Outlook

- Networks with feedback connections
- Associative memory
- Hopfield model

- Self-organizing maps
- Dimensionality reduction
- Principal Component Analysis
- Independent Component Analysis

Remember: Eight major aspects of a PDP model

- a *set of processing units*
- the current *state of activation*
- an *output function* for each unit
- a *pattern of connectivity* between units
- a *propagation rule* for propagatin patterns of activities through the network of connectivities
- an *activation rule* for combining the inputs impinging on a unit
- a *learning rule* whereby patterns of connectivity are modified by experience
- the *environment* within the system must operate

(Rumelhard, McClelland, et.al. 1986)

Remember: Multi-layer perceptrons

Multi-layer perceptrons:

- map an input-space into an output space
- using only feed-forward computations

- continous neuron activation functions (e.g. sigmoid)
- backpropagation learning algorithm

- therefore, neighborhood of input vector x is mapped to a neighborhood of output vector y (image of x)

- *continous mapping networks*

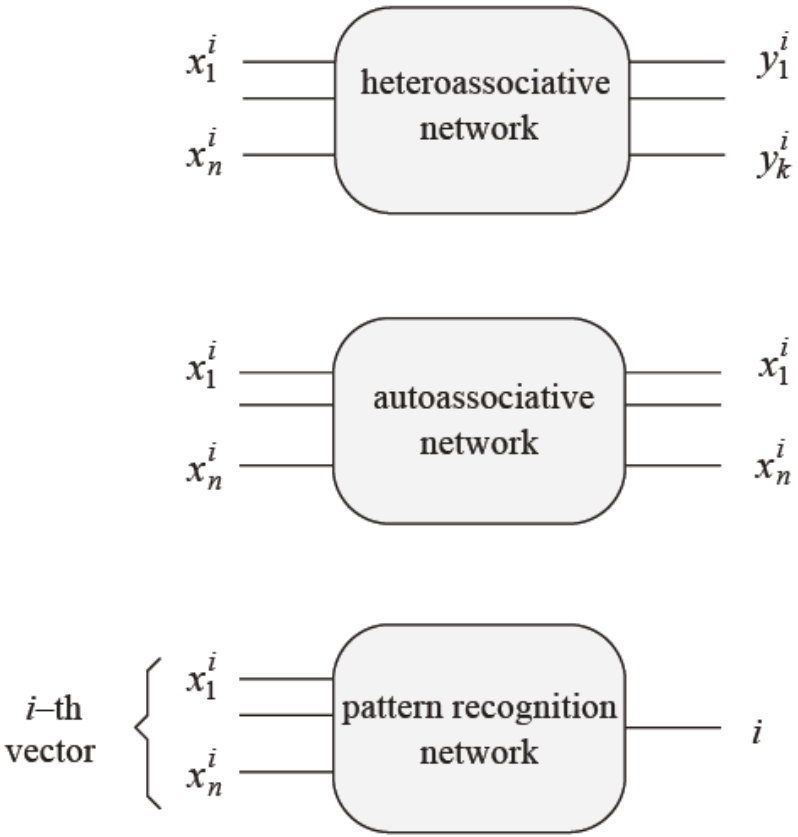
Associative memories

- goal of learning is to *associate* known-input vectors (*patterns*) with given output vectors

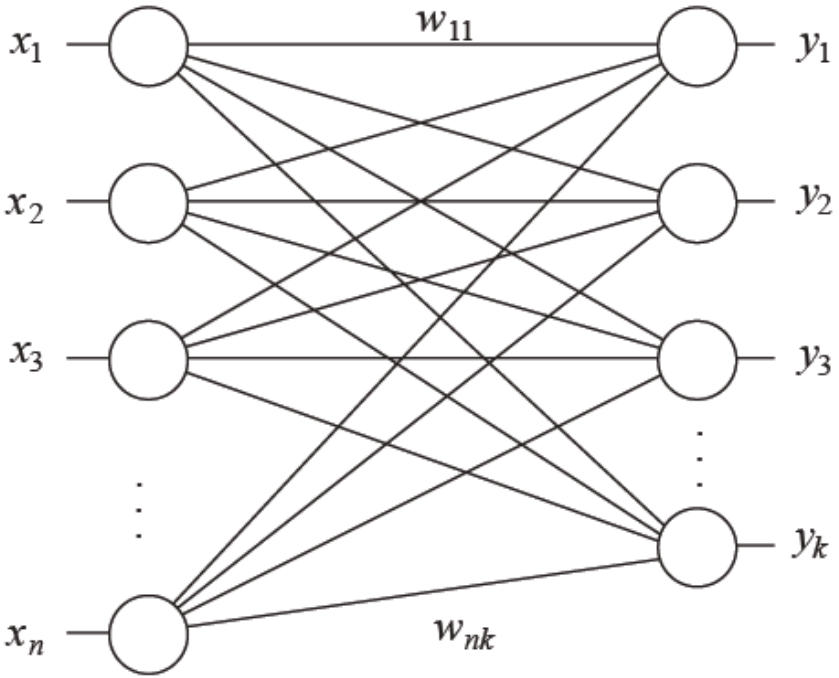
- input vectors x' near to x should be mapped to y
- e.g., noise-reduction: noisy input vectors are mapped to the original patterns
- similar to clustering algorithms

- implemented with networks with or without feedback
- but feedback networks produce better results

Variants of associative networks



BAM: network



- hetero-associative: $x \mapsto y \mapsto x' \mapsto y' \mapsto x'' \mapsto y'' \ldots$

(Kosko 1988)

Auto-associative memory

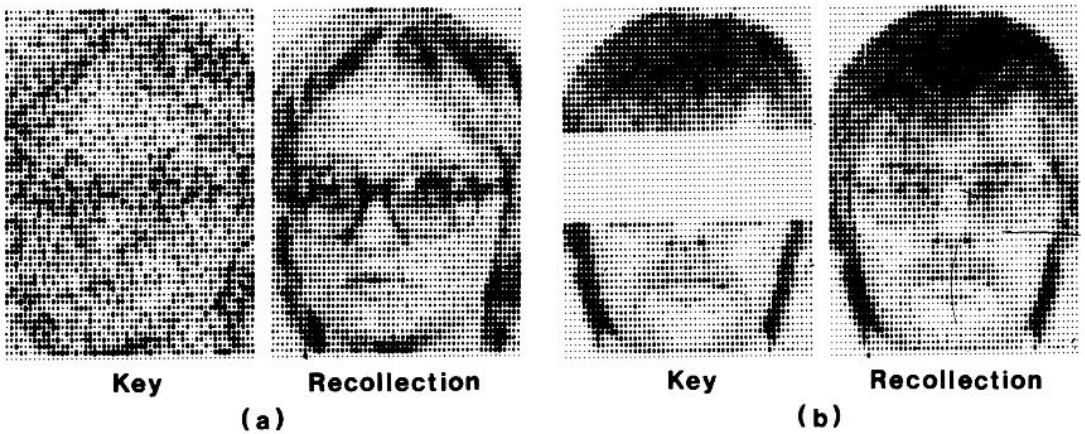
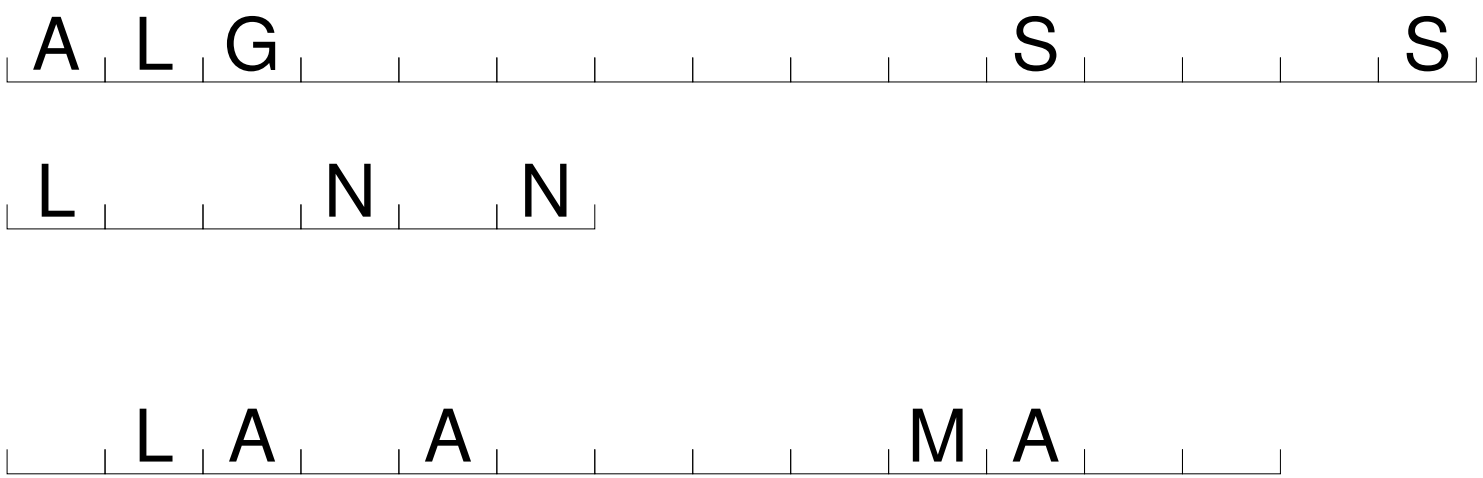


Fig. 6.2a – b. Demonstration of noise suppression and autoassociative recall in the orthogonal projection operation

