DD2437 – Artificial Neural Networks and Deep Architectures (annda)

Lecture 8: **Hopfield networks and introduction to stochastic networks**

Pawel Herman

Computational Science and Technology (CST)

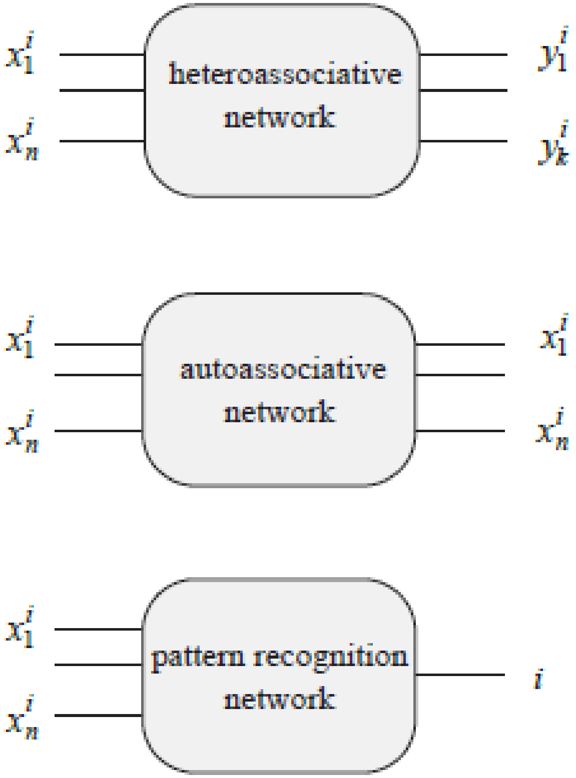
KTH Royal Institute of Technology

February 2018

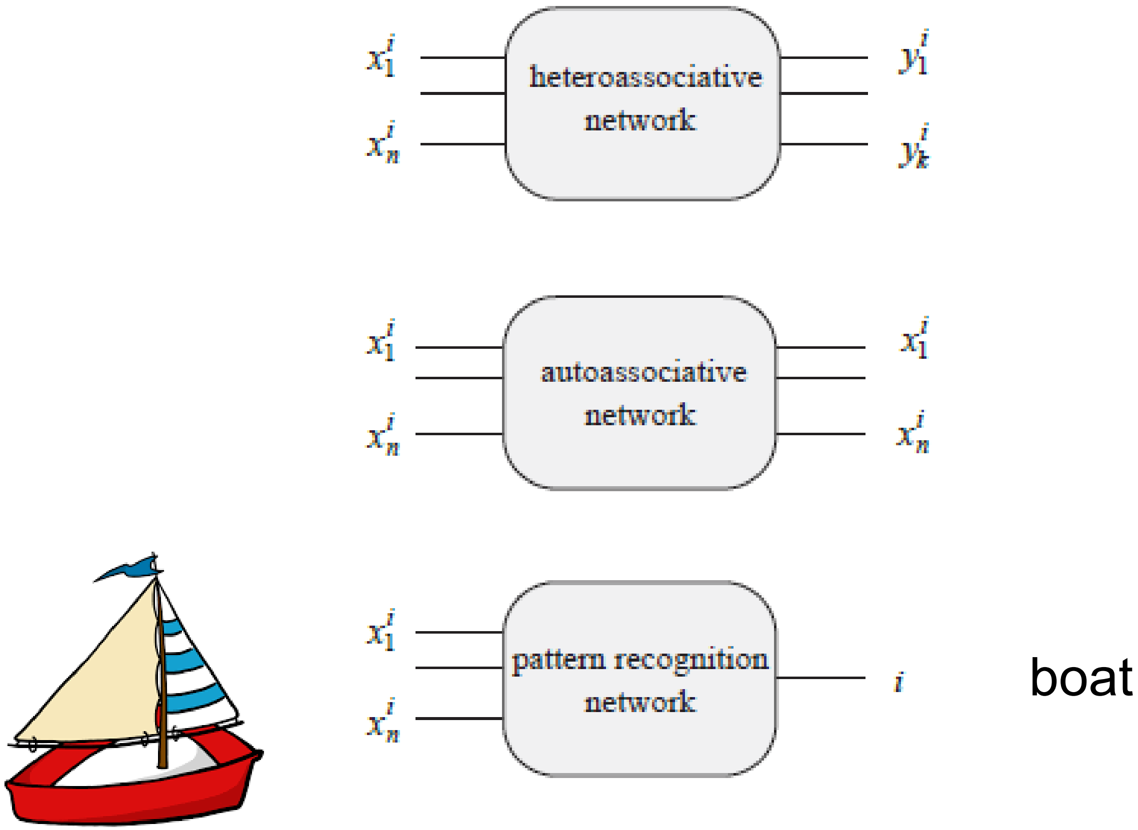
Lecture overview

- Associative memory, learning
- Hopfield networks
- Storage capacity
- Optimisation with Hopfield networks
  - travelling salesman problem (TSP) example

