Artificial Intelligence

Neural Networks

Lesson 14: Hopfield Networks

Vincenzo Piuri

Università degli Studi di Milano

Contents

- Hopfield networks
- Examples of hopfield networks
- State graph
- Convergence
- Associative memory
- Solving optimization problems
- Simulated annealing
- Simulated annealing in Hopfield networks

Hopfield Networks (1)

- A Hopfield network is a neural network with a graph $G = (U, C)$
  - $U_{hidden} = \emptyset, U_{in} = U_{out} = U$
  - $C = U \, x \, U \; - \{(u, u) \mid u \in U\}$
- Features
  - All neurons are input as well as output neurons
  - There are no hidden neurons
  - Each neuron receives input from all other neurons
  - A neuron is not connected to itself
  - The connection weights are symmetric

Hopfield Networks (2)

- The network input function of each neuron is the weighted sum of the outputs of all other neurons

$$f_{net}^{(u)}(\overrightarrow{w_u}, \overrightarrow{in_u}) = \overrightarrow{w_u^T}\overrightarrow{in_u} = \sum_{v \in U-\{u\}} w_{uv} out_v$$

- The activation function of each neuron is a threshold function

$$\forall u \in U: \quad f_{\mathrm{act}}^{(u)}(\mathrm{net}_u, \theta_u) = \begin{cases} 1, & \text{if} \quad \mathrm{net}_u \geq \theta, \\ -1, & \text{otherwise.} \end{cases}$$

- The output function of each neuron is the identity

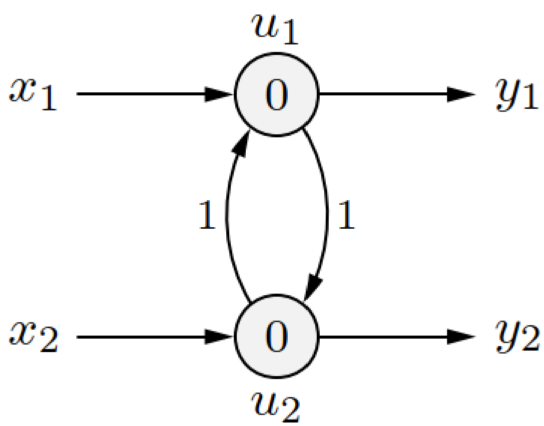
$$\forall u \in U: \quad f_{\mathrm{out}}^{(u)}(\mathrm{act}_u) = \mathrm{act}_u \,.$$

Hopfield Networks (3)

- General weight matrix of a Hopfield network

$$\mathbf{W} = \begin{pmatrix} 0 & w_{u_1u_2} & \dots & w_{u_1u_n} \\ w_{u_1u_2} & 0 & \dots & w_{u_2u_n} \\ \vdots & \vdots & & \vdots \\ w_{u_1u_n} & w_{u_1u_n} & \dots & 0 \end{pmatrix}$$

Examples of Hopfield Networks (1)



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

- The behavior of a Hopfield network can depend on the update order
  - Computations can oscillate if neurons are updated in parallel
  - Computations always converge if neurons are updated sequentially

Examples of Hopfield Networks (2)

- Synchronous (Parallel) update
  - The computations oscillate, no stable state is reached
  - Output depends on when the computations are terminated

	u_1	u_2
input phase	-1	1
work phase	1	-1
	-1	1
	1	-1
	-1	1
	1	-1
	-1	1

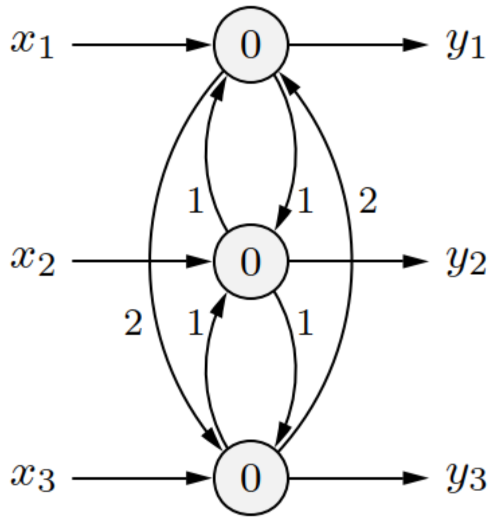
Examples of Hopfield Networks (3)