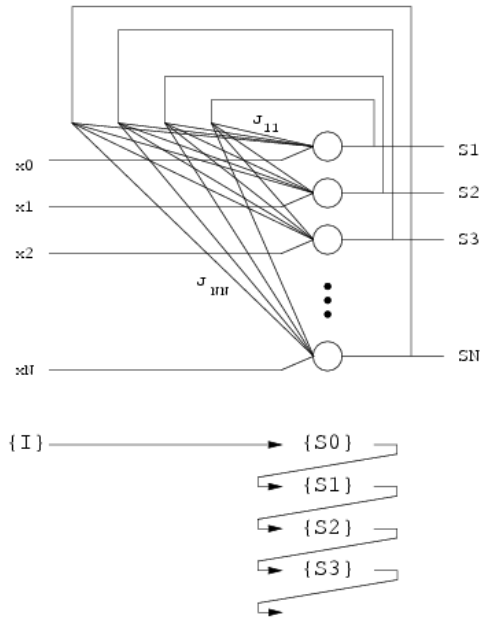
- complete whole memory from a little fragment
- complete whole memory from noisy input
- considered a main function of the human cortex

Auto-associative memory



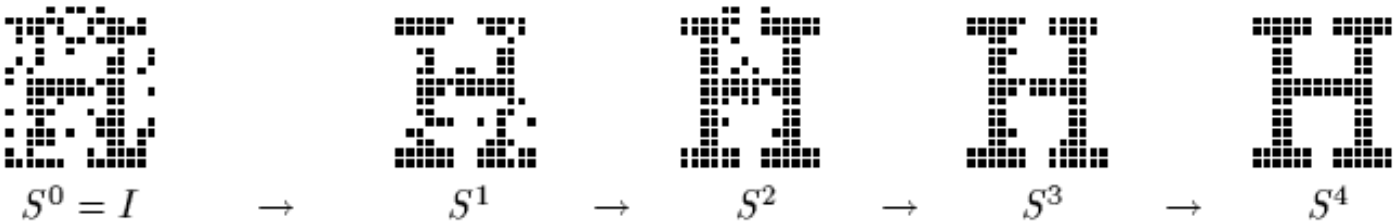
- popular example: word-completion task
- first and last letters more important than middle letters
- etc.

Pure feedback network: architecture



- network initialized with external input, $S(t = 0) = I$
- next states calculated via neuron activation
- output readout once stable (attractor states)
- output readout after fixed number of iterations

Feedback network: iterations



- example: pattern-recognition, pattern-completion
- network state updated several times
- iterative improvement of current network state
- more economical than feed-forward network with many layers

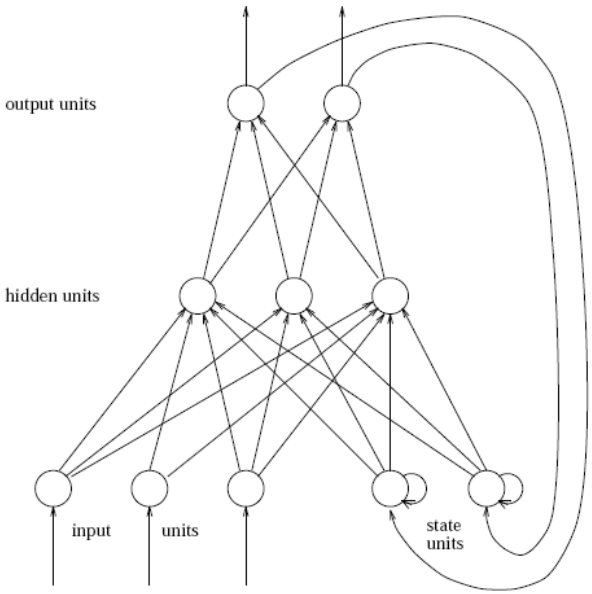
Feedback networks: unfolding

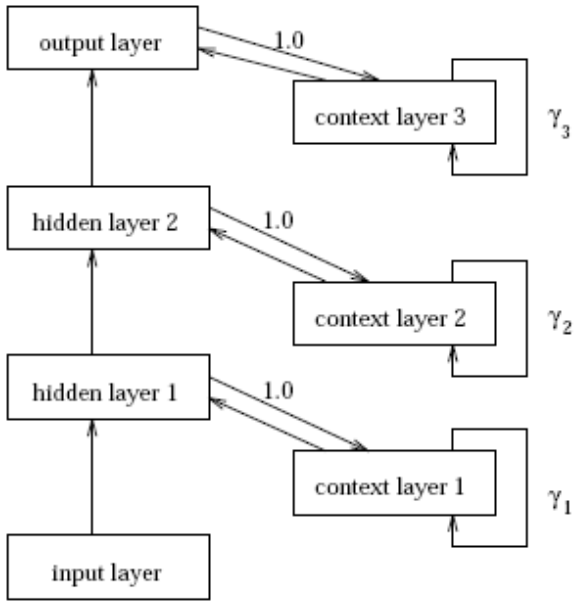
The simplest way to deal with a recurrent network is to consider a finite number of iterations only. Assume for generality that a network of n computing units is fully connected and that w_{ij} is the weight associated with the edge from node i to node j. By unfolding the network at the time steps $1, 2, \ldots, T$, we can think of this recurrent network as a feed-forward network with T stages of computation. At each time step t an external input $\mathbf{x}(t)$ is fed into the network and the outputs $(o_1^{(t)}, \ldots, o_n^{(t)})$ of all computing units are recorded. We call the n-dimensional vector of the units' outputs at time t the network state $\mathbf{o}^{(t)}$. We assume that the initial values of all unit's outputs are zero at $t = 0$, but the external input $\mathbf{x}(0)$ can be different from zero. Figure 7.24 shows a diagram of the unfolded network. This unfolding strategy which converts a recurrent network into a feed-forward network in order to apply the backpropagation algorithm is called *backpropagation through time* or just BPTT [383].

network with feedback connections:

- equivalent to *n*-layer network with same forward connections
- but only for fixed number of feedback iterations

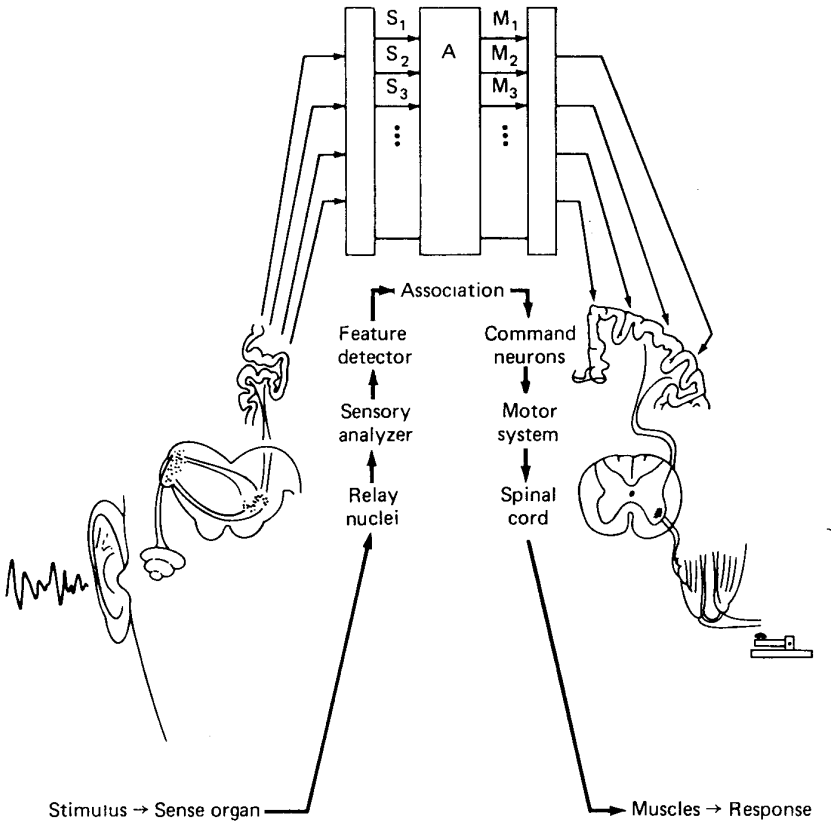
Jordan-network, Elman-network





- feed-forward architecture with some feedback connections
- internal *state* (*context*)
- complex behaviour, see SNNS/JavaNNS for implementation

Brain modeling: mostly associative?



