

# Recurrent neural networks

- Architectures
  - Fully recurrent networks
  - Partially recurrent networks
- Dynamics of recurrent networks
  - Continuous time dynamics
  - Discrete time dynamics
- Applications

# Recurrent neural networks

- Architecture
  - Contains feedback connections
  - Depending on the density of feedback connections there are:
    - Fully recurrent networks (Hopfield model)
    - Partially recurrent networks:
      - With contextual units (Elman model, Jordan model)
      - Cellular networks (Chua-Yang model)
- Applications
  - Associative memories
  - Combinatorial optimization problems
  - Prediction
  - Image processing
  - Dynamical systems and chaotical phenomena modelling

# Hopfield networks

## Architecture:

N fully connected units

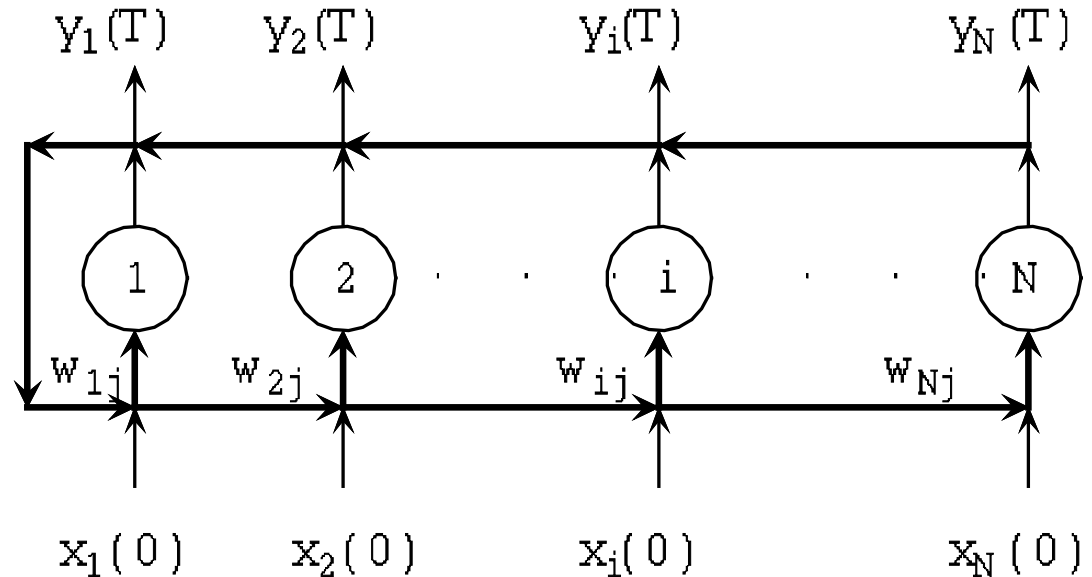
## Activation function:

Signum/Heaviside

Logistica/Tanh

## Parameters:

weight matrix



**Notations:**  $x_i(t)$  – potential (state) of the neuron  $i$  at moment  $t$

$y_i(t)=f(x_i(t))$  – the output signal generated by unit  $i$  at moment  $t$

$I_i(t)$  – the input signal

$w_{ij}$  – weight of connection between  $j$  and  $i$

# Hopfield networks

- Functioning:
- the output signal is generated by the evolution of a dynamical system
  - Hopfield networks are equivalent to dynamical systems

## Network state:

- the vector of neuron's state  $X(t)=(x_1(t), \dots, x_N(t))$

or

- output signals vector  $Y(t)=(y_1(t), \dots, y_N(t))$

## Dynamics:

- Discrete time – recurrence relations (difference equations)
- Continuous time – differential equations

# Hopfield networks

Discrete time functioning:

the network state corresponding to moment  $t+1$  depends on the network state corresponding to moment  $t$

Network's state:  $Y(t)$

Variants:

- **Asynchronous:** only one neuron can change its state at a given time
- **Synchronous:** all neurons can simultaneously change their states

Network's answer: the stationary state of the network

# Hopfield networks

Asynchronous  
variant:

$$y_{i^*}(t+1) = f\left(\sum_{j=1}^N w_{i^*j} y_j(t) + I_{i^*}(t)\right)$$

$$y_i(t+1) = y_i(t), \quad i \neq i^*$$

Choice of  $i^*$ :

- systematic scan of  $\{1, 2, \dots, N\}$
- random (but such that during  $N$  steps each neuron changes its state just once)

Network simulation:

- choose an initial state (depending on the problem to be solved)
- compute the next state until the network reach a stationary state (the distance between two successive states is less than  $\epsilon$ )

# Hopfield networks

Synchronous variant:

$$y_i(t+1) = f\left(\sum_{j=1}^N w_{ij} y_j(t) + I_i(t)\right), \quad i = \overline{1, N}$$

Either continuous or discrete activation functions can be used

Functioning:

Initial state

REPEAT

    compute the new state starting from the current one

UNTIL < the difference between the current state and the previous one is small enough >

# Hopfield networks

Continuous time functioning:

$$\frac{dx_i(t)}{dt} = -x_i(t) + \sum_{j=1}^N w_{ij} f(x_j(t)) + I_i(t), \quad i = \overline{1, N}$$

**Network simulation:** solve (numerically) the system of differential equations for a given initial state  $x_i(0)$

**Example:** Explicit Euler method

$$\frac{x_i(t+h) - x_i(t)}{h} \cong -x_i(t) + \sum_{j=1}^N w_{ij} f(x_j(t)) + I_i(t), \quad i = \overline{1, N}$$

$$x_i(t+h) \cong (1-h)x_i(t) + h\left(\sum_{j=1}^N w_{ij} f(x_j(t)) + I_i(t)\right), \quad i = \overline{1, N}$$

Constant input signal :

$$x_i^{new} \cong (1-h)x_i^{old} + h\left(\sum_{j=1}^N w_{ij} f(x_j^{old}) + I_i\right), \quad i = \overline{1, N}$$