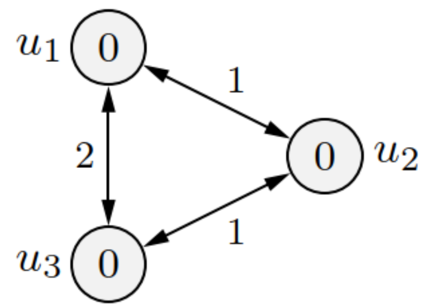
	u_1	u_2
input phase	$\mathbf{-1}$	$\mathbf{1}$
work phase	$\mathbf{1}$	1
	1	$\mathbf{1}$
	$\mathbf{1}$	1
	1	$\mathbf{1}$

	u_1	u_2
input phase	$\mathbf{-1}$	$\mathbf{1}$
work phase	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1
	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1

- Asynchronous (Sequential) update
  - Regardless of the update order a stable state is reached
  - However, which state is reached depends on the update order

Examples of Hopfield Networks (4)



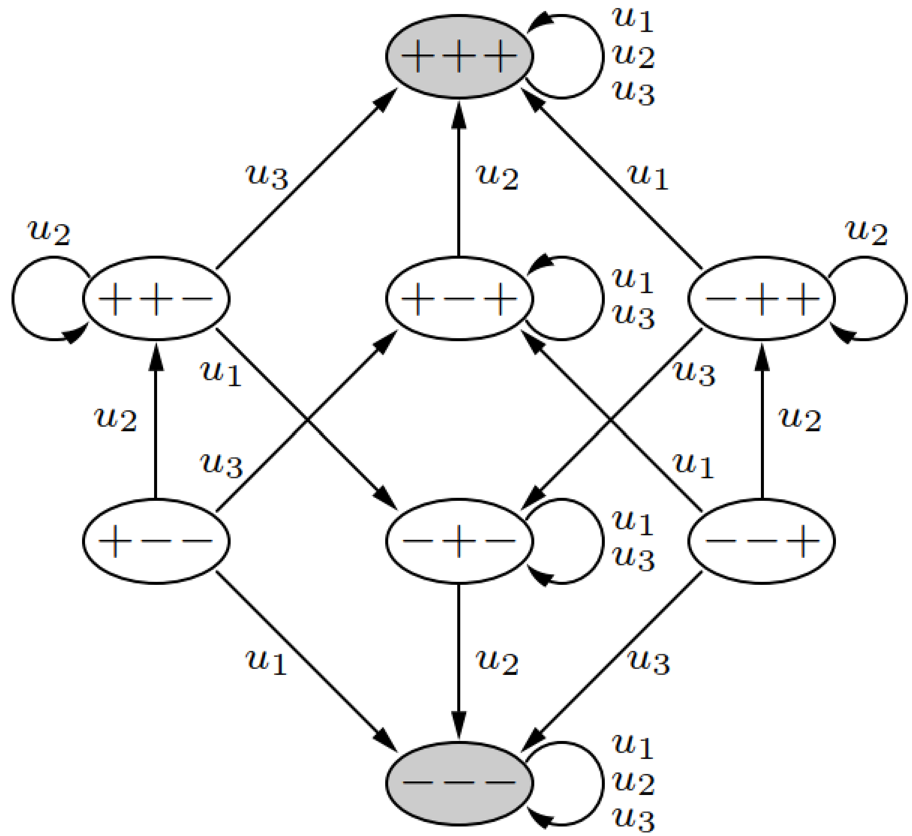


$$\mathbf{W} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

- Simplified representation of a Hopfield network
  - Symmetric connections between neurons are combined
  - Inputs and outputs are not explicitly represented

