_M$, $\{0, 1\}$ being replaced with $\{-1, +1\}$.

7.1.1 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**:

n-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

$$\boldsymbol{W} = \mathbf{q} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} \tag{7.1}$$

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:

$$\mathbf{y} = \boldsymbol{W} \cdot \boldsymbol{\xi} = \mathbf{q} \cdot \boldsymbol{\xi}^{\boldsymbol{T}} \cdot \boldsymbol{\xi} = ||\boldsymbol{\xi}|| \cdot \mathbf{q} \tag{7.2}$$

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

7.1.2 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 7–2.

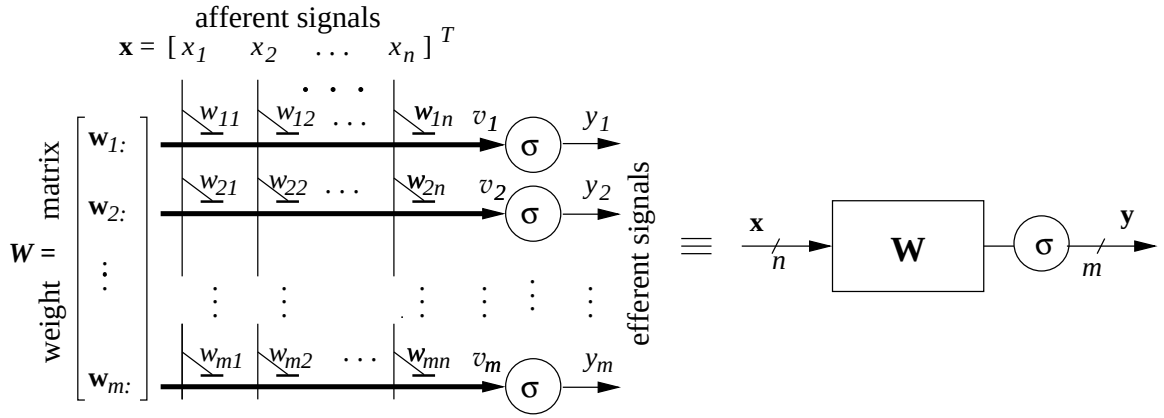


Figure 7–2: The structure of a feed-forward linear associator: $\mathbf{y} = \sigma(W \cdot \mathbf{x})$

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 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.

7.1.3 Encoding multiple memories

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

$\Xi = \xi(1) \ldots \xi(K)$ a matrix of n-component vectors representing the desired input patterns

$Q = \mathrm{q}(1) \ldots \mathrm{q}(K)$ 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:

$$W = \frac{1}{K} \sum_{k=1}^{K} \mathrm{q}(k) \cdot \xi^T(k) = \frac{1}{K} Q \cdot \Xi^T \tag{7.3}$$

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

Note that eqn (7.3) 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.

7.1.4 Decoding operation

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

$$\mathbf{y} = \boldsymbol{\sigma}(\boldsymbol{W} \cdot \mathbf{x}) \tag{7.4}$$

where the function $\boldsymbol{\sigma}$ is the two-valued **sign** function:

$$y_j = \sigma(v_j) = \begin{cases} +1 & \text{if } v_j \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{7.5}$$

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. $\mathbf{x} = \boldsymbol{\xi} + \boldsymbol{\eta}$

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

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

   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 K is more than a fraction of m and n.
- 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 $n = m$.

Eqn (7.3) describing encoding of the fundamental memories can be now written as:

$$W = \frac{1}{K} \sum_{k=1}^{K} \xi(k) \cdot \xi^T(k) = \frac{1}{K} \Xi \cdot \Xi^T \tag{7.6}$$

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.

7.1.5 Numerical examples

Assume that a fundamental memory (a pattern to be encoded) is $\boldsymbol{\xi} = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 \end{bmatrix}^T$
The weight matrix that encodes the memory is:

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

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

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

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

- The first key, $\mathrm{x}(1)$, is identical to the encoded fundamental memory,
  but the other, $\mathrm{x}(2)$, is different from $\boldsymbol{\xi}$ in two positions.

- However, in both cases the retrieved vectors $\mathrm{y}(1), \mathrm{y}(2)$ are equal to $\boldsymbol{\xi}$

7.2 Recurrent Associative Memory — Discrete Hopfield networks

- 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.