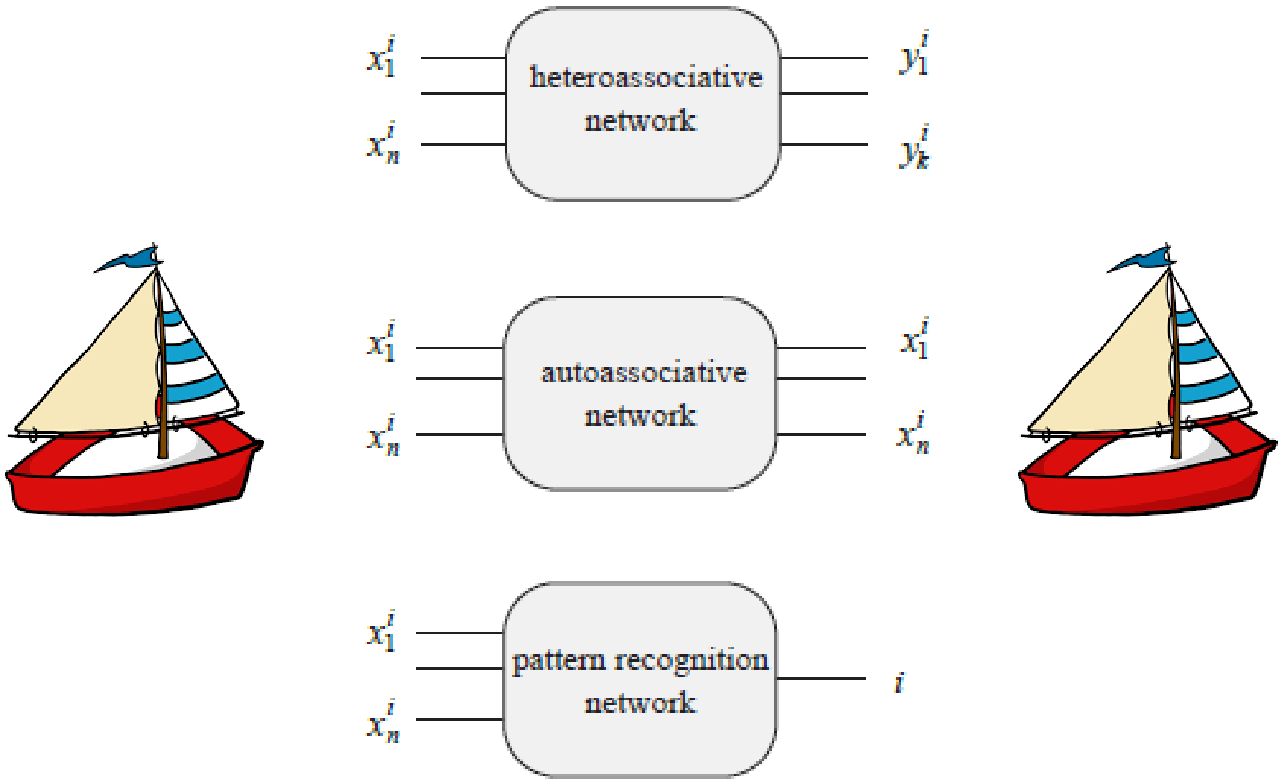
Associative pattern recognition



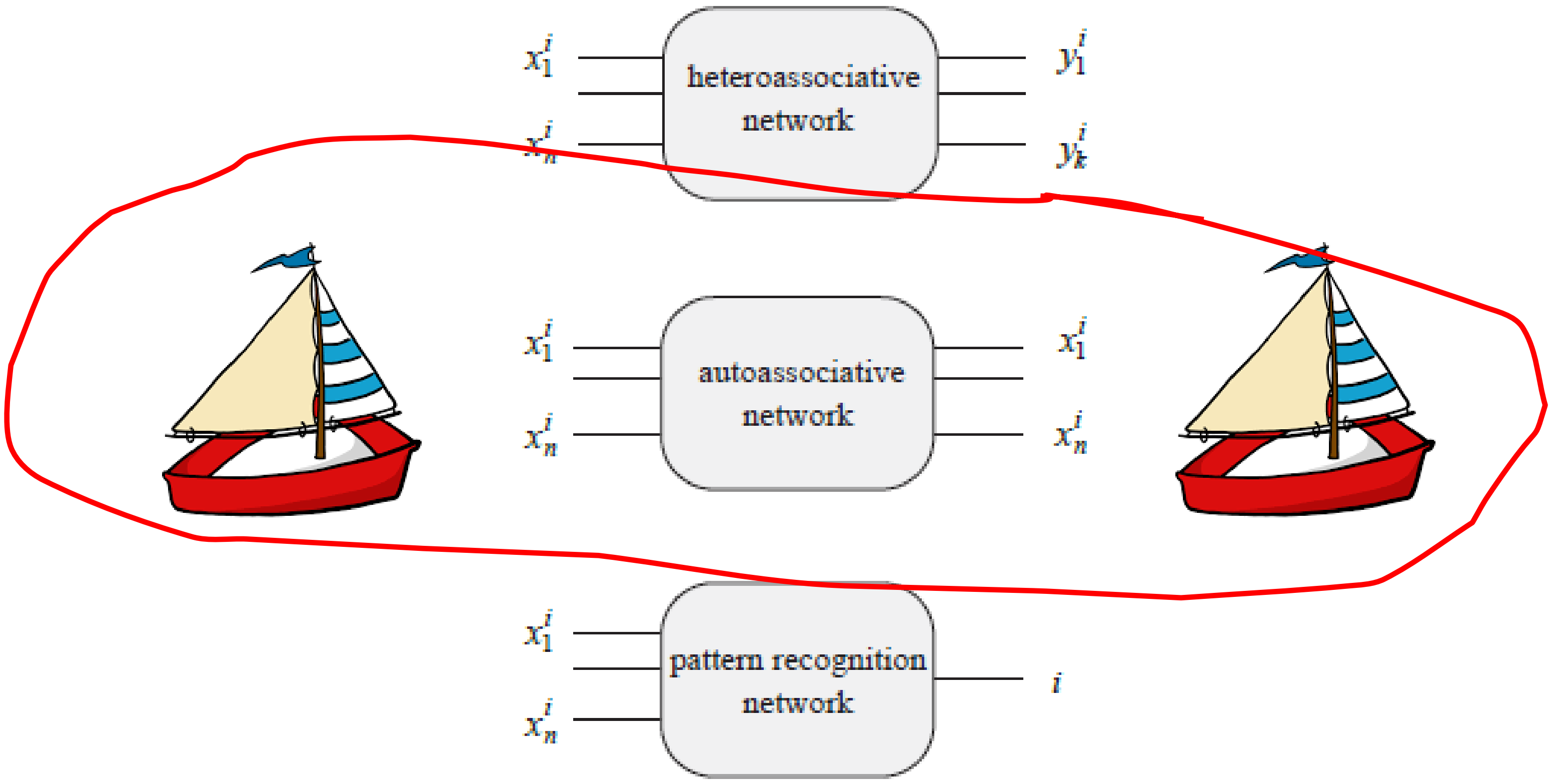
Associative pattern recognition



Associative pattern recognition

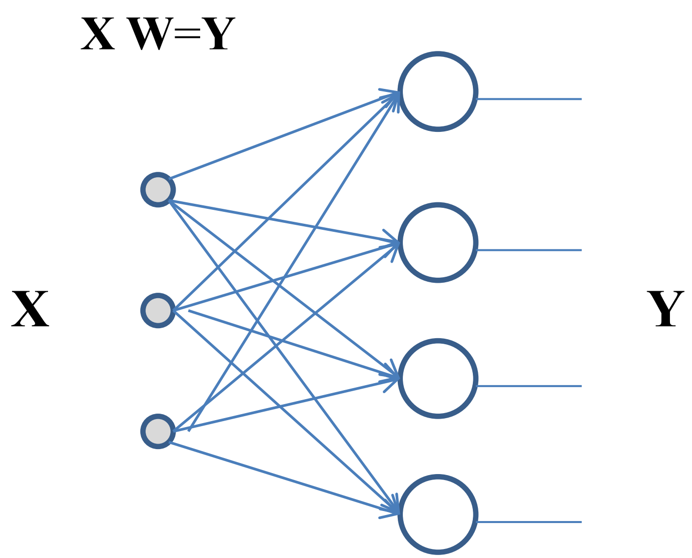