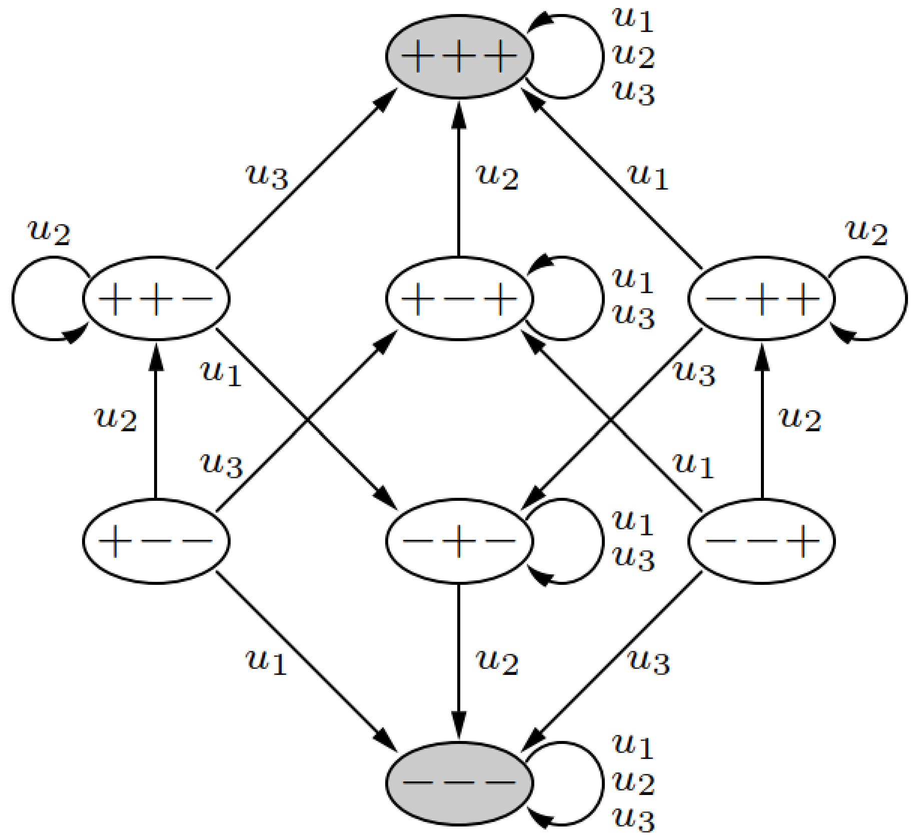
State Graph (1)

- Graph of activation states and transitions
  - "+"/"−" encode the neuron activations: "+" means +1 and "−" means −1



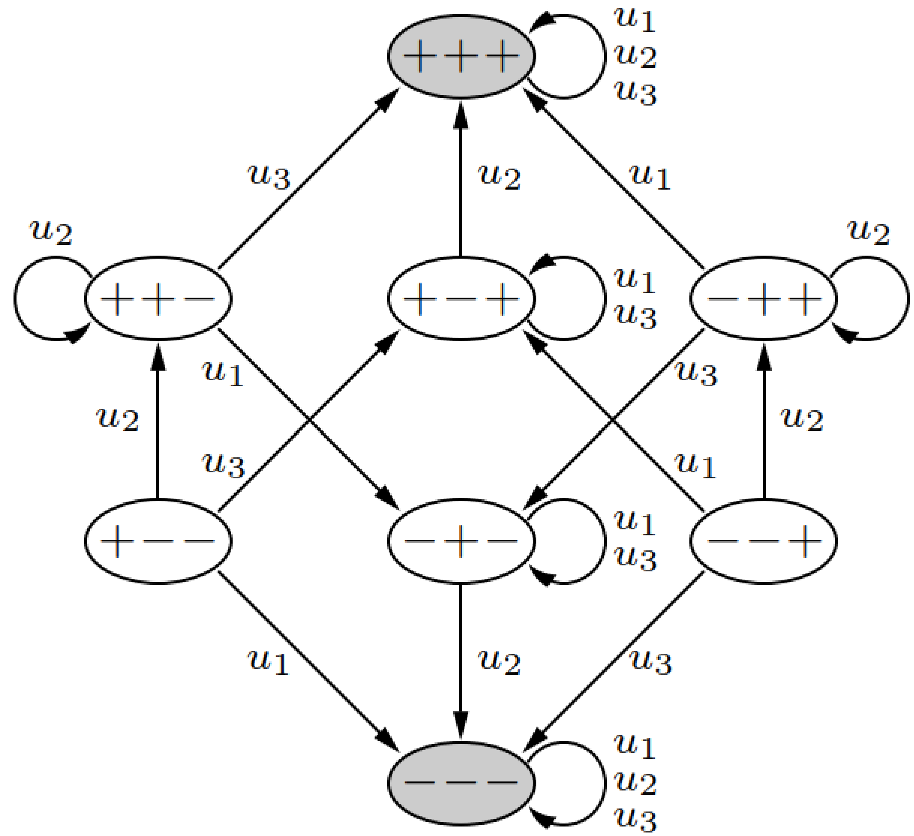
State Graph (2)