7.2.1 Structure

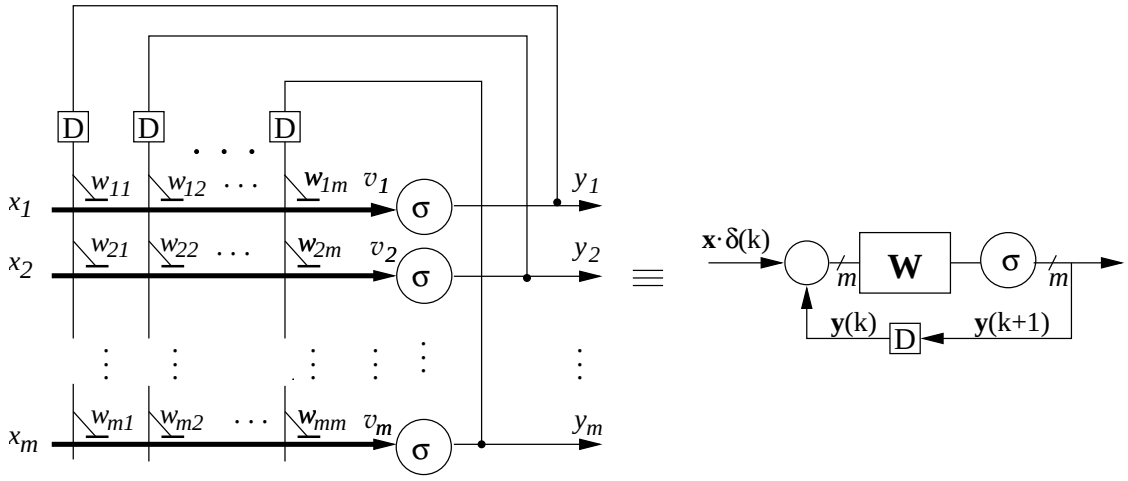


Figure 7–3: A dendritic and block diagram of a recurrent associative memory

- 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, k
- The working of the network is described by the following expressions:

$$\mathbf{y}(k+1) = \boldsymbol{\sigma}\left(W \cdot (\mathbf{x} \cdot \delta(k) + \mathbf{y}(k)\right) = \begin{cases} \boldsymbol{\sigma}\left(W \cdot \mathbf{x}\right) & \text{for } k = 0 \\ \boldsymbol{\sigma}\left(W \cdot \mathbf{y}(k)\right) & \text{for } k = 1, 2, \ldots \end{cases} \tag{7.7}$$

- The function $\delta(k)$ is called the Kronecker delta and is equal to one for $k = 0$, 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 $\mathbf{x}(0)$.
- 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:

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

  is stored in a matrix W in a way similar to the feedforward auto-associative memory networks, namely:

$$\boldsymbol{W} = \frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{k}=\mathbf{1}}^{\boldsymbol{K}} \boldsymbol{\xi}(\boldsymbol{k}) \cdot \boldsymbol{\xi}(\boldsymbol{k})^{\boldsymbol{T}} - \boldsymbol{K} \cdot \boldsymbol{I}_{\boldsymbol{m}} = \frac{\mathbf{1}}{\boldsymbol{m}} \boldsymbol{\Xi} \cdot \boldsymbol{\Xi}^{\boldsymbol{T}} - \boldsymbol{K} \cdot \boldsymbol{I}_{\boldsymbol{m}} \tag{7.8}$$

- By subtracting the appropriately scaled identity matrix $\boldsymbol{I}_{\boldsymbol{m}}$ the diagonal terms of the weight matrix are made equal to zero, $(\boldsymbol{w}_{\boldsymbol{jj}} = \mathbf{0})$.

  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

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

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

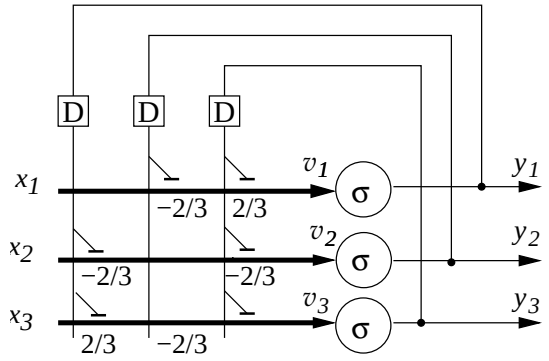
$$\mathbf{y}(\boldsymbol{k}+\mathbf{1}) = \mathbf{y}(\boldsymbol{k}) = \mathbf{y}_{\boldsymbol{s}}$$

  It is expected that the $\mathbf{y}_{\boldsymbol{s}}$ will be equal to the fundamental memory $\boldsymbol{\xi}$ closest to the key $\mathbf{x}$

7.2.2 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 7–4



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

Figure 7–4: Example of a discrete Hopfield network with $m = 3$ neurons: its structure and the weight matrix

- With $m = 3$ binary neurons, the network can be only in $2^3 = 8$ different states.
- It can be shown that out of 8 states only two states are stable, namely: $(1, -1, 1)$ and $(-1, 1, -1)$. 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 7–5.