# Stability properties

## Possible behaviours of a network:

- $X(t)$  converged to a stationary state  $X^*$  (fixed point of the network dynamics)
- $X(t)$  oscillates between two or more states
- $X(t)$  has a chaotic behavior or  $\|X(t)\|$  becomes too large

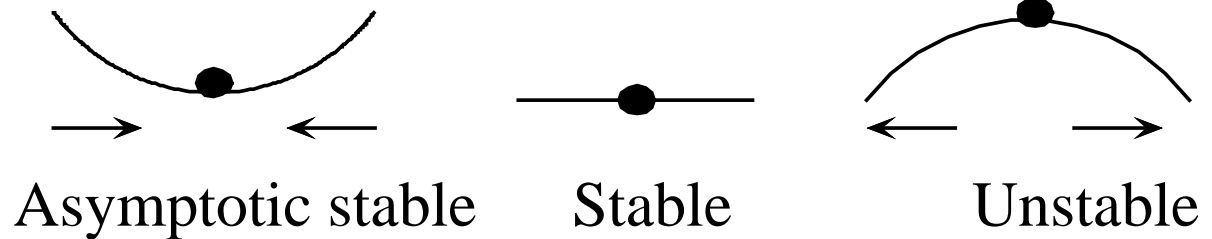
## Useful behaviors:

- **The network converges to a stationary state**
  - Many stationary states: associative memory
  - Unique stationary state: combinatorial optimization problems
- The network has a periodic behavior
  - Modelling of cycles

**Obs.** Most useful situation: the network converges to a stable stationary state

# Stability properties

Illustration:



Formalization:

$$\frac{dX(t)}{dt} = F(X(t)), \quad X(0) = X_0$$
$$F(X^*) = 0$$

$X^*$  is asymptotic stable (wrt the initial conditions) if it is  
stable  
attractive

# Stability properties

## Stability:

$X^*$  is stable if for all  $\varepsilon > 0$  there exists  $\delta(\varepsilon) > 0$  such that:

$$\|X_0 - X^*\| < \delta(\varepsilon) \text{ implies } \|X(t; X_0) - X^*\| < \varepsilon$$

## Attractive:

$X^*$  is attractive if there exists  $\delta > 0$  such that:

$$\|X_0 - X^*\| < \delta \text{ implies } X(t; X_0) \rightarrow X^*$$

In order to study the asymptotic stability one can use the Lyapunov method.

# Stability properties

Lyapunov  
function:

$$V : R^N \rightarrow R, \quad \text{bounded}$$
$$\frac{dV(X(t))}{dt} < 0, \quad t > 0$$

- If one can find a Lyapunov function for a system then its stationary solutions are asymptotically stable
- The Lyapunov function is similar to the energy function in physics (the physical systems naturally converges to the lowest energy state)
- The states for which the Lyapunov function is minimum are stable states
- Hopfield networks satisfying some properties have Lyapunov functions.

# Stability properties

## Stability result for continuous neural networks

If:

- the weight matrix is symmetrical ( $w_{ij}=w_{ji}$ )
- the activation function is strictly increasing ( $f'(u)>0$ )
- the input signal is constant ( $I(t)=I$ )

Then all stationary states of the network are asymptotically stable

Associated Lyapunov function:

$$V(x_1, \dots, x_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij} f(x_i) f(x_j) - \sum_{i=1}^N f(x_i) I_i + \sum_{i=1}^N \int_0^{f(x_i)} f^{-1}(z) dz$$

# Stability properties

Stability result for discrete neural networks (asynchronous case)

If:

- the weight matrix is symmetrical ( $w_{ij}=w_{ji}$ )
- the activation function is signum or Heaviside
- the input signal is constant ( $I(t)=I$ )

Then all stationary states of the network are asymptotically stable

Corresponding Lyapunov function

$$V(y_1, \dots, y_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij} y_i y_j - \sum_{i=1}^N y_i I_i$$

# Stability properties

This result means that:

- All stationary states are stable
- Each stationary state has attached an attraction region (if the initial state of the network is in the attraction region of a given stationary state then the network will converge to that stationary state)

Remarks:

- This property is useful for associative memories
- For synchronous discrete dynamics this result is no more true, but the network converges toward either fixed points or cycles of period two

# Associative memories

**Memory** = system to store and recall the information

## Address-based memory:

- Localized storage: all components bytes of a value are stored together at a given address
- The information can be recalled based on the address

## Associative memory:

- The information is distributed and the concept of address does not have sense
- The recall is based on the content (one starts from a clue which corresponds to a partial or noisy pattern)



# Associative memories

## Properties:

- Robustness

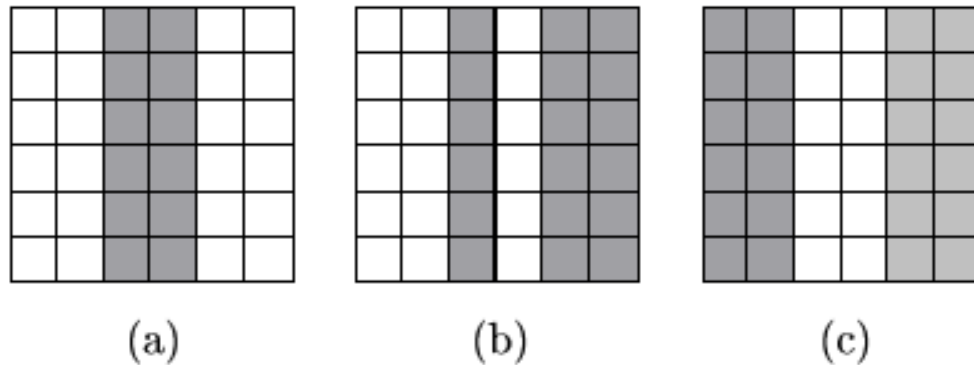
## Implementation:

- Hardware:
  - Electrical circuits
  - Optical systems
- Software:
  - Hopfield networks simulators

# Associative memories

Software simulations of associative memories:

- The information is binary: vectors having elements from  $\{-1,1\}$
- Each component of the pattern vector corresponds to a unit in the networks



Example (a)

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

# Associative memories

## Associative memories design:

- Fully connected network with  $N$  signum units ( $N$  is the patterns size)

## Patterns storage:

- Set the weights values (elements of matrix  $W$ ) such that the patterns to be stored become fixed points (stationary states) of the network dynamics

## Information recall:

- Initialize the state of the network with a clue (partial or noisy pattern) and let the network to evolve toward the corresponding stationary state.