Associative pattern recognition



Linear associative memory networks

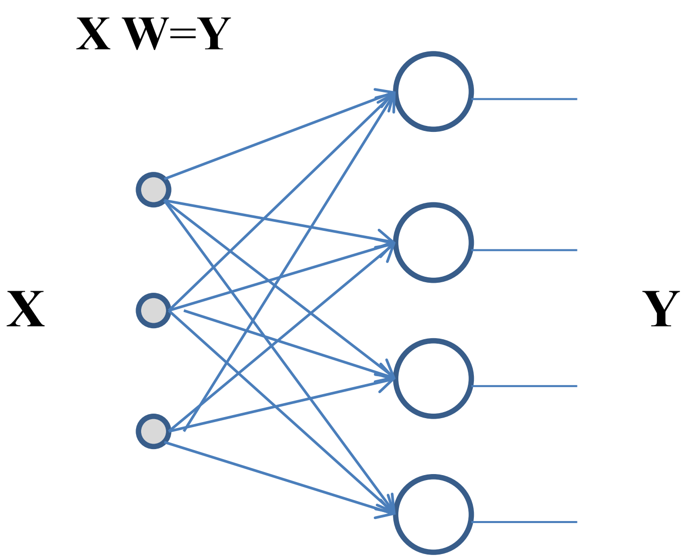
- Simple single layer networks

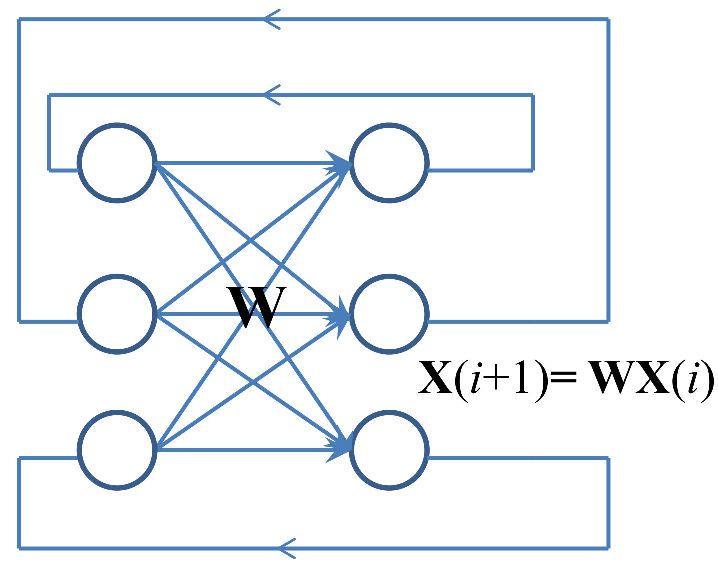


without feedback
(recall is a feedforward step)

Linear associative memory networks

- Simple single layer or recurrent networks





without feedback
(recall is a feedforward step)

<u>autoassociative</u> recurrent network, <u>with feedback</u>
(recall is an iterative process)