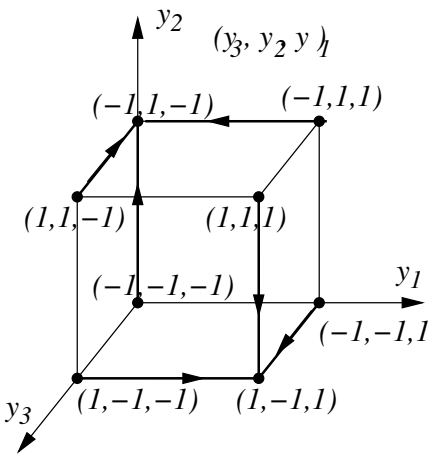


Figure 7–5: Evolution of states for two stable states

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`)

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

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

It is expected that after a number of relaxation steps

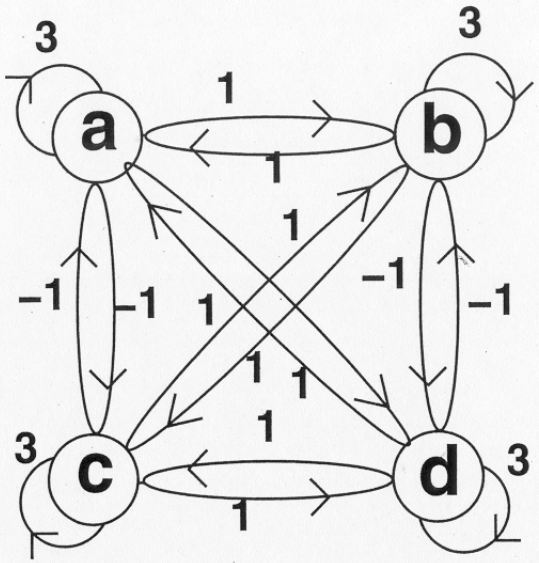
$$\boldsymbol{Y} = \boldsymbol{W} \cdot \boldsymbol{Y}$$

all patterns converge to one of two fundamental memories as in Figure 7–5.
We will test such examples in our practical work.

7.2.3 Another example of Hopfield network (from Lytton)

- The Hopfield network, or a recurrent binary associative memory consists of four neurons, each with four synapses.
- The example demonstrate the relationship between the dendritic and the flow diagram representations.
- Note that the weight matrix has the non-zero terms on the main diagonal therefore the stable pattern retrieval is not guaranteed.

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



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

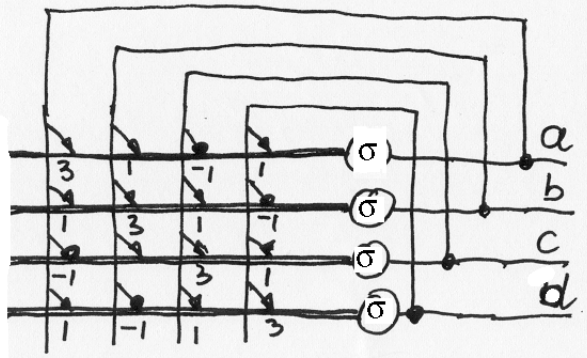


Fig. 10.1: Stick and ball diagram of a four-unit neural network from the summed matrix above. In its weights, this network carries the bittersweet memories of three orthogonal vectors.

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

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

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

(from Haykin, pages 697, 698)

The network consists of $m = 120$ neurons therefore $m^2 = 14,400$ synapses (synaptic weights) and was designed to retrieve eight digit-like patterns coded $+1$ for the black pixel and -1 for the white pixel (left part of the figure).