# Associative memories

Patterns to be stored:  $\{X^1, \dots, X^L\}$ ,  $X^l$  in  $\{-1, 1\}^N$

Methods:

- Hebb rule
- Pseudo-inverse rule (Diederich – Opper algorithm)

Hebb rule:

- It is based on the Hebb's principle: “the synaptic permeability of two neurons which are simultaneously activated is increased”

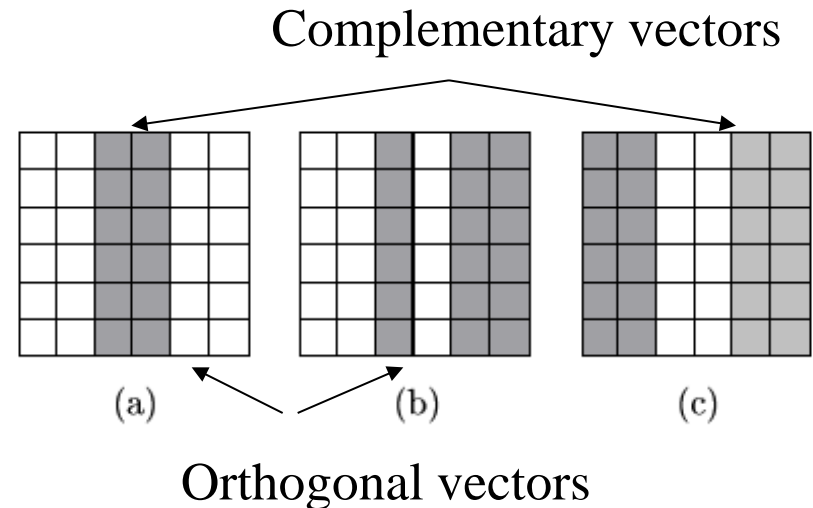
$$w_{ij} = \frac{1}{N} \sum_{l=1}^L x_i^l x_j^l$$

# Associative memories

$$w_{ij} = \frac{1}{N} \sum_{l=1}^L x_i^l x_j^l$$

Properties of the Hebb's rule:

- If the vectors to be stored are orthogonal (statistically uncorrelated) then all of them become fixed points of the network dynamics
- Once the vector  $X$  is stored the vector  $-X$  is also stored
- An improved variant: the pseudo-inverse method



# Associative memories

Pseudo-inverse method:

$$w_{ij} = \frac{1}{N} \sum_{l,k} x_i^l (Q^{-1})_{lk} x_j^l$$

$$Q_{lk} = \frac{1}{N} \sum_{i=1}^N x_i^l x_i^k$$

- If  $Q$  is invertible then all elements of  $\{X^1, \dots, X^L\}$  are fixed points of the network dynamics
- In order to avoid the costly operation of inversion one can use an iterative algorithm for weights adjustment

# Associative memories

Diederich-Opper algorithm :

---

$t := 0$

Initialize  $W(0)$  using the Hebb rule

REPEAT

FOR  $l := 1, L$  DO

$$y_i^l := \sum_{j=1}^N w_{ij}(t) x_j^l, \quad i = \overline{1, N}$$

$$w_{ij}(t+1) := w_{ij}(t) + \frac{1}{N} (x_i^l - y_i^l) x_j^l, \quad i = \overline{1, N}, j = \overline{1, N}$$

$t := t + 1$

UNTIL  $\|W(t) - W(t-1)\| < \epsilon$

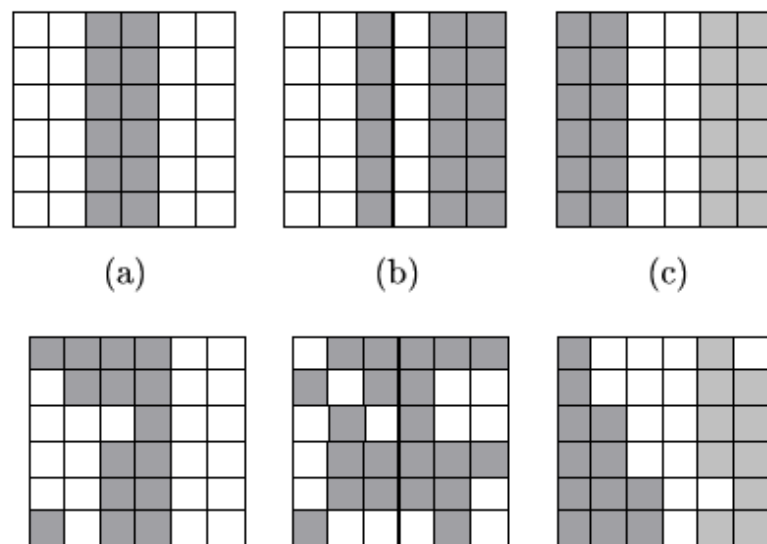
---

# Associative memories

## Recall process:

- Initialize the network state with a starting clue
- Simulate the network until the stationary state is reached.

## Stored patterns



## Noisy patterns (starting clues)



# Associative memories

## Storage capacity:

- The number of patterns which can be stored and recalled (exactly or approximately)
- Exact recall:  $\text{capacity} = N / (4 \ln N)$
- Approximate recall ( $\text{prob}(\text{error}) = 0.005$ ):  $\text{capacity} = 0.15 * N$

## Spurious attractors:

- These are stationary states of the networks which were not explicitly stored but they are the result of the storage method.