- Labels on arrows indicate the neurons, whose updates (activation changes) lead to the corresponding state transitions



State Graph (3)

- States shown in gray:
  - Stable states, cannot be left again
- States shown in white:
  - Unstable states, may be left again



Convergence (1)

- **Convergence Theorem**
  - If the activations of the neurons of a Hopfield network are updated sequentially (asynchronously), then a stable state is reached in a finite number of steps
  - If the neurons are traversed and updated cyclically in an arbitrary, but fixed order, at most $n \cdot 2^n$ steps (updates of individual neurons) are needed, where n is the number of neurons of the Hopfield network

Convergence (2)

- **Convergence Theorem**
  - Proof is carried out with the help of an energy function

$$
\begin{aligned}
E &= -\frac{1}{2}\,\vec{\mathrm{act}}^{\top}\mathbf{W}\vec{\mathrm{act}} + \vec{\theta}^{\top}\,\vec{\mathrm{act}} \\
&= -\frac{1}{2}\sum_{u,v\in U, u\neq v} w_{uv}\,\mathrm{act}_u\,\mathrm{act}_v + \sum_{u\in U}\theta_u\,\mathrm{act}_u\,.
\end{aligned}
$$

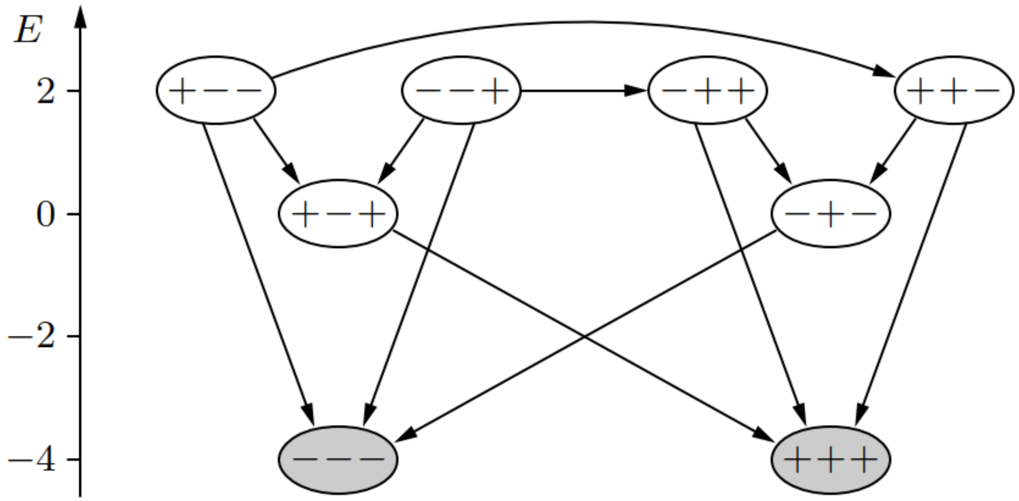
Convergence (3)