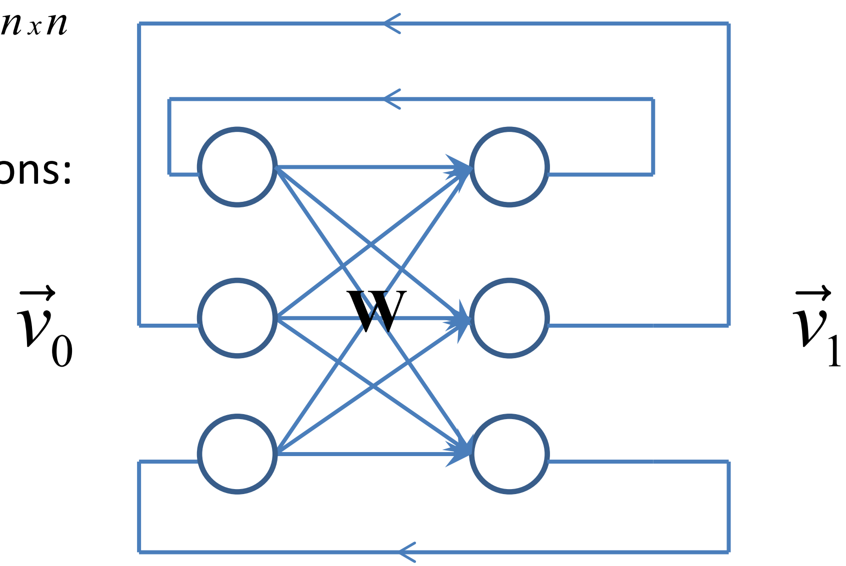
Autoassociative memory network with feedback

- Eigenvector automaton

$\mathbf{W}\vec{x}_i = \lambda_i \vec{x}_i$, $\vec{x}_1,..,\vec{x}_n$ - eigenvectors of $\mathbf{W}_{n\,x\,n}$

For any vector $\vec{v}_0$ and simultaneous computations:

$$\vec{v}_0 = \sum_i \alpha_i \vec{x}_i$$



with feedback, **recurrent** (iterative recall)

Autoassociative memory network with feedback

- Eigenvector automaton

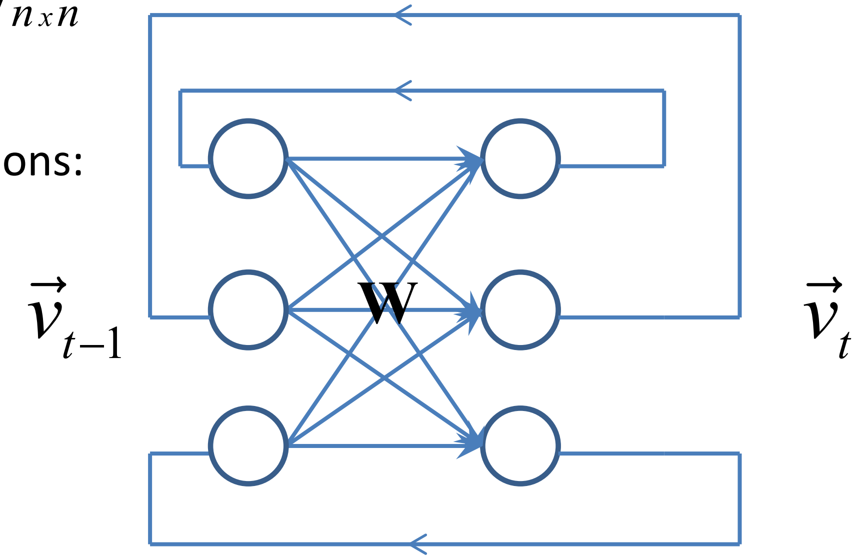
$\mathbf{W}\vec{x}_i = \lambda_i \vec{x}_i$, $\vec{x}_1,..,\vec{x}_n$ - eigenvectors of $\mathbf{W}_{n\,x\,n}$

For any vector $\vec{v}_0$ and simultaneous computations:

$$\vec{v}_0 = \sum_i \alpha_i \vec{x}_i$$

$$\vec{v}_1 = \mathbf{W}\vec{v}_0 = \sum_i \alpha_i \lambda_i \vec{x}_i$$

$$\vec{v}_t = \mathbf{W}\vec{v}_{t-1} = \sum_i \alpha_i \lambda_i^t \vec{x}_i$$



with feedback, **recurrent** (iterative recall)

Autoassociative memory network with feedback

- Eigenvector automaton

$\mathbf{W}\vec{x}_i = \lambda_i \vec{x}_i$, $\vec{x}_1,..,\vec{x}_n$ - eigenvectors of $\mathbf{W}_{n x n}$