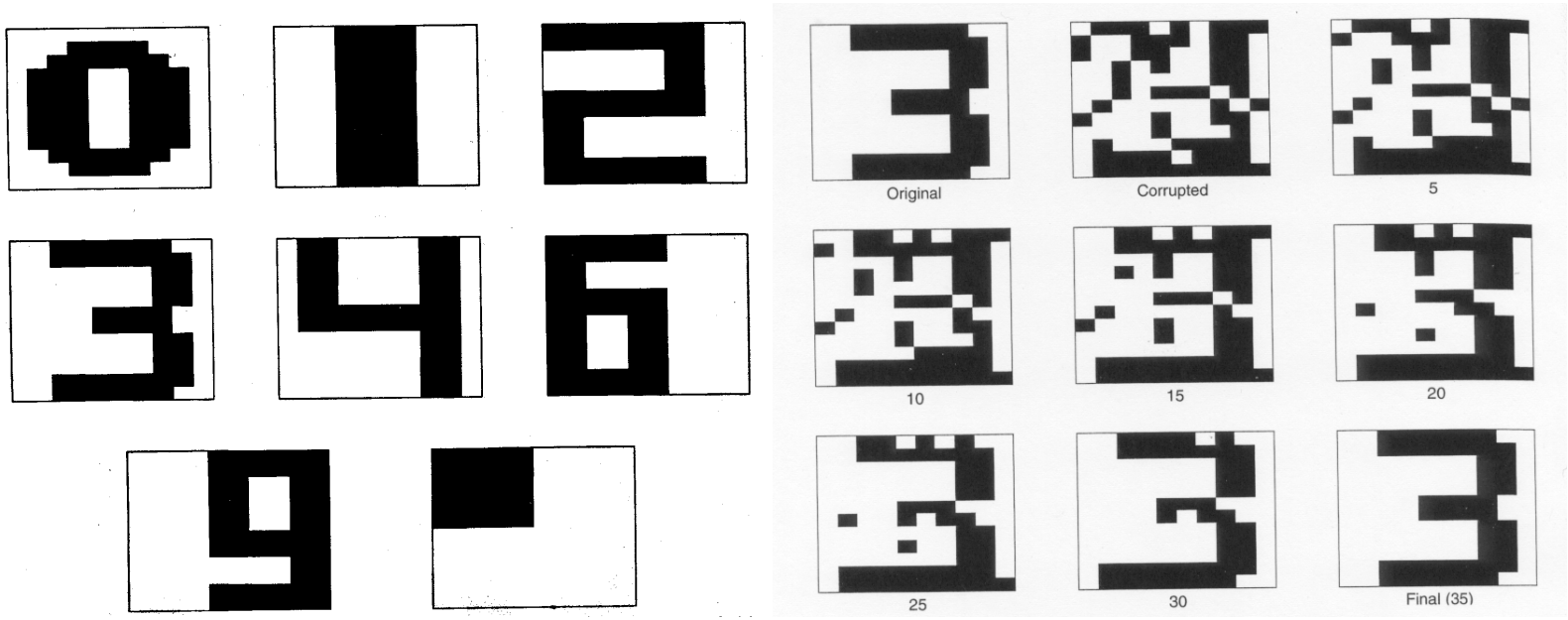


Figure 7–6: Retrieval of numerical patterns in by a Hopfield network

- To demonstrate error-correcting capability of the network, a corrupted pattern representing ‘3’ was applied to the network. After 35 iterations the output pattern was the perfect re-call of the pattern.
- 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.

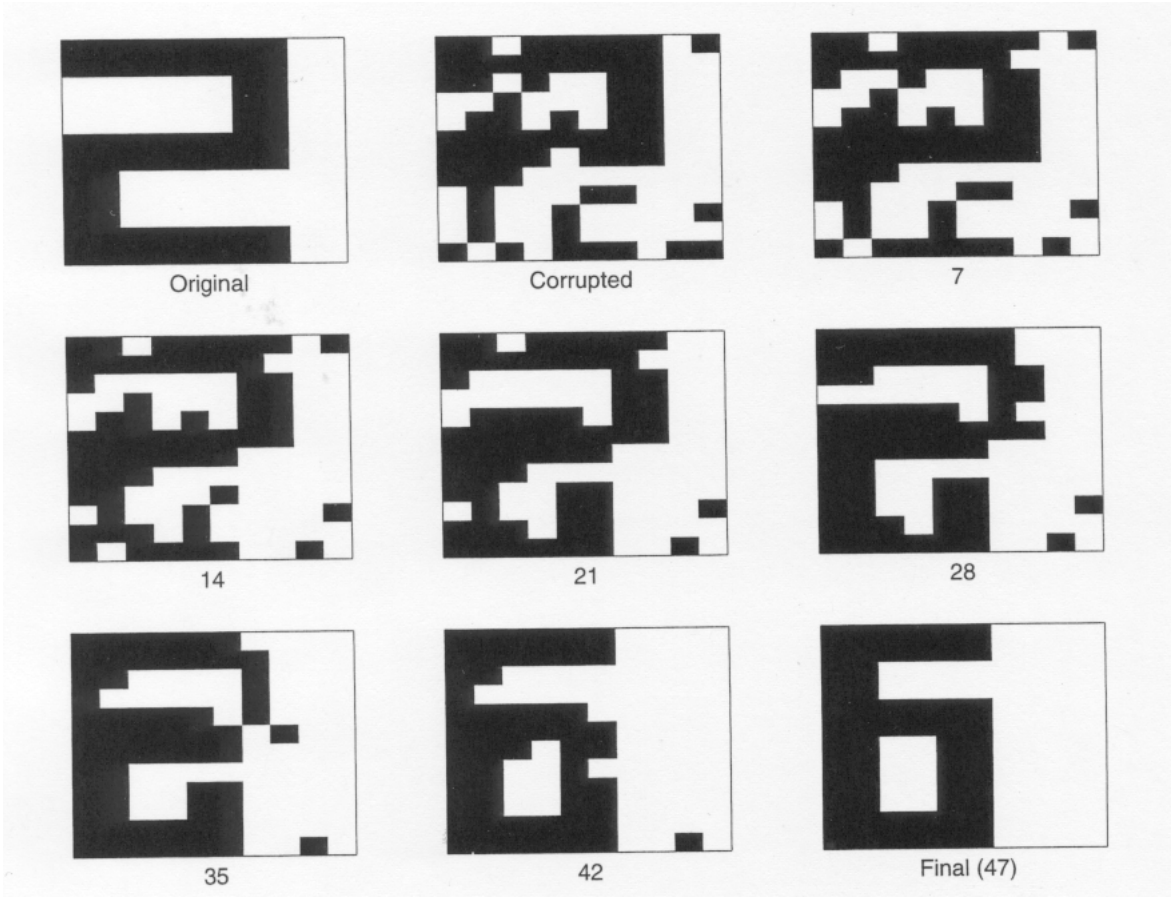


Figure 7–7: Unsuccessful retrieval of a numerical pattern in by a Hopfield network

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.