- It takes at most $n \cdot 2^n$ update steps to reach convergence
  - If the neurons are updated in an arbitrary, but fixed order, since this guarantees that the neurons are traversed cyclically, and therefore each neuron is updated every n steps

Convergence (3)

- If in a traversal of all n neurons no activation changes: **a stable state has been reached**
- If in a traversal of all n neurons at least one activation changes: **the previous state cannot be reached again**
  - Either the new state has a smaller energy than the old (no way back: updates cannot increase the network energy)
  - Or the number of +1 activations has increased (no way back: equal energy is possible only for $net_u \geq \theta_u$)

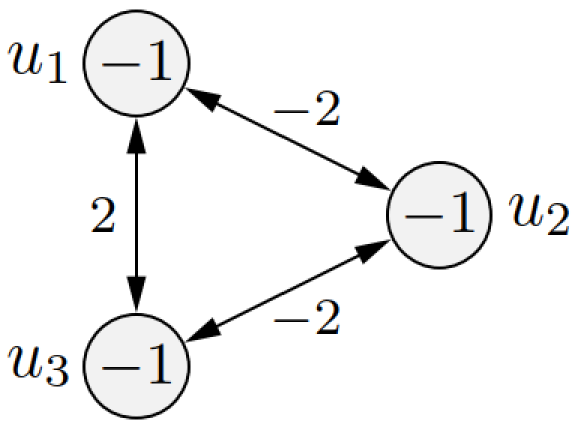
Convergence (4)



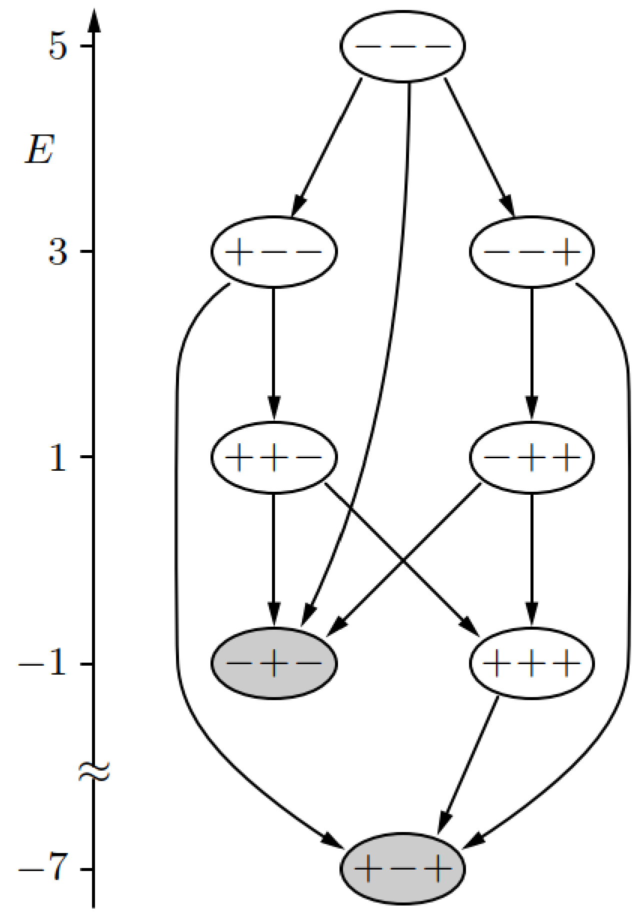
- Arrange states in state graph according to their energy
  - Energy function for example Hopfield network

$$E = -\operatorname{act}_{u_1}\operatorname{act}_{u_2} - 2\operatorname{act}_{u_1}\operatorname{act}_{u_3} - \operatorname{act}_{u_2}\operatorname{act}_{u_3}.$$

Convergence (5)