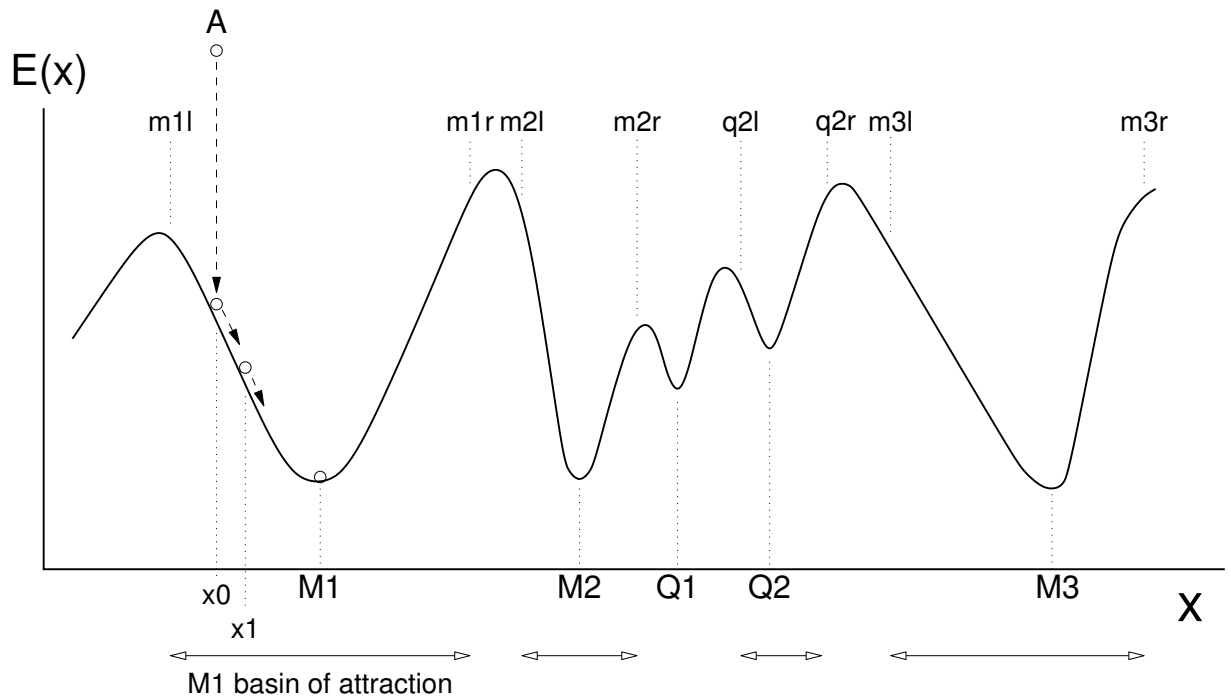
(Amit 1986)

Network state evolution

- network initialized in (random) input state
- will the network stabilize at all?
- will the network converge to the nearest attractor?
- how fast will the network converge?

- *basins of attraction*

Energy landscape model



Basins of attraction: single iteration

Table 12.1. Percentage of 10-dimensional vectors with a Hamming distance (H) from 0 to 4, which converge to a stored vector in a single iteration. The number of stored vectors increases from 1 to 10.

	Number of stored vectors (dimension 10)									
H	1	2	3	4	5	6	7	8	9	10
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1	100.0	100.0	90.0	85.0	60.0	60.0	54.3	56.2	45.5	33.0
2	100.0	86.7	64.4	57.2	40.0	31.8	22.5	23.1	17.0	13.3
3	100.0	50.0	38.6	25.4	13.5	8.3	4.8	5.9	3.1	2.4
4	100.0	0.0	9.7	7.4	4.5	2.7	0.9	0.8	0.3	0.2

- example network, $n = 10$, $p = 1 \ldots 4$ stored patterns

Basins of attraction: five iterations

Table 12.2. Percentage of 10-dimensional vectors with a Hamming distance (H) from 0 to 4, which converge to a stored vector in five iterations. The number of stored vectors increases from 1 to 7.

	Number of stored vectors (dimension 10)						
H	1	2	3	4	5	6	7
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1	100.0	100.0	90.0	85.0	60.0	60.0	54.3
2	100.0	100.0	72.6	71.1	41.8	34.8	27.9
3	100.0	80.0	48.6	47.5	18.3	10.8	8.5
4	100.0	42.8	21.9	22.3	6.8	3.7	2.0

Basins of attraction: multiple iterations

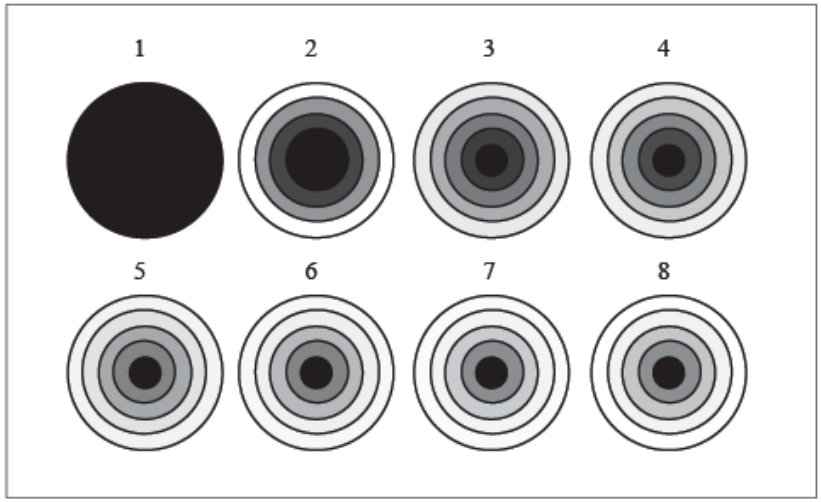
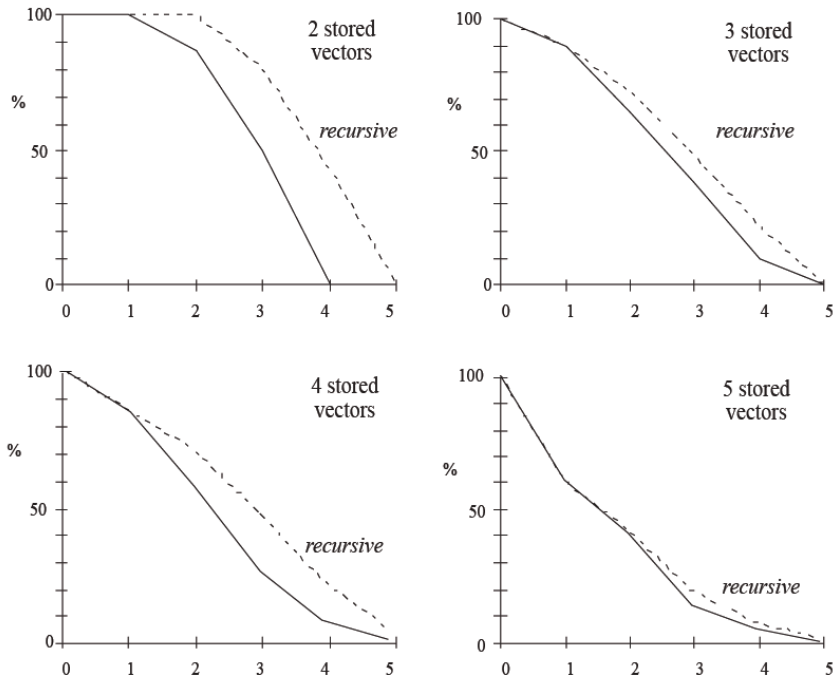


Fig. 12.5. The grey shading of each concentric circle represents the percentage of vectors with Hamming distance from 0 to 4 to stored patterns and which converge to them. The smallest circle represents the vectors with Hamming distance 0. Increasing the number of stored patterns from 1 to 8 reduces the size of the basins of attraction.

Basins of attraction: few patterns ($n = 10$)



- all patterns stable
- recognition rate vs. Hamming distance of input patterns

Basins of attraction: saturated network ($n = 10$)

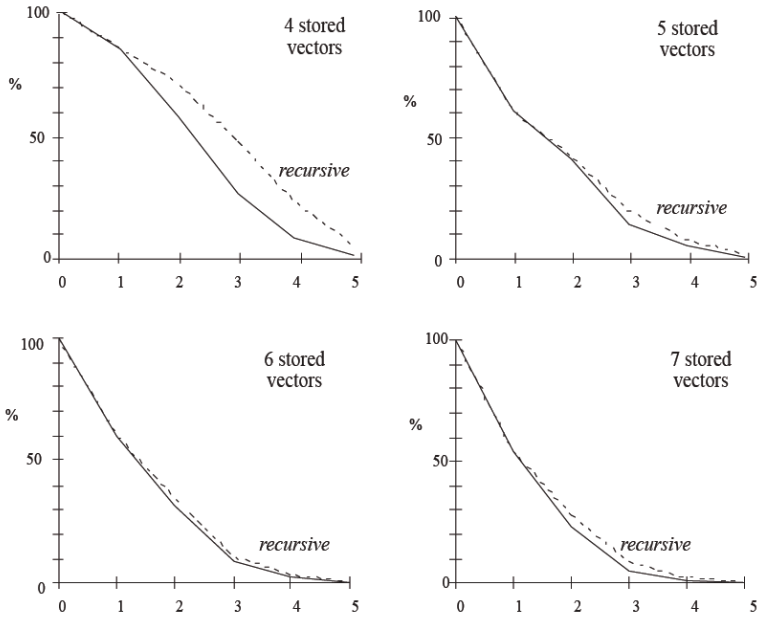


Fig. 12.6. Comparison of the percentage of vectors with a Hamming distance from 0 to 5 from stored patterns and which converge to them

- all patterns still stable
- but basins of attraction smaller

Basins of attraction ($n = 100$)