In addition this network stores at least 108 spurious attractors found in computer simulations:

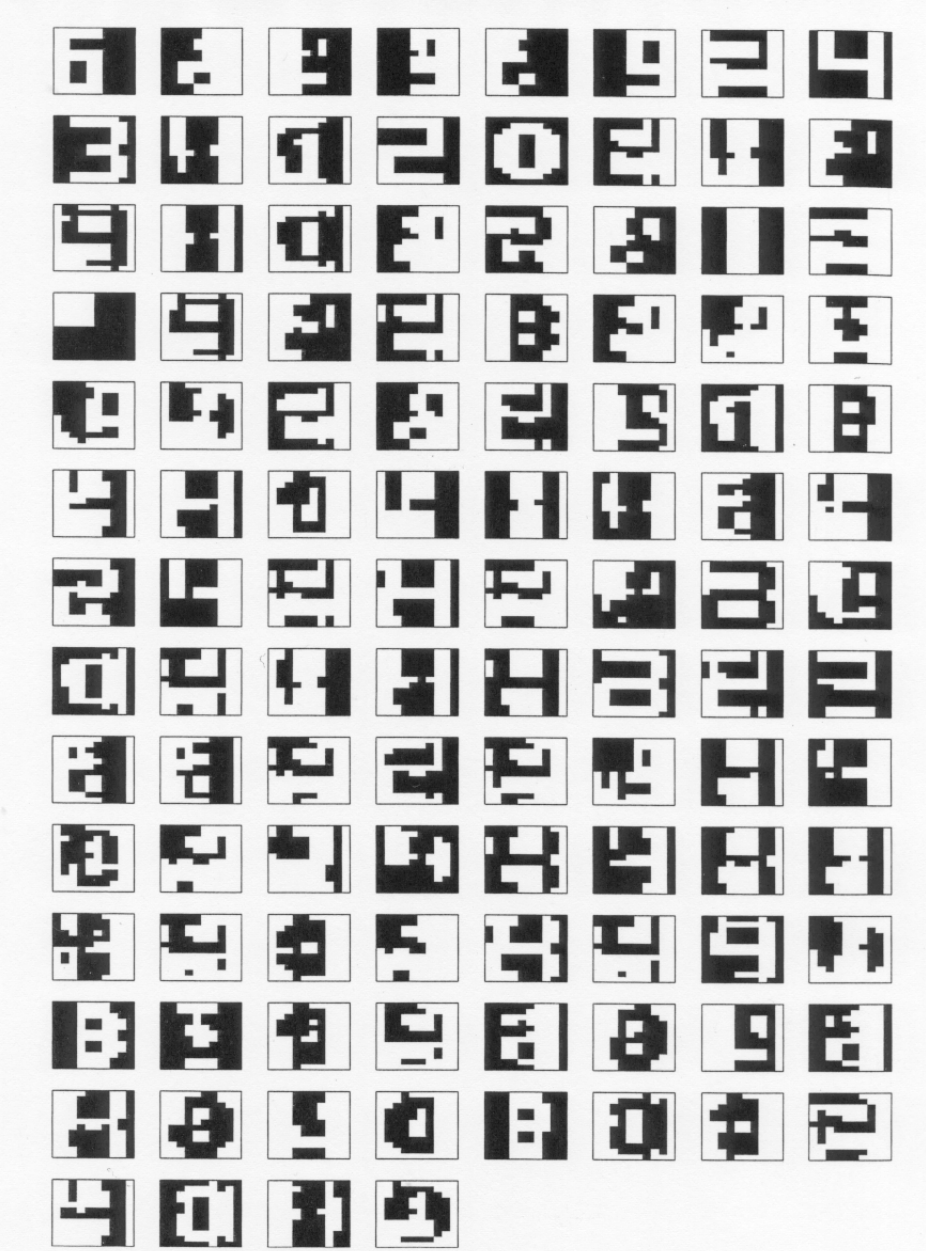


FIGURE 14.20 Compilation of the spurious states produced in the computer experiment on the Hopfield network.

- Then what does all this mean?
- It means that you cannot store memories that are similar to each other,

  because if you have a slightly corrupted version of one of two similar memories,

  then you can easily end up in the other one.
- It also means that even if the stored memories are not similar to each other, there will be other, spurious memories "in-between".
- And if your corrupted initial input is closer to its own fundamental memory than to all the other fundamental memories

  it is still not certain that the proper fundamental memory will be retrieved,

  it might well be a spurious attractor instead.
- The risk of retrieving the opposite of a fundamental memory is usually not great — your initial input has to be very corrupted for that to happen.
- All this means that the associative memory networks as we have described them are far from ideal from a legal witness point of view.
- But their shortcomings are not unheard of from human experience.
- So their shortcomings do not rule them out as first-order models of human memory.