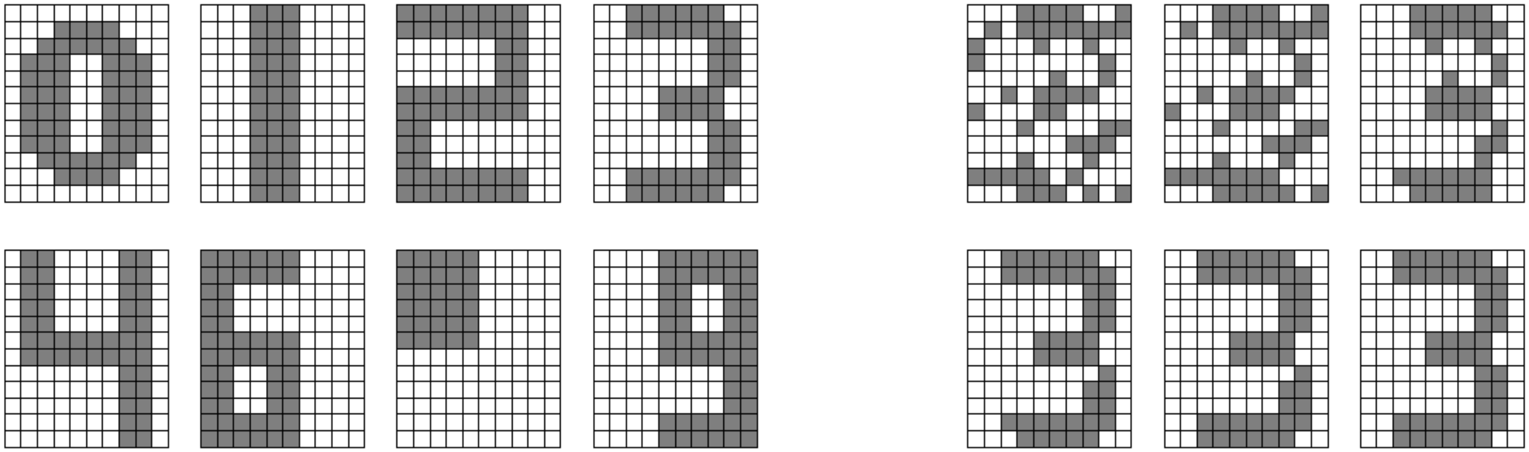
- The state graph need not be symmetric



Associative Memory (1)

- Use stable states to store patterns: **Hebbian learning rule**
  - First: Store only one pattern $\mathbf{p}$
    - Find weights so that pattern is a stable state
    - $\mathbf{W} = \mathbf{p}\mathbf{p}^{\mathrm{T}} - \mathbf{E}$
    - $w_{uv} = \begin{cases} 0 & \text{if } u = v \\ 1 & \text{if } u \neq v,\ act_u^{(p)} = act_v^{(p)} \\ -1 & \text{otherwise} \end{cases}$
  - Extension: storing several patterns
    - Compute $\mathbf{W}_i$ for each pattern $\mathbf{p}_i$
    - $\mathbf{W} = \sum_i \mathbf{W}_i$

Associative Memory (2)

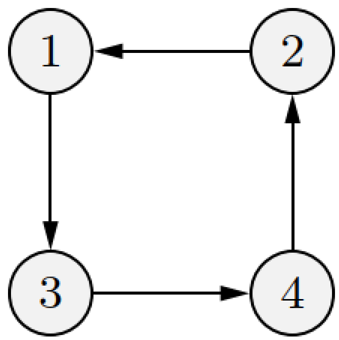


- Example: storing bit maps of numbers
  - Left: bit maps stored in a Hopfield network
  - Right: reconstruction of a pattern from a random input

Solving Optimization Problems (1)

- Use energy minimization to solve optimization problems
  - Transform function to optimize into a function to minimize
  - Transform function into the form of an energy function of a Hopfield network
  - Read the weights and threshold values from the energy function
  - Construct the corresponding Hopfield network
  - Initialize Hopfield network randomly and update until convergence
  - Read solution from the stable state reached
  - Repeat several times and use best solution found