## Avoiding the spurious states

- Modifying the storage method
- Introducing random perturbations in the network's dynamics

# Solving optimization problems

- First approach: Hopfield & Tank (1985)
  - They propose the use of a Hopfield model to solve the traveling salesman problem.
  - The basic idea is to design a network whose **energy function** is similar to the **cost function** of the problem (e.g. the tour length) and to let the network to **naturally evolve** toward the state of minimal energy; this state would represent the problem's solution.

# Solving optimization problems

A constrained optimization problem:

find  $(y_1, \dots, y_N)$  satisfying:

it minimizes a cost function  $C: \mathbb{R}^N \rightarrow \mathbb{R}$

it satisfies some constraints as  $R_k(y_1, \dots, y_N) = 0$  with  
 $R_k$  nonnegative functions

Main steps:

- Transform the constrained optimization problem in an unconstrained optimization one (penalty method)
- Rewrite the cost function as a Lyapunov function
- Identify the values of the parameters ( $W$  and  $I$ ) starting from the Lyapunov function
- Simulate the network

# Solving optimization problems

Step 1: Transform the constrained optimization problem in an unconstrained optimization one

$$C^*(y_1, \dots, y_N) = aC(y_1, \dots, y_N) + \sum_{k=1}^r b_k R_k(y_1, \dots, y_N)$$

$$a, b_k > 0$$

The values of a and b are chosen such that they reflect the relative importance of the cost function and constraints

# Solving optimization problems

Step 2: Reorganizing the cost function as a Lyapunov function

$$C(y_1, \dots, y_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij}^{obj} y_i y_j - \sum_{i=1}^N I_i^{obj} y_i$$

$$R_k(y_1, \dots, y_N) = -\frac{1}{2} \sum_{i,j=1}^N w_{ij}^k y_i y_j - \sum_{i=1}^N I_i^k y_i, \quad k = \overline{1, r}$$

**Remark:** This approach works only for cost functions and constraints which are linear or quadratic

# Solving optimization problems

Step 3: Identifying the network parameters:

$$w_{ij} = aw_{ij}^{obj} + \sum_{k=1}^r b_k w_{ij}^k, \quad i, j = \overline{1, N}$$

$$I_i = aI_i^{obj} + \sum_{k=1}^r b_k I_i^k, \quad i = \overline{1, N}$$

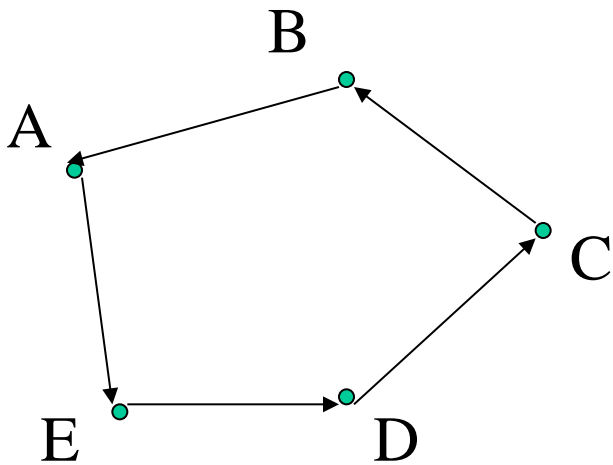
# Solving optimization problems

Designing a neural network for TSP (n towns):

$N=n*n$  neurons

The state of the neuron (i,j) is interpreted as follows:

- 1 - the town i is visited at time j
- 0 - otherwise



AEDCB

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

# Solving optimization problems

## Constraints:

- at a given time only one town is visited (each column contains exactly one value equal to 1)
- each town is visited only once (each row contains exactly one value equal to 1)

	1	2	3	4	5
A	1	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	1	0	0
E	0	1	0	0	0

## Cost function:

the tour length = sum of distances between towns visited at consecutive time moments



# Solving optimization problems

Constraints and cost function:

$$\sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 = 0$$

$$\sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 = 0$$

$$C(Y) = \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1})$$

Cost function in the unconstrained case:

$$C^*(Y) = \frac{a}{2} \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1}) + \frac{b}{2} \left( \sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 + \sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 \right)$$

# Solving optimization problems

$$V(Y) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n w_{ij,kl} y_{ij} y_{kl} - \sum_{i=1}^n \sum_{j=1}^n y_{ij} I_{ij}$$

$$C^*(Y) = \frac{a}{2} \sum_{i=1}^n \sum_{k=1, k \neq i}^n \sum_{j=1}^n c_{ik} y_{ij} (y_{k,j-1} + y_{k,j+1}) +$$

$$\frac{b}{2} \left( \sum_{j=1}^n \left( \sum_{i=1}^n y_{ij} - 1 \right)^2 + \sum_{i=1}^n \left( \sum_{j=1}^n y_{ij} - 1 \right)^2 \right)$$

Identified parameters:

$$w_{ij,kl} = -ac_{ik} (\delta_{l,j-1} + \delta_{l,j+1}) - b(\delta_{ik} + \delta_{jl} + \delta_{ik} \delta_{jl})$$

$$w_{ij,ij} = 0$$

$$I_{ij} = 2b$$

# Prediction in time series

- Time series = sequence of values measured at successive moments of time
- Examples:
  - Currency exchange rate evolution
  - Stock price evolution
  - Biological signals (EKG)
- Aim of time series analysis: predict the future value(s) in the series

# Time series

The prediction (forecasting) is based on a model which describes the dependency between previous values and the next value in the series.

$$x(t + 1) = F(x(t), x(t - 1), \dots, x(t - p + 1), \varphi_1, \varphi_2, \dots, \varphi_s)$$

Order of the model



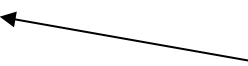
Parameters corresponding  
to external factors

# Time series

The model associated to a time series can be:

- Linear
- Nonlinear
- Deterministic
- Stochastic

Example: autoregressive model (AR(p))

$$x(t+1) = \sum_{i=0}^{p-1} \alpha_i x(t-i) + \epsilon$$


noise = random variable from  
 $N(0,1)$

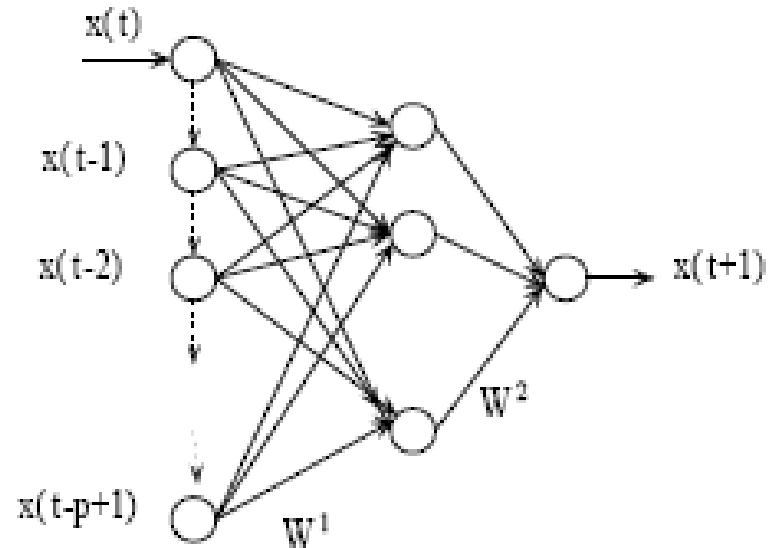
# Time series

## Neural networks. Variants:

- The order of the model is known
  - Feedforward neural network with delayed input layer (p input units)
- The order of the model is unknown
  - Network with contextual units (Elman network)

# Networks with delayed input layer

Architecture:



Functioning:

$$y = \sum_{k=1}^K w_k^2 f\left(\sum_{j=0}^{p-1} w_{kj}^1 x(t-j)\right)$$

# Networks with delayed input layer

## Training:

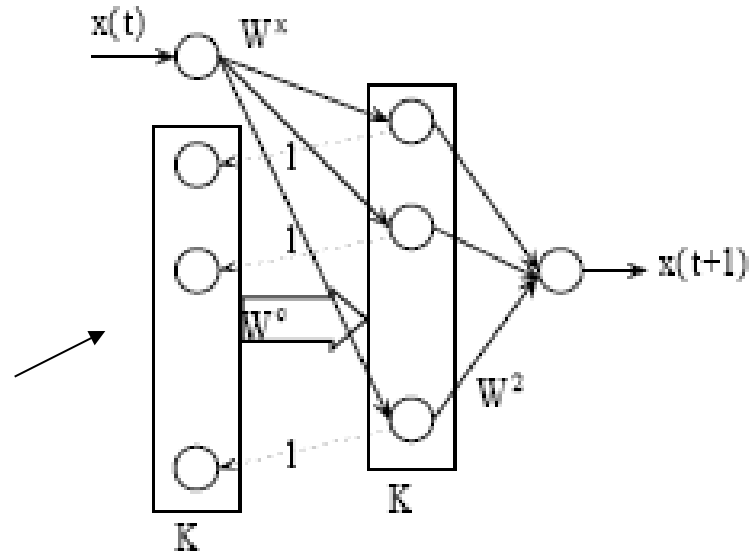
- Training set:  $\{((x_l, x_{l-1}, \dots, x_{l-p+1}), x_{l+1})\}_{l=1..L}$
- Training algorithm: BackPropagation
- Drawback: needs the knowledge of  $p$



# Elman network

Architecture:

Contextual units



Functioning:

$$y = \sum_{k=1}^K w_k^2 h_k(t)$$

$$h_k(t) = f \left( w_k^x x(t) + \sum_{j=1}^K w_{kj}^c h_j(t-1) \right) \quad h_j(0) = 0.$$

Rmk: the contextual units contain copies of the outputs of the hidden layers corresponding to the previous moment

# Elman network

## Training

Training set :  $\{(x(1),x(2)),(x(2),x(3)),\dots,(x(t-1),x(t))\}$

Sets of weights:

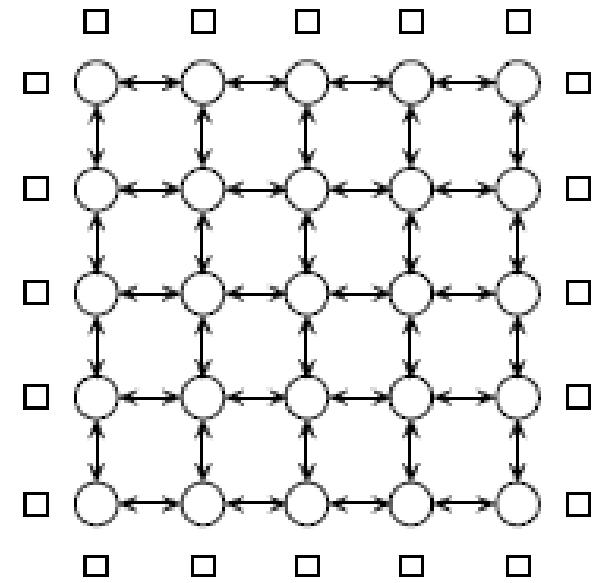
- Adaptive:  $W^x, W^c$  si  $W^2$
- Fixed: the weights of the connections between the hidden and the contextual layers.