7.3 The energy landscape

- There is an "energy" associated with the states of a recurrent associative memory network.
- It is called energy because Hopfield, who used the energy concept to describe the retrieval process in an associative memory network (in the beginning of the 1980's) is a physicist and saw the purely formal similarity with energy functions in mechanics.
- Each attractor gives rise to a minimum, i.e. a lower point than its immediate surroundings, in this energy landscape.
- If the retrieval process starts from a corrupted memory, then it starts at a high energy and, like a ball in a real landscape, it rolls down to a minimum, hopefully to the right one.
- A problem is that the ball rolls in a high-dimensional landscape, which makes it difficult to illustrate on paper.
- The energy of the spurious attractors is generally higher than the energy of the fundamental memories, so if you can "feel this energy" then you have a chance to say that what you seem to remember might be wrong.
- The opposite attractors of the fundamental memories have the same low energies as the attractors themselves so in this case you are left without assistance.

The energy associated with a particular state $\mathbf{y}$ is defined as:

$$\boldsymbol{E} = -\frac{\mathbf{1}}{\mathbf{2}} \sum_{i=1}^{m} \sum_{j=1}^{m} \boldsymbol{w}_{ji}\, \boldsymbol{y}_i\, \boldsymbol{y}_j = -\frac{\mathbf{1}}{\mathbf{2}}\, \mathbf{y}^{\boldsymbol{T}} \cdot \boldsymbol{W} \cdot \mathbf{y} \quad (\boldsymbol{w}_{ii} = \mathbf{0})$$

where $\boldsymbol{m}$ is the number of neurons, each with $\boldsymbol{m}$ synapses.

The minus sign ensures that we have minima for the "ball to roll into", rather than peaks to climb. The main diagonal terms in the weight matrix should be zero to ensure that a stable solution can be attained.

Example of an imaginary energy landscape (from Lytton):

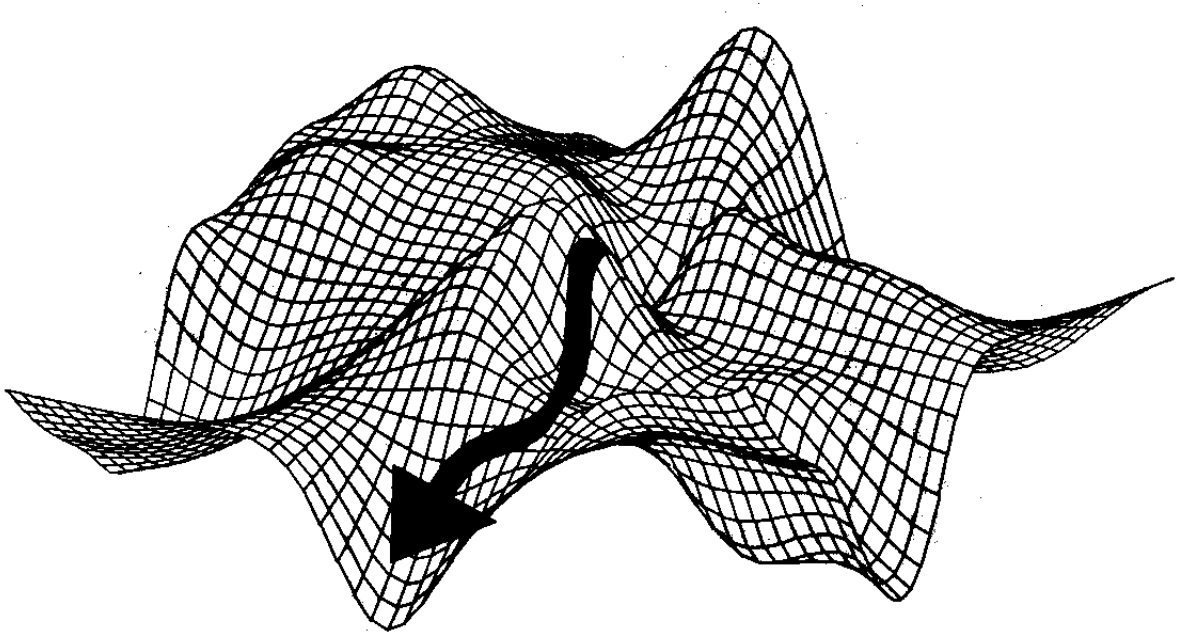
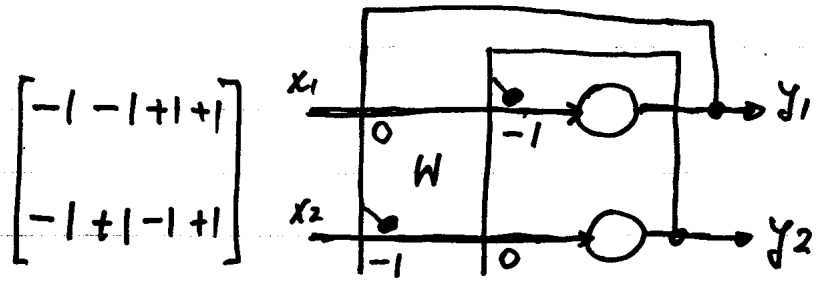


Fig. 10.2: A made-up memory landscape for a two-dimensional system. Starting on top of a hill, the update rule will produce a trajectory moving downhill to recover a memory (arrow).