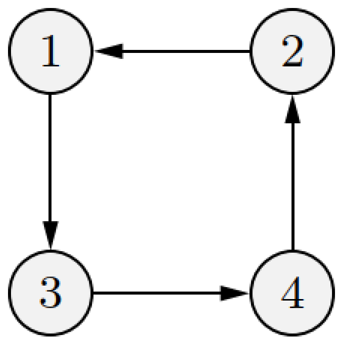
Solving Optimization Problems (2)



$$\begin{array}{c c l}
 & \text{city} & \\
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} & \\
 & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} & \begin{array}{l} 1. \\ 2. \\ 3. \\ 4. \end{array} \ \text{step}
\end{array}$$

- Traveling salesman problem
  - Idea: Represent tour by a matrix
  - An element m_{ij} of the matrix is 1 if the i-th city is visited in the j-th step and 0 otherwise
  - Each matrix element will be represented by a neuron

Solving Optimization Problems (3)



$$
\begin{array}{cc}
\text{city} & \\
\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} & \\
\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} & \begin{array}{l} 1. \\ 2. \\ 3. \\ 4. \end{array} \quad \text{step}
\end{array}
$$

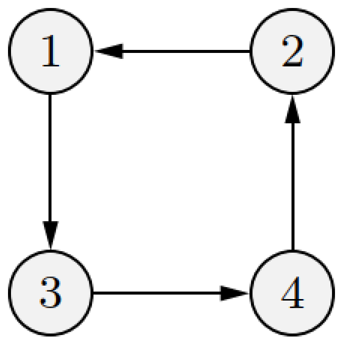
- Minimization of the tour length

$$E_1 = \sum_{j_1=1}^{n} \sum_{j_2=1}^{n} \sum_{i=1}^{n} d_{j_1 j_2} \cdot m_{i j_1} \cdot m_{(i \bmod n)+1, j_2}.$$

  - Energy function

$$E_1 = -\frac{1}{2} \sum_{\substack{(i_1,j_1)\in\{1,\dots,n\}^2 \\ (i_2,j_2)\in\{1,\dots,n\}^2}} -d_{j_1 j_2} \cdot \left(\delta_{(i_1 \bmod n)+1, i_2} + \delta_{i_1, (i_2 \bmod n)+1}\right) \cdot m_{i_1 j_1} \cdot m_{i_2 j_2}.$$

Solving Optimization Problems (4)



$$
\begin{array}{c c}
\text{city} & \\
\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} & \\
\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} & \begin{array}{l} 1. \\ 2. \\ 3. \\ 4. \end{array} \quad \text{step}
\end{array}
$$

- Additional conditions that have to be satisfied:
  - Each city is visited on exactly one step of the tour

$$E_2 = -\frac{1}{2} \sum_{\substack{(i_1,j_1)\in\{1,\dots,n\}^2 \\ (i_2,j_2)\in\{1,\dots,n\}^2}} -2\delta_{j_1 j_2} \cdot m_{i_1 j_1} \cdot m_{i_2 j_2} + \sum_{(i,j)\in\{1,\dots,n\}^2} -2m_{ij}$$

  - On each step of the tour exactly one city is visited

$$E_3 = -\frac{1}{2} \sum_{\substack{(i_1,j_1)\in\{1,\dots,n\}^2 \\ (i_2,j_2)\in\{1,\dots,n\}^2}} -2\delta_{i_1 i_2} \cdot m_{i_1 j_1} \cdot m_{i_2 j_2} + \sum_{(i,j)\in\{1,\dots,n\}^2} -2m_{ij}.$$

Solving Optimization Problems (5)

- Combining the energy functions

$$E = aE_1 + bE_2 + cE_3$$

 - From the resulting energy function we can read the weights

$$w_{(i_1,j_1)(i_2,j_2)} = \underbrace{-ad_{j_1 j_2} \cdot (\delta_{(i_1 \bmod n)+1, i_2} + \delta_{i_1, (i_2 \bmod n)+1})}_{\text{from } E_1} \underbrace{-2b\delta_{j_1 j_2}}_{\text{from } E_2} \underbrace{-2c\delta_{i_1 i_2}}_{\text{from } E_3}$$