Training algorithm: BackPropagation

# Cellular networks

## Architecture:

- All units have a double role: input and output units
- The units are placed in the nodes of a two dimensional grid
- Each unit is connected only with units from its neighborhood (the neighborhoods are defined as in the case of Kohonen's networks)
- Each unit is identified through its position  $p=(i,j)$  in the grid



virtual cells  
(used to define  
the context for  
border cells)

# Cellular networks

Activation function: ramp

$$f(u) = (|u + 1| - |u - 1|) / 2$$

Notations:

$X_p(t)$  – state of unit p at time t

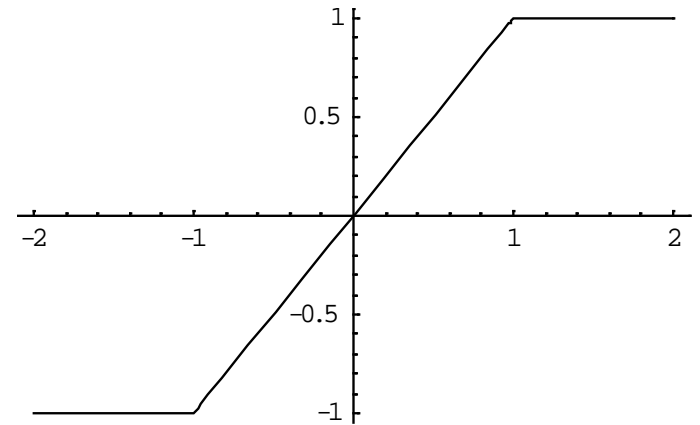
$Y_p(t)$  - output signal

$U_p(t)$  – control signal

$I_p(t)$  – input from the environment

$a_{pq}$  – weight of connection between unit q and unit p

$b_{pq}$  - influence of control signal  $U_q$  on unit p



# Cellular networks

Functioning:

$$\frac{dx_p(t)}{dt} = -x_p(t) + \sum_{q \in V(p)} a_{pq} y_q(t) + \sum_{q \in V(p)} b_{pq} u_q(t) + i_p(t), \quad p \in \mathcal{L}$$

Signal generated by other units
Control signal

Input signal

Remarks:

- The grid has a boundary of fictitious units (which usually generate signals equal to 0)
- Particular case: the weights of the connections between neighboring units do not depend on the positions of units

**Example:** if  $p=(i,j)$ ,  $q=(i-1,j)$ ,  $p'=(i',j')$ ,  $q'=(i'-1,j')$  then

$$a_{pq} = a_{p'q'} = a_{-1,0}$$

# Cellular networks

These networks are called cloning template cellular networks

Example:

$$V_1(i, j) = \{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$$

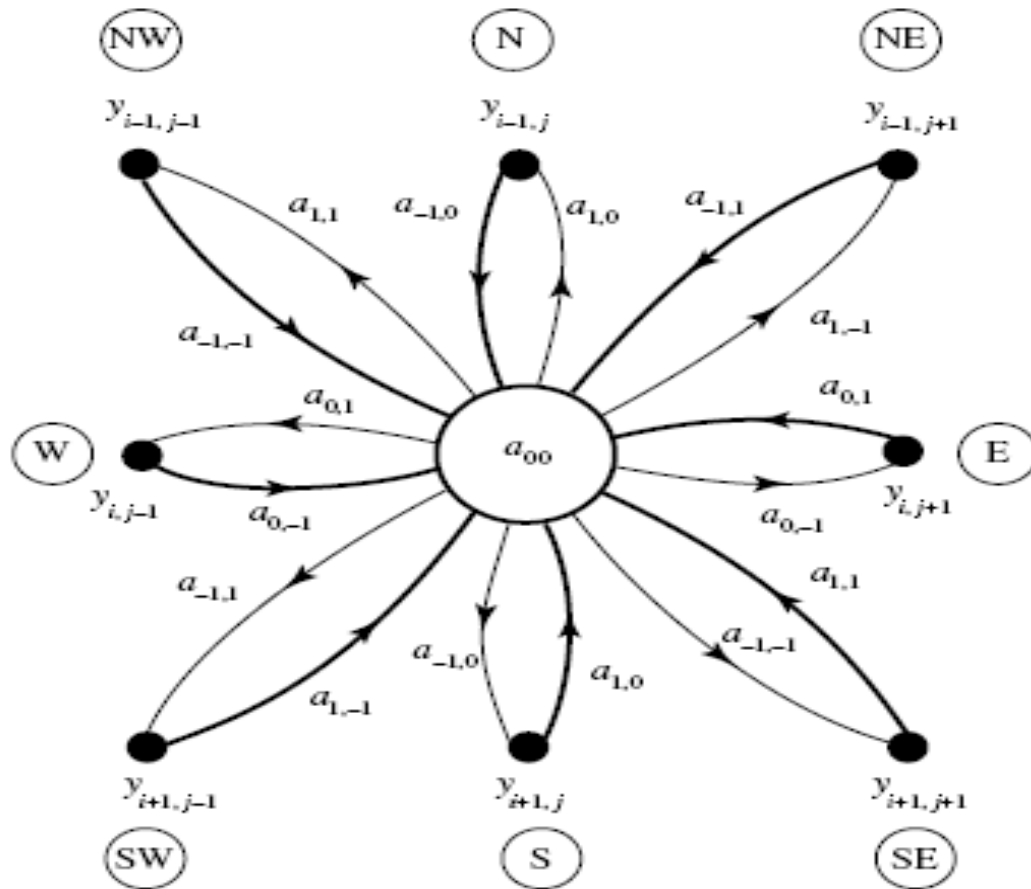
$$A = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \quad B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

$$\frac{dx_{i,j}(t)}{dt} = -x_{i,j}(t) + \sum_{(k,l) \in V^*} a_{k,l} y_{i+k,j+l}(t) + \sum_{(k,l) \in V^*} b_{k,l} u_{i+k,j+l}(t) + I, \quad i, j \in \{1, \dots, n\}$$

$$V^* = \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\}$$

# Cellular networks

Illustration of the cloning template elements



# Cellular networks

Software simulation = equivalent to numerical solving of a differential system (initial value problem)

Explicit Euler method

$$x_p(t+1) = (1-h)x_p(t) + h\left(\sum_{(k,l) \in V^*} a_{k,l}y_{i+k,j+l}(t) + \sum_{(k,l) \in V^*} b_{k,l}u_{i+k,j+l}(t) + I\right), \quad i, j \in \{1, \dots, n\}$$

## Applications:

- Gray level image processing
- Each pixel corresponds to a unit of the network
- The gray level is encoded by using real values from  $[-1, 1]$



# Cellular networks

Image processing:

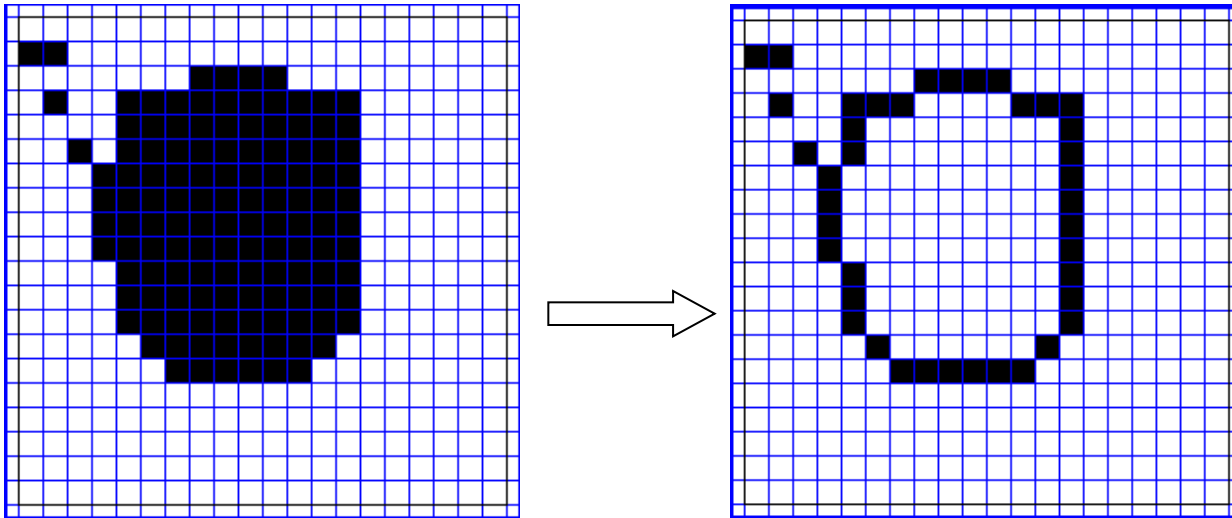
- Depending on the choice of templates, of control signal ( $u$ ), initial condition ( $x(0)$ ), boundary conditions ( $z$ ) different image processing tasks can be solved:
  - Edge detection in binary images
  - Gap filling in binary images
  - Noise elimination in binary images
  - Identification of horizontal/vertical line segments

# Cellular networks

Example 1: edge detection  
 $z=-1$ ,  $U$ =input image,  $h=0.1$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$I = -1, X(0) = U$$



[http://www.isiweb.ee.ethz.ch/haenggi/CNN\\_web/CNNsim\\_adv.html](http://www.isiweb.ee.ethz.ch/haenggi/CNN_web/CNNsim_adv.html)

# Cellular networks

Example 2: gap filling

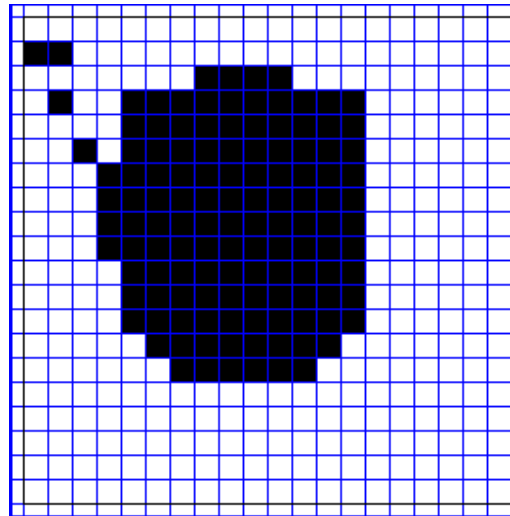
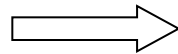
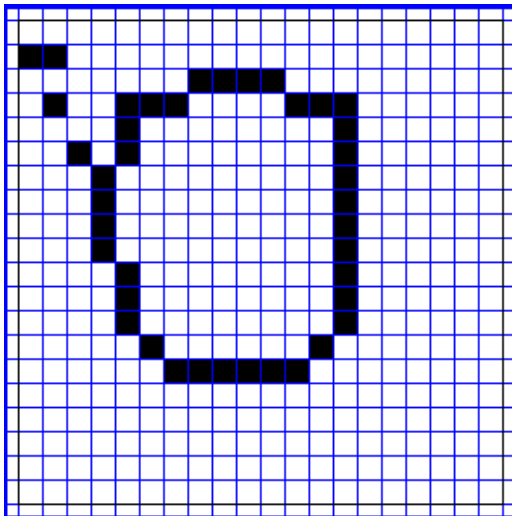
$z=-1$ ,

$U$ =input image,

$h=0.1$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1.5 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$I = 0.5, x_{ij}(0) = 1$  (all pixels are 1)



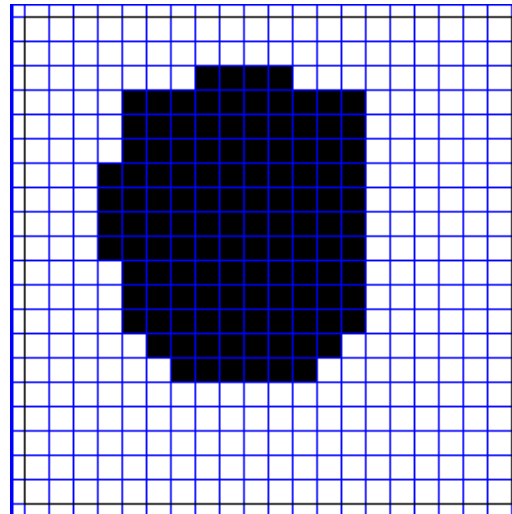
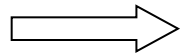
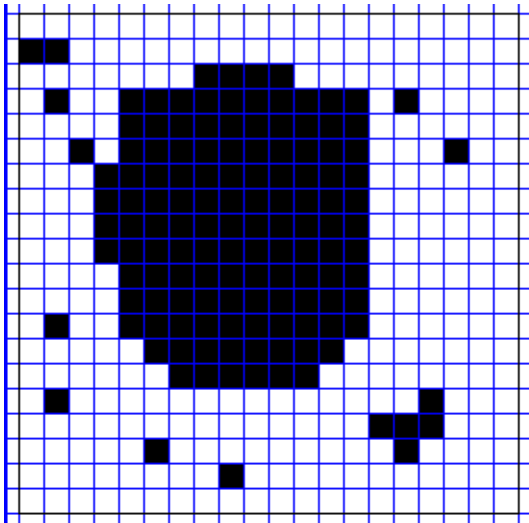
# Cellular networks