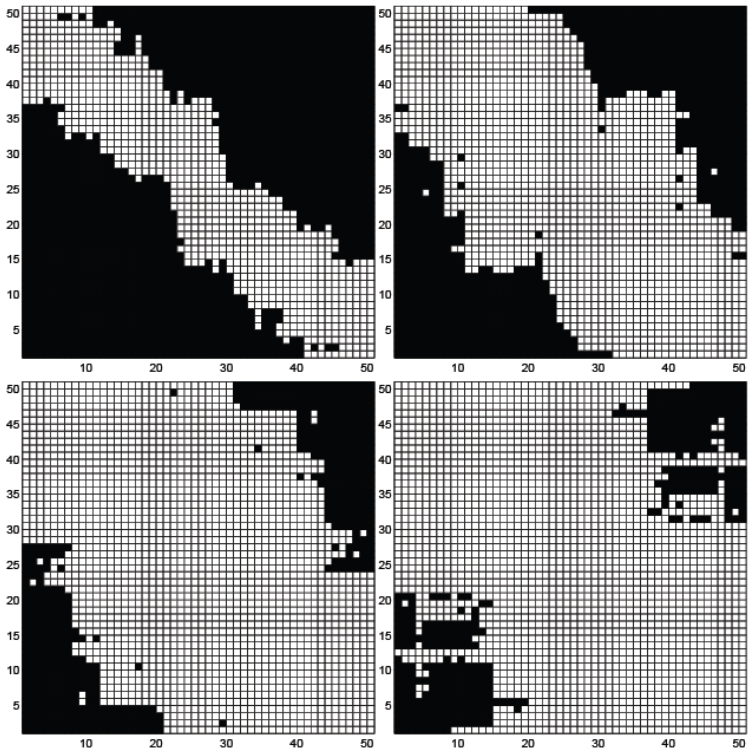
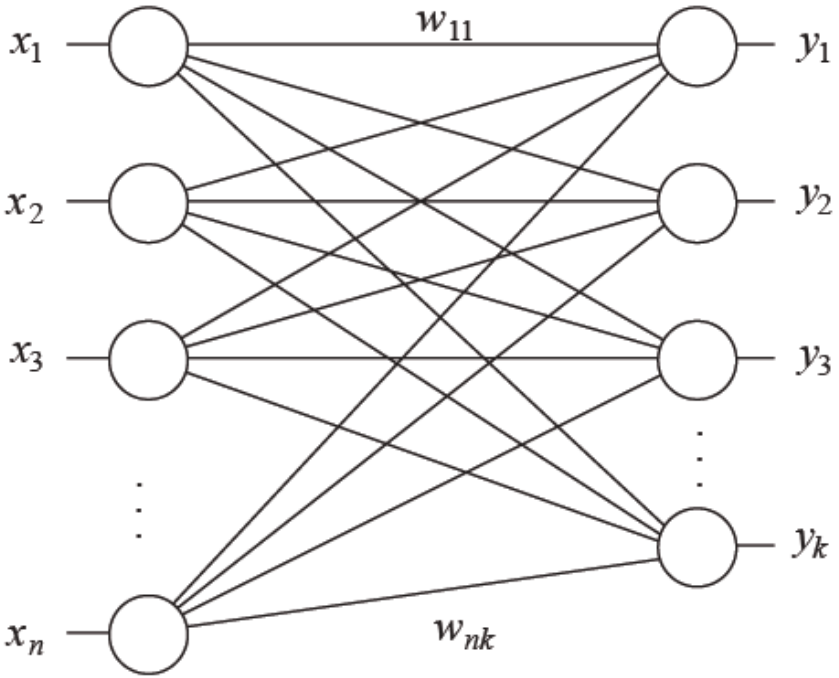


Fig. 12.8. Basin of attraction in 100-dimensional space of a stored vector. The number of stored patterns is 4, 6, 10 and 15, from top to bottom, left to right.

BAM: network



- hetero-associative: $x \mapsto y \mapsto x' \mapsto y' \mapsto x'' \mapsto y'' \ldots$

BAM: energy function

With the BAM we can motivate and explore the concept of an *energy function* in a simple setting. Assume that a BAM is given for which the vector pair $(\mathbf{x}, \mathbf{y})$ is a stable state. If the initial vector presented to the network from the left is $\mathbf{x}_0$, the network will converge to $(\mathbf{x}, \mathbf{y})$ after some iterations. The vector $\mathbf{y}_0$ is computed according to $\mathbf{y}_0 = \text{sgn}(\mathbf{x}_0\mathbf{W})$. If $\mathbf{y}_0$ is now used for a new iteration from the right, excitation of the units in the left layer can be summarized in an excitation vector $\mathbf{e}$ computed according to

$$\mathbf{e}^{\mathrm{T}} = \mathbf{W}\mathbf{y}_0.$$

The vector pair $(\mathbf{x}_0, \mathbf{y}_0)$ is a stable state of the network if $\text{sgn}(\mathbf{e}) = \mathbf{x}_0$. All vectors $\mathbf{e}$ close enough to $\mathbf{x}_0$ fulfill this condition. These vectors differ from $\mathbf{x}_0$ by a small angle and therefore the product $\mathbf{x}_0\mathbf{e}^{\mathrm{T}}$ is larger than for other vectors of the same length but further away from $\mathbf{x}_0$. The product

$$E = -\mathbf{x}_0\mathbf{e}^{\mathrm{T}} = -\mathbf{x}_0\mathbf{W}\mathbf{y}_0^{\mathrm{T}}$$

is therefore smaller (because of the minus sign) if the vector $\mathbf{W}\mathbf{y}_0^{\mathrm{T}}$ lies closer to $\mathbf{x}_0$. The scalar value E can be used as a kind of index of convergence to the stable states of an associative memory. We call E the *energy function* of the network.

BAM: energy function (with neuron thresholds θ)

This can be done by extending the input vectors with an additional constant component. Each n-dimensional vector $\mathbf{x}$ will be transformed into the vector $(x_1, \ldots, x_n, 1)$. We proceed in a similar way with the k-dimensional vector $\mathbf{y}$. The weight matrix $\mathbf{W}$ must be extended to a new matrix $\mathbf{W}'$ with an additional row and column. The negative thresholds of the units in the right layer of the BAM are included in row $n+1$ of $\mathbf{W}'$, whereas the negative thresholds of the units in the left are used as the entries of the column $k+1$ of the weight matrix. The entry $(n+1, k+1)$ of the weight matrix can be set to zero. This transformation is equivalent to the introduction of an additional unit with constant output 1 into each layer. The weight of each edge from a constant unit to each one of the others is the negative threshold of the connected unit. It is straightforward to deduce that the energy function of the extended network can be written as

$$E(\mathbf{x}_i, \mathbf{y}_i) = -\frac{1}{2}\mathbf{x}_i \mathbf{W} \mathbf{y}_i^{\mathrm{T}} + \frac{1}{2}\theta_r \mathbf{y}_i^{\mathrm{T}} + \frac{1}{2}\mathbf{x}_i \theta_\ell^{\mathrm{T}}. \qquad (13.5)$$

The row vector of thresholds of the k units in the left layer is denoted in the above expression by θ_ℓ. The row vector of thresholds of the n units in the right layer is denoted by θ_r.

BAM: stable states

Proposition 19. *A bidirectional associative memory with an arbitrary weight matrix* $\mathbf{W}$ *reaches a stable state in a finite number of iterations using either synchronous or asynchronous updates.*

Proof. For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, a vector $\mathbf{y} = (y_1, y_2, \ldots, y_k)$ and an $n \times k$ weight matrix $\mathbf{W} = \{w_{ij}\}$ the energy function is the bilinear form

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(x_1, x_2, \ldots, x_n) \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}.$$

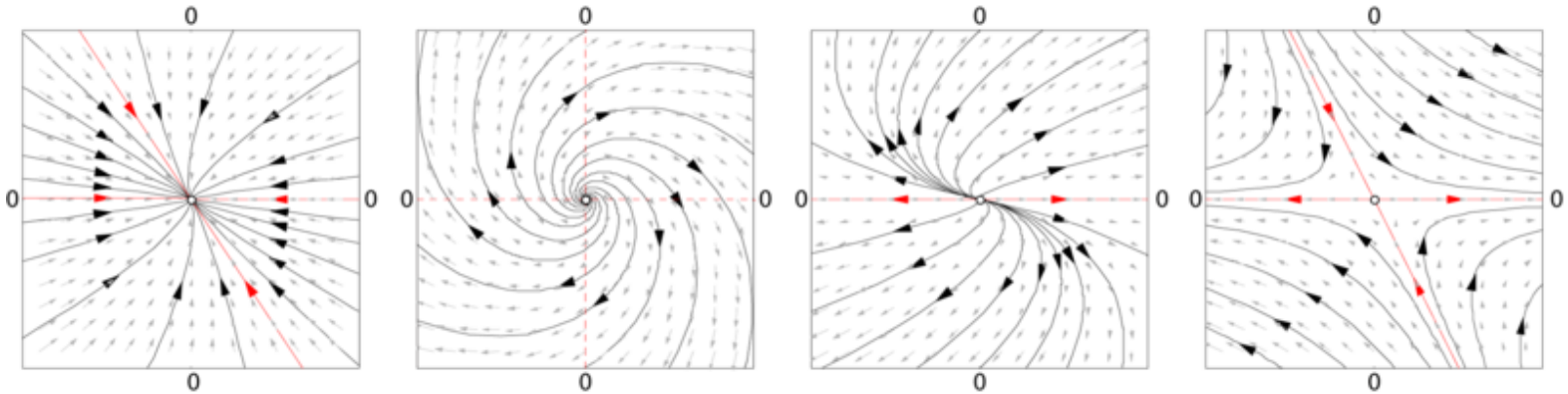
The value of $E(\mathbf{x}, \mathbf{y})$ can be computed by multiplying first $\mathbf{W}$ by $\mathbf{y}^{\mathrm{T}}$ and the result with $-\mathbf{x}/2$. The product of the i-th row of $\mathbf{W}$ and $\mathbf{y}^{\mathrm{T}}$ represents the excitation of the i-th unit in the left layer. If we denote these excitations by $g_1, g_2, \ldots, g_n$ the above expression transforms to

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(x_1, x_2, \ldots, x_n) \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}.$$

Digression: dynamic systems and chaos

- dynamic systems: differential equations, $\dot{x} = \Phi(x)$
- dynamic maps: discrete time, $x(t+1) = \Phi(x)$
- study the time evolution of the systems