 - And the threshold values

$$\theta_{(i,j)} = \underbrace{0a}_{\text{from } E_1} \underbrace{-2b}_{\text{from } E_2} \underbrace{-2c}_{\text{from } E_3} = -2(b+c).$$

Solving Optimization Problems (6)

- Hopfield network only rarely finds a tour, let alone an optimal one
  - Hopfield network is unable to switch from a found tour to another with a lower total length
  - Transforming a matrix into another matrix that represents a different tour requires that four neurons (matrix elements) to change their activations
  - Each of these changes violates at least one of the constraints and thus increases the energy
  - All four changes together can result in a smaller energy, but cannot be executed together due to the asynchronous update

Solving Optimization Problems (7)

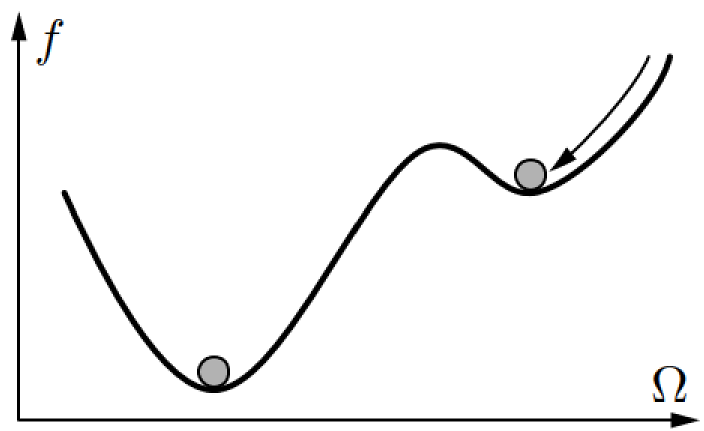
- Discrete Hopfield networks: neuron activation function is integer
- Optimization can be improved if **continuous Hopfield networks** are used
  - Neuron activation function is a real number in [0,1]
  - Fundamental problem of local convergence is not solved in this way

Solving Optimization Problems (8)

- The reason for the difficulties is if the update procedure may get stuck in a local optimum
  - The problem of local optima occurs also with many other optimization methods (for example: gradient descent, hill climbing, alternating optimization)
- Ideas to overcome this difficulty for other optimization methods may be transferred to Hopfield networks
  - One such method, which is very popular, is simulated annealing

Simulated Annealing (1)

- Extension of random or gradient descent that tries to avoid getting stuck
  - Transitions from higher to lower (local) minima should be more probable than vice versa



Simulated Annealing (2)

- Features
  - Random variants of the current solution (candidate) are created
  - Better solution (candidates) are always accepted
  - Worse solution (candidates) are accepted with a probability that depends on:
    - The quality difference between the new and the old solution (candidate)
    - A temperature parameter that is decreased with time
- There is no guarantee that the global optimum is found

Simulated Annealing (3)

- Motivation
  - Physical minimization of the energy if a heated piece of metal is cooled slowly
    - This process is called annealing.
    - It serves the purpose to make the metal easier to work or to machine by relieving tensions and correcting lattice malformations
  - A ball rolls around on an (irregularly) curved surface; minimization of the potential energy of the ball
    - In the beginning the ball is endowed with a certain kinetic energy, which enables it to roll up some slopes of the surface
    - In the course of time, friction reduces the kinetic energy of the ball, so that it finally comes to a rest in a valley of the surface

Simulated Annealing in Hopfield Networks

- Algorithm
  - All neuron activations are initialized randomly
  - The neurons of the Hopfield network are traversed repeatedly (for example, in some random order)
  - For each neuron, it is determined whether an activation change leads to a reduction of the network energy or not
  - An activation change that reduces the network energy is always accepted (in the normal update process, only such changes occur)
  - However, if an activation change increases the network energy, it is accepted with a certain probability