Example 3: noise removing

$z=-1$ ,  $U$ =input image,  $h=0.1$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I = 0, X(0) = U$$



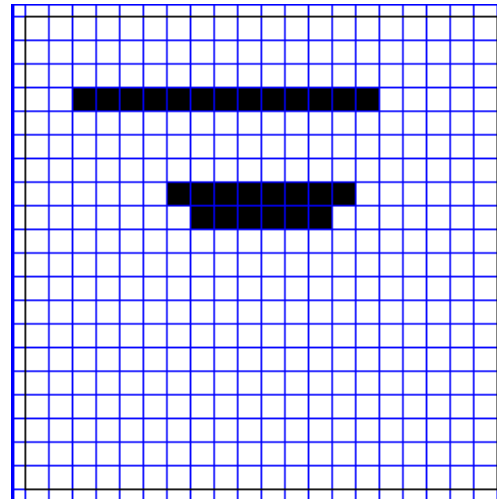
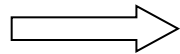
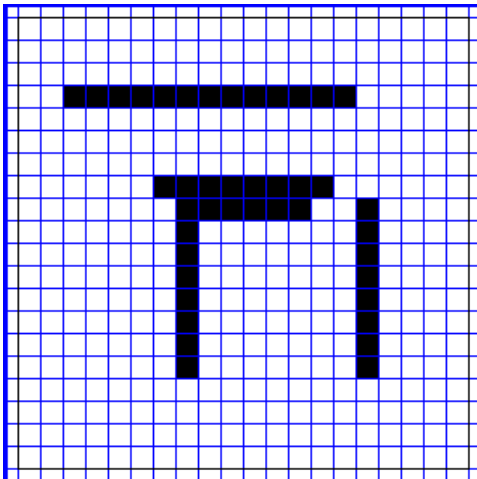
# Cellular networks

Example 4: horizontal line detection

$z=-1$ ,  $U$ =input image,  $h=0.1$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I = -1, X(0) = U$$



# Other related models

Reservoir computing ([www.reservoir-computing.org](http://www.reservoir-computing.org))

## Particularities:

- These models use a set of hidden units (called reservoir) which are arbitrarily connected (their connection weights are randomly set; each of these units realize a nonlinear transformation of the signals received from the input units.
- The output values are obtained by a linear combination of the signals produced by the input units and by the reservoir units.
- Only the weights of connections toward the output units are trained

# Other related models

Reservoir computing ([www.reservoir-computing.org](http://www.reservoir-computing.org))

Variants:

- Temporal Recurrent Neural Network (Dominey 1995)
- Liquid State Machines (Natschläger, Maass and Markram 2002)
- Echo State Networks (Jaeger 2001)
- Decorrelation-Backpropagation Learning (Steil 2004)

# Other related models

## Echo State Networks:

$U(t)$  = input vector

$X(t)$  = reservoir state vector

$Z(t)=[U(t);X(t)]$  = concatenated input and state vectors

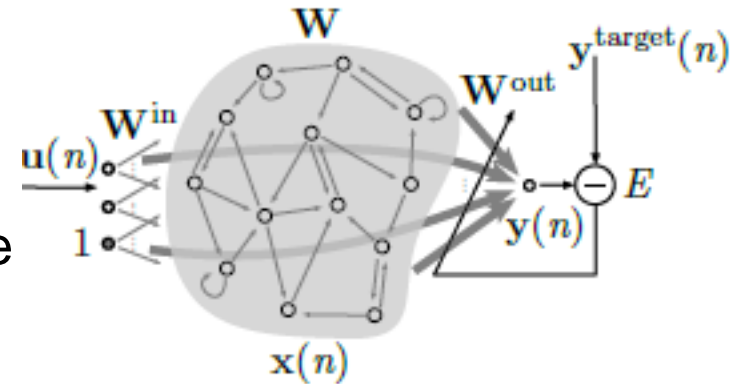
$Y(t)$  = output vector

$X(t)=(1-a)X(t-1)+a \tanh(W^{\text{in}} U(t)+W X(t-1))$

$Y(t)=W^{\text{out}} Z(t)$

$W^{\text{in}}, W$  – random matrices ( $W$  is scaled such that the spectral radius has a predefined value);

$W^{\text{out}}$  - set by training



M. Lukosevicius – Practical Guide to Applying Echo State Networks



# Other related models

Applications of reservoir computing:

- Speech recognition
- Handwritten text recognition
- Robot control
- Financial data prediction
- Real time prediction of epilepsy seizures

# Other related models

Deep learning (<http://deeplearning.net/>)

Particularities:

- Deep architecture = many layers (aim: hierarchical extraction of data features);
- Unsupervised training based on Restricted Boltzmann Machines) followed by a fine tuning of weights using a supervised training (e.g. Backpropagation)

Remarks:

- Boltzmann Machines = recurrent neural networks with binary stochastic units
- Restricted BM = recurrent neural networks with bidirectional connections only between the units belonging to different subsets of units (e.g. subsets: visible units, hidden units)
- There are feed-forward deep neural networks (e.g: Convolutional Neural Networks)

# Other related models

Deep learning (<http://deeplearning.net/>)

## Applications:

- Image classification, objects detection (e.g. Face recognition – Deep Face)
- Speech recognition (Google Brain, Siri)
- Semantic indexing (ex: word2vec) and automated translation
- Dream simulation (<http://npcontemplation.blogspot.ca/2012/02/machine-that-can-dream.html>)