- but non-linear systems typically show chaotic behaviour



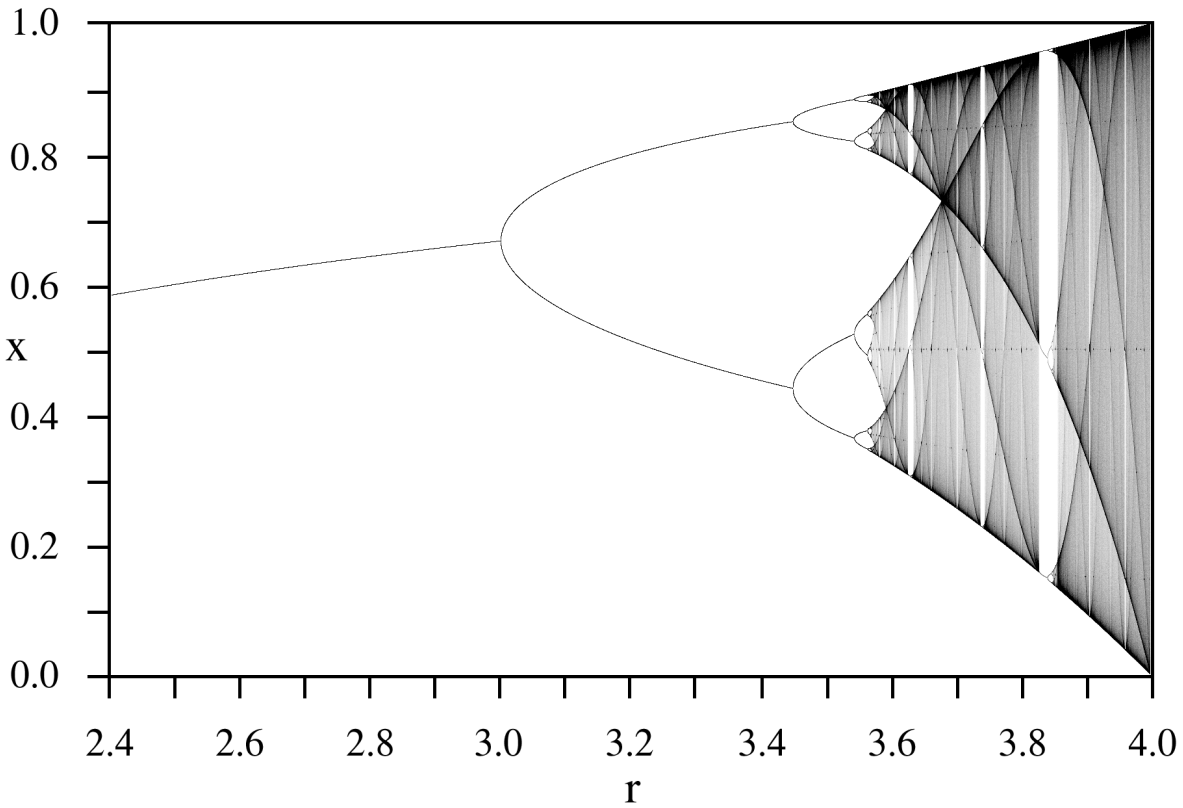
Example: logistic map

- Logistic map: $x_{n+1} = r\,x_n(1 - x_n)$
- Models population growth, population growth parameter r
- $x \in [0, 1]$

- $0 < r < 1$: population eventually dies
- $1 < r < 2$: population stabilizes at $(r - 1)/r$
- $2 < r < 3$: population oscillates, finally reaches $(r - 1)/r$
- $3 < r < 1 + \sqrt{6}$: population osciallates between two values
- $3.45 < r < 3.54$: oscillations between four values
- $3.57 < r < 4$: chaos
- $4 < r$: population diverges
- typical behaviour for non-linear systems with feedback

Wikipedia, Logistic_Map

Logistic map: bifurcation diagram



Chaotic brain?!

- chaos typical even in the 1-dimensional case
- gets a lot worse in high-dimensional systems

- neurons are highly non-linear
- brain has 10^{11} neurons
- with lots of feedback connections

- how to explain stable behaviour at all?
- how to explain memory?

Asynchronous dynamics: energy decreases

In asynchronous networks at each time t we randomly select a unit from the left or right layer. The excitation is computed and its sign is the new activation of the unit. If the previous activation of the unit remains the same after this operation, then the energy of the network has not changed.

The state of unit i on the left layer will change only when the excitation g_i has a different sign than x_i, the present state. The state is updated from x_i to x_i', where x_i' now has the same sign as g_i. Since the other units do not change their state, the difference between the previous energy $E(\mathbf{x}, \mathbf{y})$ and the new energy $E(\mathbf{x}', \mathbf{y})$ is

$$E(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}', \mathbf{y}) = -\frac{1}{2} g_i (x_i - x_i').$$

Since both x_i and $-x_i$ have a different sign than g_i it follows that

$$E(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}', \mathbf{y}) > 0.$$

The new state $(\mathbf{x}', \mathbf{y})$ has a lower energy than the original state $(\mathbf{x}, \mathbf{y})$. The same argument can be made if a unit on the right layer has been selected, so that for the new state $(\mathbf{x}, \mathbf{y}')$ it holds that

$$E(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}, \mathbf{y}') > 0,$$

Energy: equivalent forms

The value of $E(\mathbf{x}, \mathbf{y})$ can be computed by multiplying first $\mathbf{W}$ by $\mathbf{y}^{\mathrm{T}}$ and the result with $-\mathbf{x}/2$. The product of the i-th row of $\mathbf{W}$ and $\mathbf{y}^{\mathrm{T}}$ represents the excitation of the i-th unit in the left layer. If we denote these excitations by $g_1, g_2, \ldots, g_n$ the above expression transforms to

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(x_1, x_2, \ldots, x_n) \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}.$$

We can also compute $E(\mathbf{x}, \mathbf{y})$ multiplying first $\mathbf{x}$ by $\mathbf{W}$. The product of the i-th column of $\mathbf{W}$ with $\mathbf{x}$ corresponds to the excitation of unit i in the right layer. If we denote these excitations by $e_1, e_2, \ldots, e_k$, the expression for $E(\mathbf{x}, \mathbf{y})$ can be written as

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(e_1, e_2, \ldots, e_k) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}.$$

Therefore, the energy function can be written in the two equivalent forms

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2} \sum_{i=1}^{k} e_i y_i \quad \text{and} \quad E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2} \sum_{i=1}^{n} g_i x_i.$$

Energy: limit cycles with asymmetric couplings

A connection matrix with a zero diagonal can also lead to oscillations in the case where the weight matrix is not symmetric. The weight matrix

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

describes the network of Figure 13.3. It transforms the state vector $(1, -1)$ into the state vector $(1, 1)$ when the network is running asynchronously. After this transition the state $(-1, 1)$ can be updated to $(-1, -1)$ and finally to $(1, -1)$. The state vector changes cyclically and does not converge to a stable state.

Energy: non-zero diagonal elements

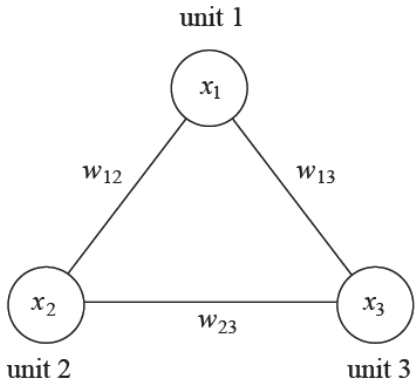


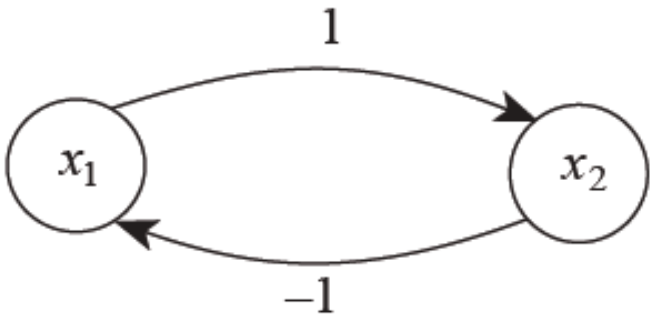
Fig. 13.2. A Hopfield network of three units

It is easy to show that if the weight matrix does not contain a zero diagonal, the network dynamics does not necessarily lead to stable states. The weight matrix

$$\mathbf{W} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix},$$

for example, transforms the state vector $(1, 1, 1)$ into the state vector $(-1, -1, -1)$ and conversely. In the case of asynchronous updating, the network chooses randomly among the eight possible network states.

Energy: asymmetric couplings



- network oscillates

- Note: we can try to use asymmetric couplings to store state sequences in the network
- Note: possible, but doesn't work very well

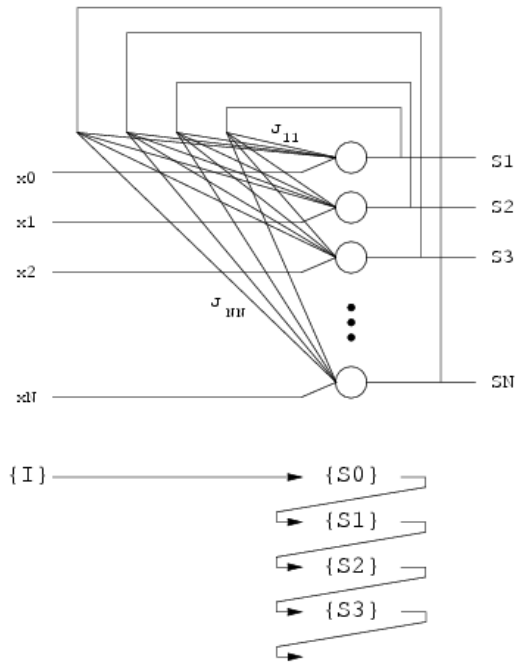
Hopfield model

- fully-connected network
- binary neurons, $s_i = \{-1, +1\}$
- network initialized with external input state $s(0)$
- asynchronous network update
- $s_i(t+1) = \mathrm{sgn}(\sum_j w_{ij} s_j(t))$

- symmetric couplings, zero diagonal ($w_{ii} = 0$)
- allows to define an energy function
- stable attractors
- use Hebb learning to store a set of patterns x^μ

(Hopfield 1982)

Feedback network: architecture



- network initialized via separate input layer
- network dynamics via feedback connections

Hopfield model: symmetric couplings

The symmetry of the weight matrix and a zero diagonal are thus *necessary conditions* for the convergence of an asynchronous totally connected network to a stable state. These conditions are also sufficient, as we show later.