7.3.1 Example of a two-neuron recurrent associative memory



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

The four possible states of a two-dimensional memory network are shown.
One is a fundamental memory, one is its opposite and two lie on a limit cycle as Lytton sees it.

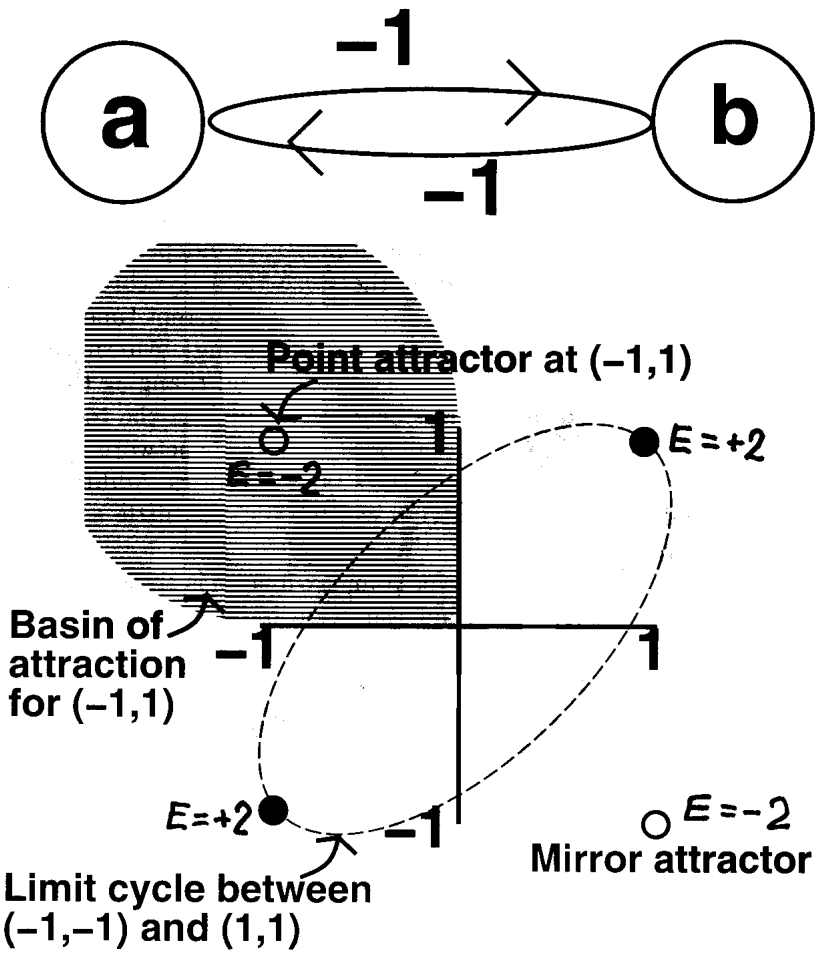


Fig. 10.3: A simple two-unit mutual inhibition network and the corresponding state space diagram. The point attractor represents the autoassociative memory. Half of the limit cycle is the heteroassociative memory. The network also has two spurious memories: a mirror of the point attractor at (1,−1), and the other half of the limit cycle, which maps $(1 \quad 1) \rightarrow (-1 \quad -1)$. Only one of the three basins of attraction is shown, for simplicity.

Chapter 10 Associative Memory Networks

A higher dimensional example

Let us study an example. We begin by storing one vector as a fundamental memory. The particular choice of vector is not important , let us choose $\xi 1$ = [1 1 1 1 1 1 1 1 1 1]'.

The energy minimum has two minima at E = -90, one for x = [1 1 1 1 1 1 1 1 1 1]' and one for x = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]', just as was stated before.

There are several more energy levels, but the levels are discrete since the elements can only assume the values 1 and –1.

If we store two vectors, $\xi 1$ = [1 1 1 1 1 1 1 1 1 1]' and $\xi 2$ = [1 1 1 1 1 -1 -1 -1 -1 -1]', then as expected we have minima at E = -80 for $\xi 1$ and $\xi 2$ and their "opposites".

Chapter 10 Associative Memory Networks

A higher dimensional example

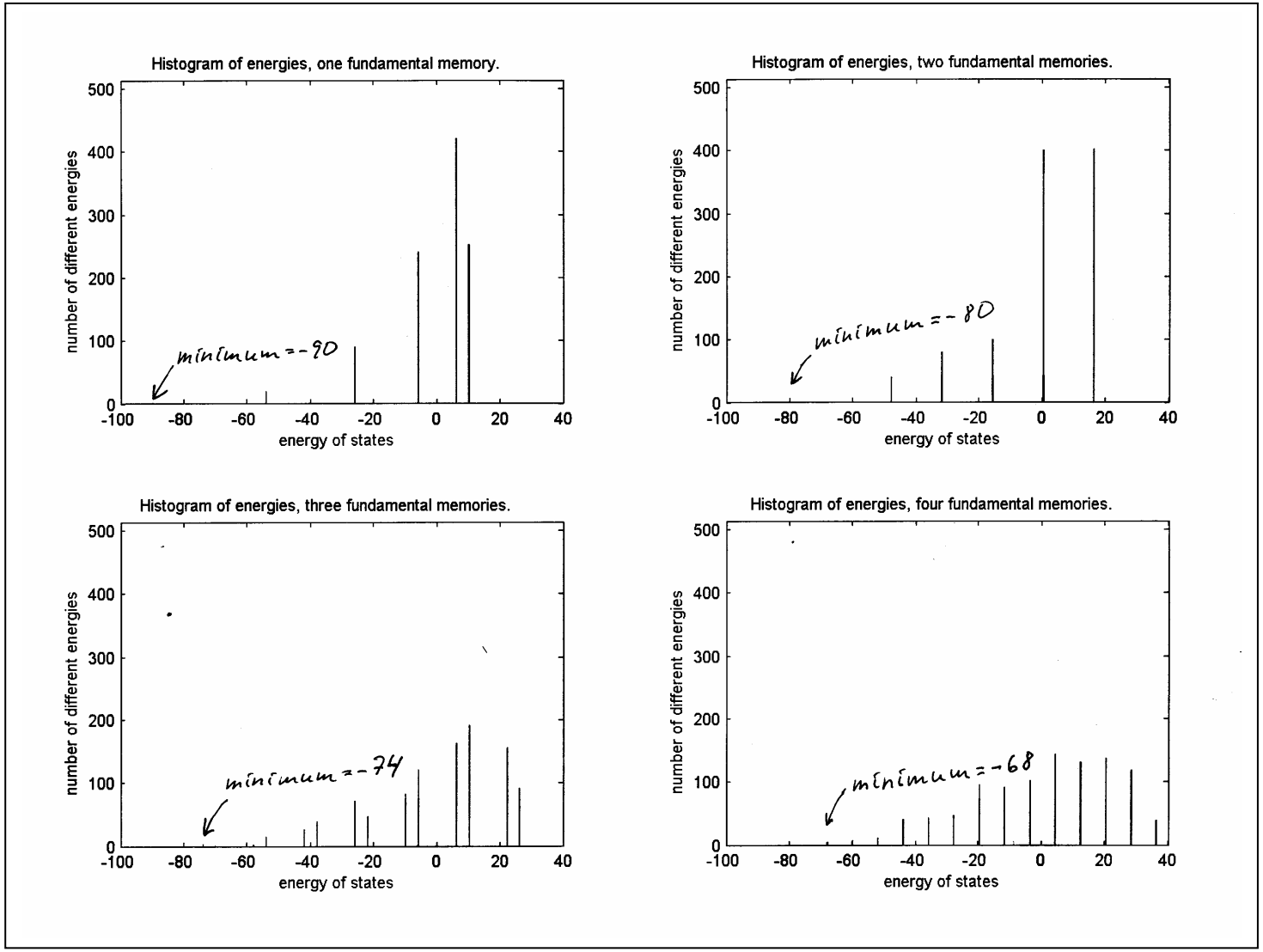
If we store three vectors, $\xi 1$ = [1 1 1 1 1 1 1 1 1 1]' and $\xi 2$ = [1 1 1 1 1 -1 -1 -1 -1 -1]' and $\xi 3$=[1 -1 1 -1 1 -1 1 -1 1 -1]', then we have minima at E = -74 but only at $\xi 2$ and $\xi 3$. The energy at $\xi 1$ is slightly higher, at –70.

If we store four vectors, $\xi 1$ = [1 1 1 1 1 1 1 1 1 1]' and $\xi 2$ = [1 1 1 1 1 -1 -1 -1 -1 -1]', $\xi 3$ = [1 -1 1 -1 1 -1 1 -1 1 -1]' and $\xi 4$ = [1 1 -1 -1 1 1 -1 -1 -1 1]', then we have minima at E = -68 but only at $\xi 2$, $\xi 3$ and $\xi 4$. Again the energy at $\xi 1$ is slightly higher, at –60.

There are more differences between are four cases which become clear when we study the histograms for the energies. Let us do that.

Chapter 10 Associative Memory Networks

A higher dimensional example



Chapter 10 Associative Memory Networks

A higher dimensional example

We see that as the number of fundamental memories increase we get many more intermediate energy levels. Also there is much less difference between the minimum energy levels and other low energy levels. In other words, the energy minima of the attractors are not as distinct when we have more fundamental memories.

We can also expect some spurious attractors to have appeared, i.e. attractors that don't correspond to fundamental memories (others than opposite attractors, which are always present).

We might wonder if we have lost $\xi 1$ as a fundamental memory, but that is happily not the case. If we let the input to the network be $x = \xi 1$ then the output will be $\xi 1$. This is of course not a big feat, but we haven't lost the fundamental memory.

If we let the input be $\xi 1$ with any one element changed from 1 to –1 we can calculate the energy level at – 36, thus all these slightly corrupted versions of $\xi 1$ lie on a higher energy level than $\xi 1$ itself. We would expect to retrieve $\xi 1$ from such corrupted versions.