The units of a Hopfield network can be assigned a threshold θ different from zero. In this case each unit selected for a state update adopts the state 1 if its total excitation is greater than θ, otherwise the state -1. This is the activation rule for perceptrons, so that we can think of Hopfield networks as asynchronous recurrent networks of perceptrons.

The energy function of a Hopfield network composed of units with thresholds different from zero can be defined in a similar way as for the BAM. In this case the vector $\mathbf{y}$ of equation (13.5) is $\mathbf{x}$ and we let $\theta = \theta_\ell = \theta_r$.

Hopfield model: energy function

Definition 17. *Let* $\mathbf{W}$ *denote the weight matrix of a Hopfield network of* n *units and let* θ *be the* n*-dimensional row vector of units' thresholds. The energy* $E(\mathbf{x})$ *of a state* $\mathbf{x}$ *of the network is given by*

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}\mathbf{W}\mathbf{x}^{\mathrm{T}} + \theta\mathbf{x}^{\mathrm{T}}.$$

The energy function can also be written in the form

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i x_i.$$

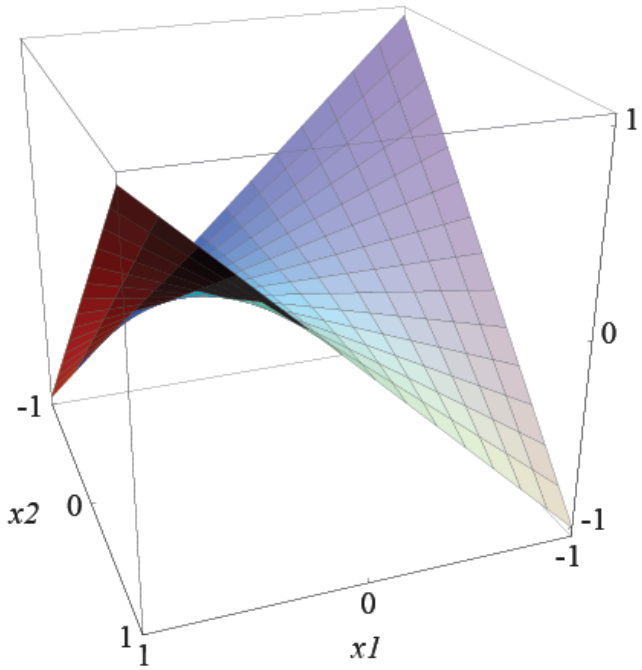
Hopfield model: flipflop network

The energy function of a Hopfield network is a quadratic form. A Hopfield network always finds a local minimum of the energy function. It is thus interesting to look at an example of the shape of such an energy function. Figure 13.4 shows a network of just two units with threshold zero. It is obvious that the only stable states are $(1, -1)$ and $(-1, 1)$. In any other state, one of the units forces the other to change its state to stabilize the network. Such a network is a flip-flop, a logic component with two outputs which assume complementary logic values.

The energy function of a flip-flop with weights $w_{12} = w_{21} = -1$ and two units with threshold zero is given by

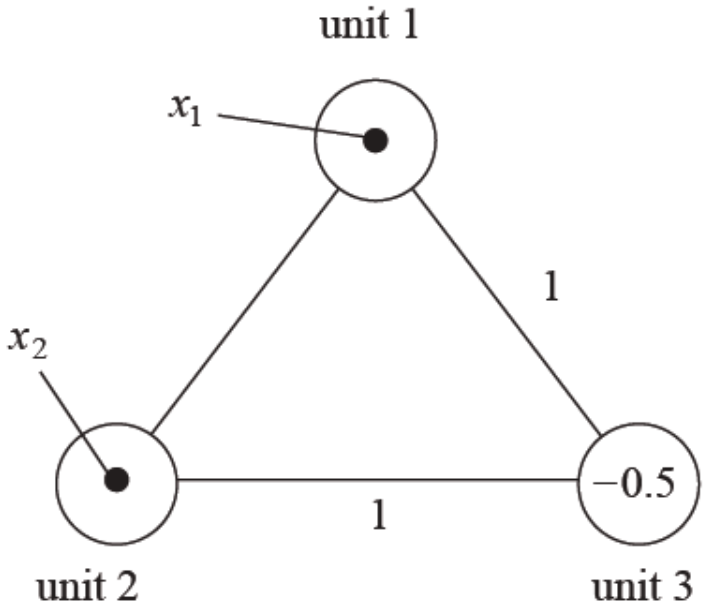
$$E(x_1, x_2) = x_1 x_2,$$

Hopfield model: flipflop network energy function



- one of two stable states (lowest energy)
- assumes "smooth" activation functions (more about this later)

Hopfield model: OR function



- computations with the Hopfield network?
- example: OR function

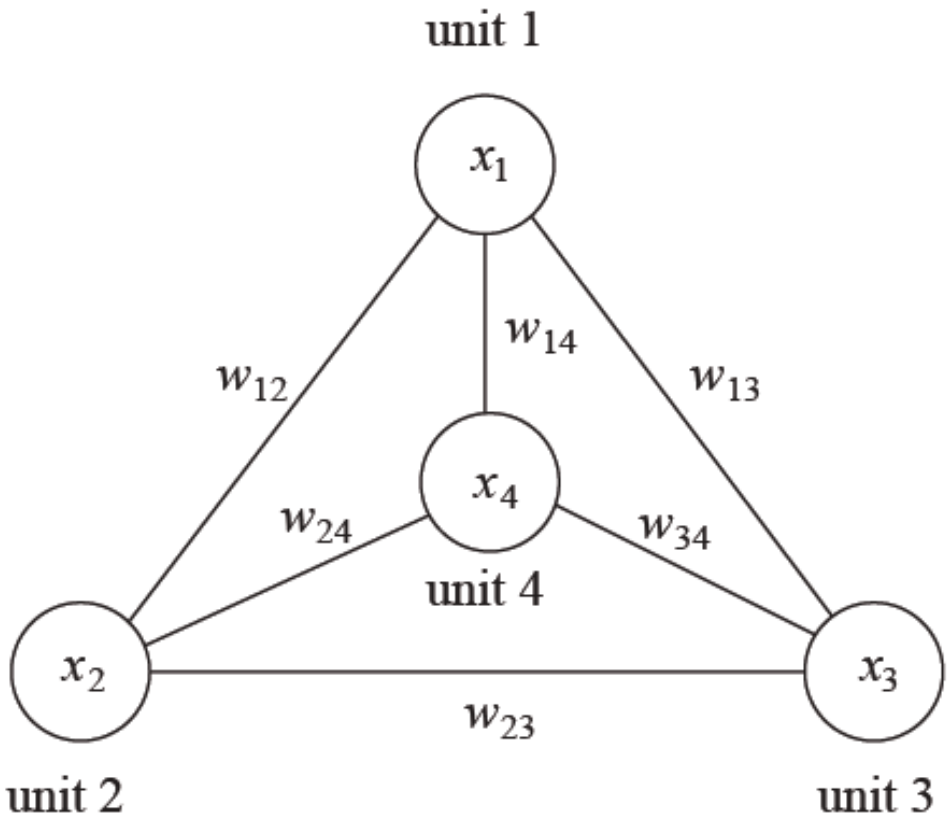
Hopfield model: XOR function tables

unit	1	2	3
state 1	−1	−1	−1
state 2	1	−1	1
state 3	−1	1	1
state 4	1	1	−1

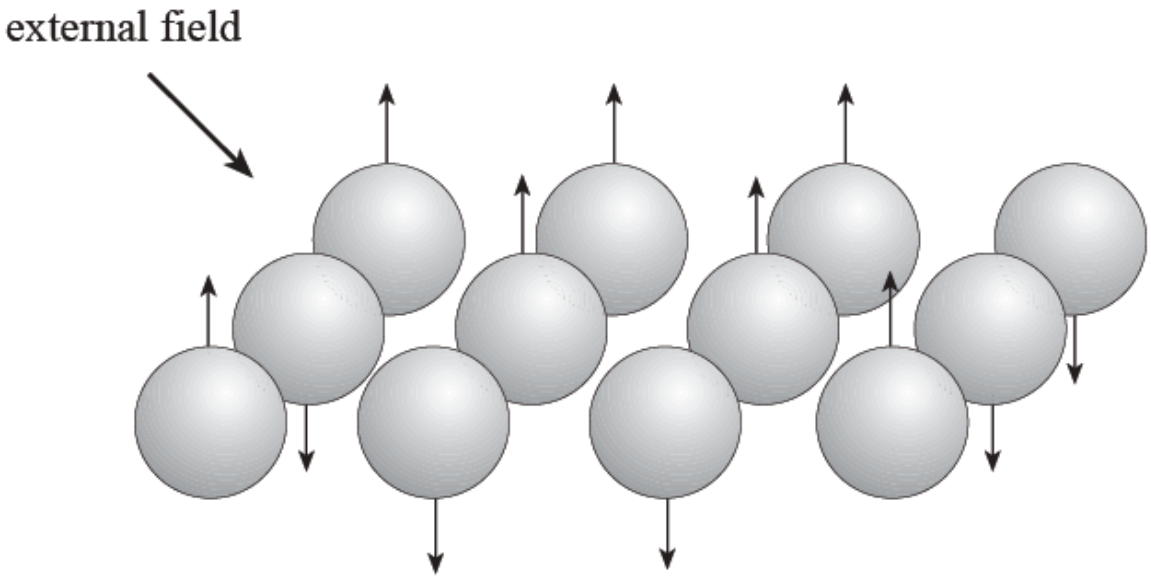
From the point of view of the third unit (third column) this is the XOR function. If the four vectors shown above are to become stable states of the network, the third unit cannot change state when any of these four vectors has been loaded in the network. In this case the third unit should be capable of linearly separating the vectors $(-1,-1)$ and $(1,1)$ from the vectors $(-1,1)$ and $(1,-1)$, which we know is impossible. The same argument is valid for any of the three units, since the table given above remains unchanged after a permutation of the units' labels. This shows that no Hopfield network of three units can have these stable states. However, the XOR problem can be solved if the network is extended to four units. The network of Figure 13.7 can assume the following stable states, if adequate weights and thresholds are selected:

unit	1	2	3	4
state 1	−1	−1	−1	1
state 2	1	−1	1	1
state 3	−1	1	1	1
state 4	1	1	−1	−1

Hopfield model: XOR network diagram



Hopfield model: Ising spin model



- neuron model is equivalent to *Ising* model of magnetism
- magnetic field $h_i = \sum_{j=1}^{n} w_{ij} x_j + h^*$
- can reuse techniques from theoretical phyiscs for network analysis
- possible to calculate the storage capacity of the network

Hopfield model: Hebbian learning

A Hopfield network can be used as an associative memory. If we want to "imprint" m different stable states in the network we have to find adequate weights for the connections. In the case of the BAM we already mentioned that Hebbian learning is a possible alternative. Since Hopfield networks are a specialization of BAM networks, we also expect Hebbian learning to be applicable in this case. Let us first discuss the case of a Hopfield network with n units and threshold zero.

Hebbian learning is implemented by loading the m selected n-dimensional stable states $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ on the network and by updating the network's weights (initially set to zero) after each presentation according to the rule

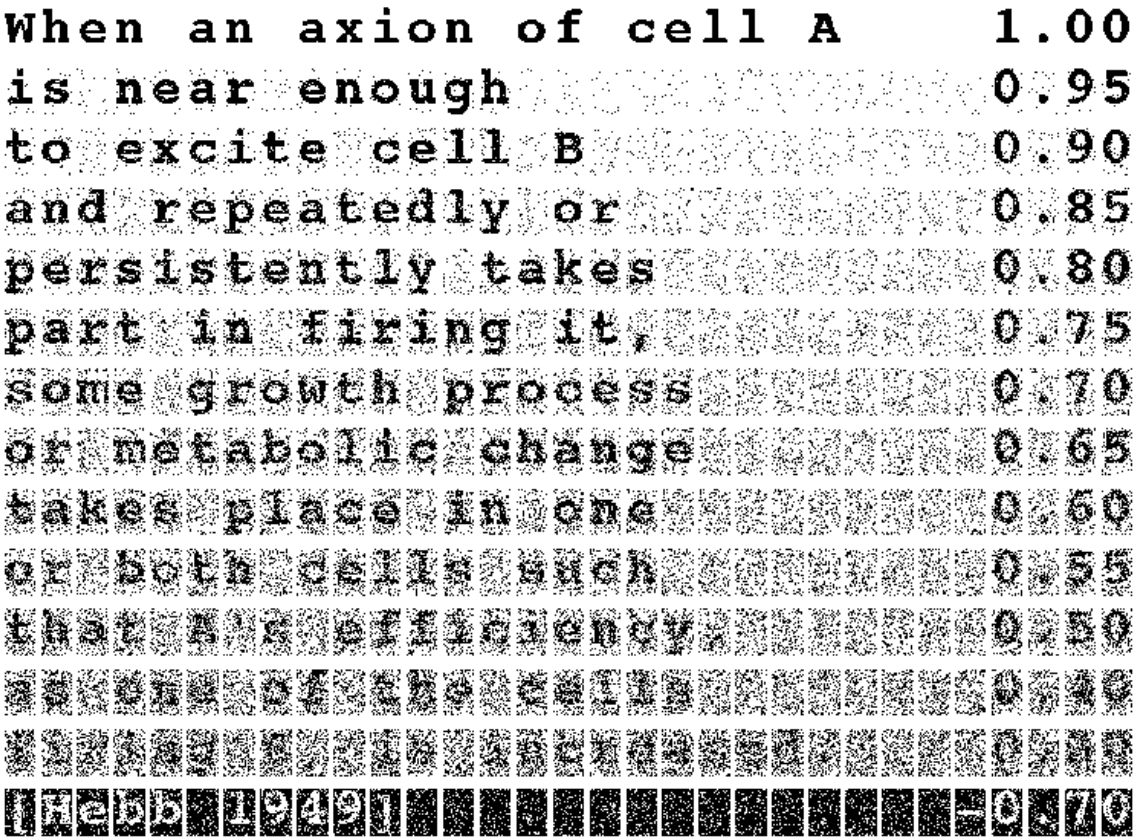
$$w_{ij} \leftarrow w_{ij} + x_i^k x_j^k, \qquad i, j = 1, \ldots, n \quad \text{and} \quad i \neq j.$$

The symbols x_i^k and x_j^k denote the i-th and j-th component respectively of the vector $\mathbf{x}_k$. The only difference from an autoassociative memory is the

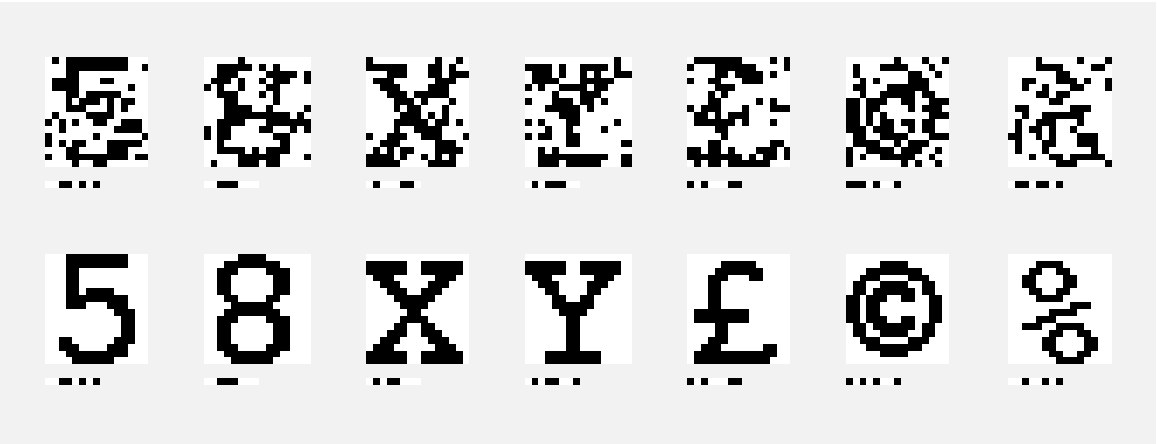
Hopfield model: Minover learning

Minover_PSG_learning(network $\mathcal{N}$, patterns ξ^μ, desired stabilities $\{\Lambda_{i\mu}\}$):
begin
    initialize the couplings J_{ij} to random or Hebb values
    for all neurons S_i do (parallel)
        repeat
            calculate all pattern stabilities $\kappa_{i\mu}$
            select the pattern ξ^ν with lowest stability $\kappa_{i\nu}$
            $J_{ij} \rightarrow J_{ij} + N^{-1} \cdot \Theta(\Lambda_{i\nu} - \kappa_{i\nu}) \cdot \xi_i^\nu \xi_j^\nu$
        until (all $\kappa_{i\mu} > \Lambda_{i\mu}$)
    end for
    calculate new norm $||J_{ij}||$
    return J_{ij}
end

Hopfield model: letter recognition



Hopfield model: letter recognition



- example input patterns and stable attractors
- append additional *index bits* to patterns

Hopfield model: Perceptron equivalence

The interesting result which can immediately be inferred from the equivalence of Hopfield networks and perceptrons is that every learning algorithm for perceptrons can be transformed into a learning method for Hopfield networks. The delta rule or algorithms that proceed by finding inner points of solution polytopes can also be used to train Hopfield networks.

We have already shown in Chap. 10 that learning problems for multilayer networks are in general *NP*-complete. However, some special architectures can be trained in polynomial time. We saw in Chap. 4 that the learning problem for Hopfield networks can be solved in polynomial time, because there are learning algorithms for perceptrons whose complexity grows polynomially with the number of training vectors and their dimension (for example, Karmarkar's algorithm). Since the transformation described in the previous section converts m desired stable states into nm vectors to be linearly separated, and since this can be done in polynomial time, it follows that the learning problem for Hopfield networks can be solved in polynomial time. In Chap. 6 we also showed how to compute an upper bound for the number of linearly separable functions. This upper bound, valid for perceptrons, is also valid for Hopfield networks, since the stable states must be linearly separable (for the equivalent perceptron). This equivalence simplifies computation of the capacity of a Hopfield network when it is used as an associative memory.

Hopfield model: applications?

- model works well for associative memory
- other applications?

- in particular, parallel fast computation of difficult problems?
- interest triggered by paper on traveling-salesman problem

(Hopfield 1984)

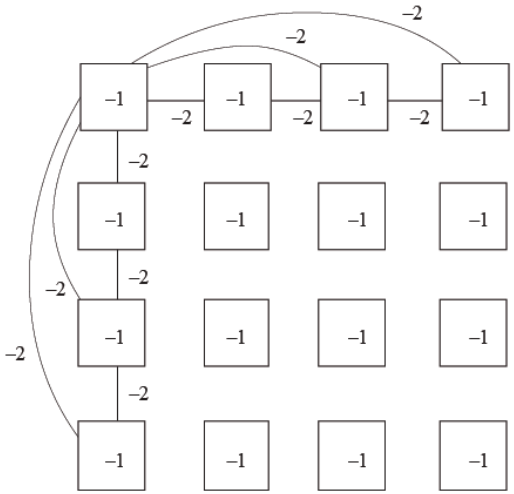
Hopfield model: eight rooks problem

- Put *n* rooks on an *nxn* chessboard
- so that they cannot take each other

- solve with Hopfield network, *nxn* neurons
- consider sorted into rows and columns
- active neuron suppresses other neurons in same row/column
- network converges to solution

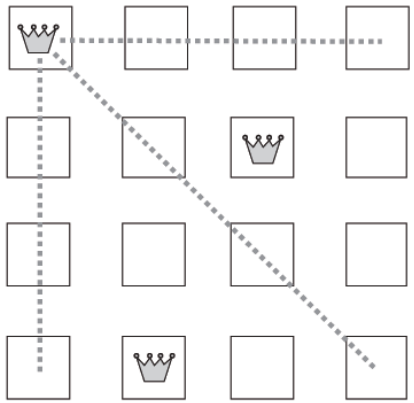
Hopfield model: eight rooks problem

The network of Figure 13.16 can solve this problem for a 4×4 board. Each field is represented by a unit. Only the connections of the first unit in the board are shown to avoid cluttering the diagram. The connections of each unit to all elements in the same row or column have the weight -2, all others have a weight zero. All units have the threshold -1. Any unit set to 1 inhibits any other units in the same row or column. If a row or column is all set to 0, when one of its elements is selected it will immediately switch its state to 1, since the total excitation (zero) is greater than the threshold -1.



Hopfield model: eight queens problem

The well-known eight queens problem can also be solved with a Hopfield network. It is just a generalization of the rooks problem, since now the diagonals of the board will also be considered. Each diagonal can be occupied at most once by a queen. As before with the rooks problem, we solve this task by overlapping multiflop problems at each square. Figure 13.17 shows how three multiflop chains have to be considered for each field. The diagram shows a 4×4 board and the overlapping of multiflop problems for the upper left square on the board. This overlapping provides us with the necessary weights, which are set to $w_{ij} = -2$, when unit i is different from unit j and belongs to the same row, column or diagonal as unit j. Otherwise we set w_{ij} to zero. The thresholds of all units are set to -1.



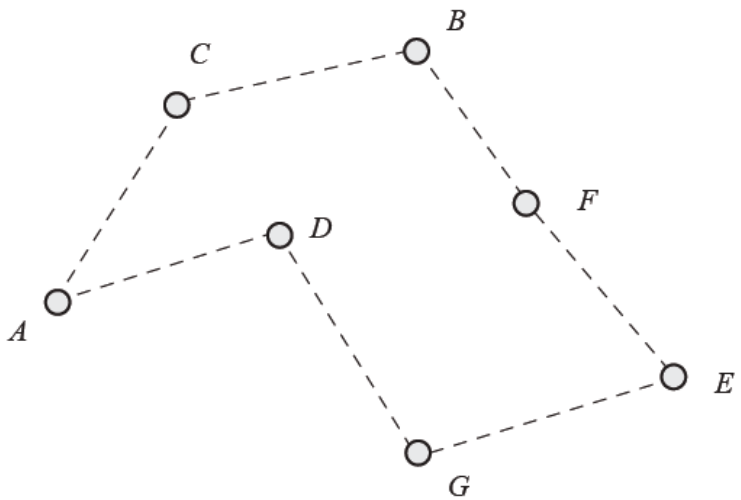
Hopfield model: eight queens problem

- $w_{ij} = -2$ for neurons on same row, column, diagonal
- neuron threshold -1

- problem: this doesn't work always
- energy function has local minima with less than n queens

Hopfield model: traveling salesman problem

The Traveling Salesman Problem (TSP) is one of the most celebrated benchmarks in combinatorics. It is simple to state and visualize and it is *NP*-complete. If we can solve the TSP efficiently, we can provide an efficient solution for other problems in the class *NP*. Hopfield and Tank [200] were the first to try to solve the TSP using a neural network.



- use continuous neuron activation functions (e.g. sigmoid)

Hopfield model: solving the TSP

Solving the TSP requires minimizing the length of the round trip, that is of

$$L = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{ik} x_{j,k+1},$$

where x_{ik} represents the state of the unit corresponding to the entry ik in the matrix. When x_{ik} and $x_{j,k+1}$ are both 1, this means that the city S_i is visited in the k-th step and the city S_j in the step $k+1$. The distance between both cities must be added to the total length of the round trip. We use the convention that the column $n+1$ is equal to the first column of the matrix, so that we always obtain a round trip.

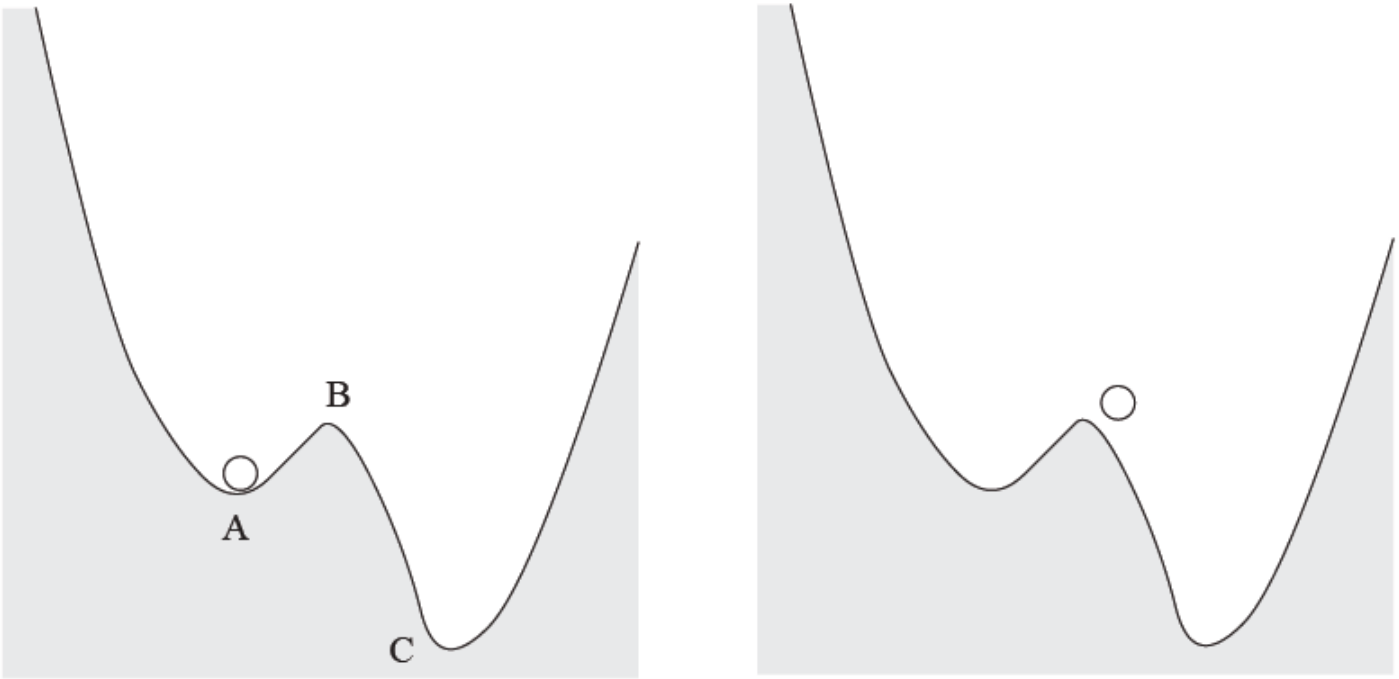
In the minimization problems we must include the constraints for a valid round trip. It is necessary to add the energy function of the rooks problem to the length L to obtain the new energy function E, which is given by

$$E = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{ik} x_{j,k+1} + \frac{\gamma}{2} \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{n} x_{ij} - 1 \right)^2 + \sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2 \right),$$

Hopfield model: storage capacity

The properties of Hopfield networks have been investigated since 1982 using the theoretical tools of statistical mechanics [322]. Gardner [155] published a classical treatise on the capacity of the perceptron and its relation to the Hopfield model. The total field sensed by particles with a spin can be computed using the methods of mean field theory. This simplifies a computation which is hopeless without the help of some statistical assumptions [189]. Using these methods Amit et al. [24] showed that the number of stable states in a Hopfield network of n units is bounded by $0.14n$. A *recall* error is tolerated only 5% of the time. This upper bound is one of the most cited numbers in the theory of Hopfield networks.

Hopfield model: stochastic noise



- noise can improve network convergence
- because network can escape from local minima