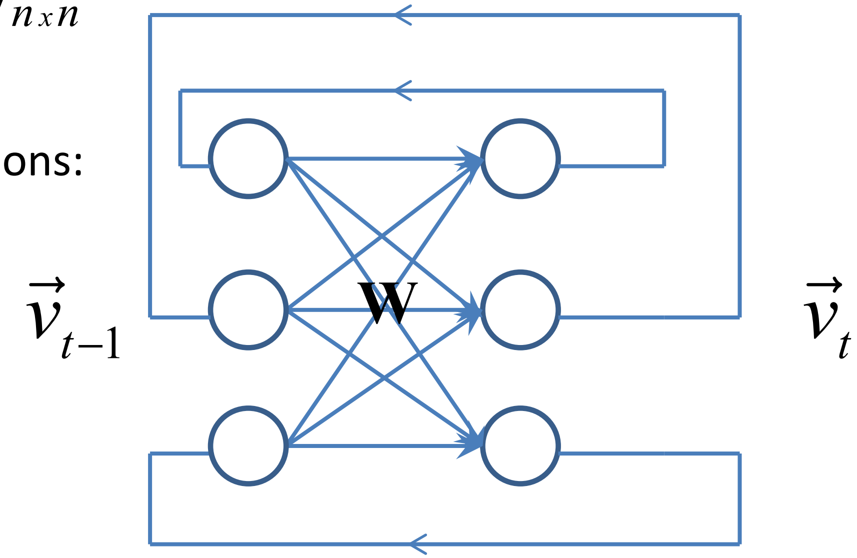
For any vector $\vec{v}_0$ and simultaneous computations:

$$\vec{v}_0 = \sum_i \alpha_i \vec{x}_i$$

$$\vec{v}_1 = \mathbf{W}\vec{v}_0 = \sum_i \alpha_i \lambda_i \vec{x}_i$$

$$\vec{v}_t = \mathbf{W}\vec{v}_{t-1} = \sum_i \alpha_i \lambda_i^t \vec{x}_i$$

$\vec{v}_t \rightarrow \vec{x}_k$ Eigenvector with the highest eigenvalue provided that the corresponding α is non-zero.

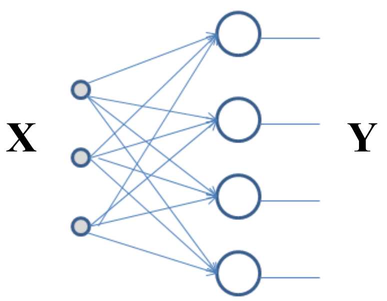


with feedback, **recurrent** (iterative recall)

Associative learning in a single layer network

- Bipolar coding {-1, 1} with sign transform:

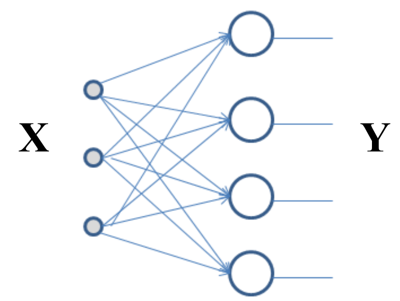
$$\operatorname{sgn}(x) = \begin{cases} 1 \ , x \geq 0 \\ -1 \ , x < 0 \end{cases}$$



Associative learning in a single layer network

- Bipolar coding {-1, 1} with sign transform:

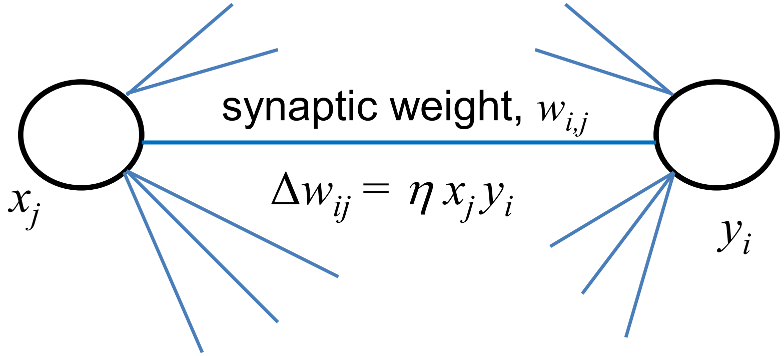
$$\mathrm{sgn}(x) = \begin{cases} 1 \ , x \geq 0 \\ -1 \ , x < 0 \end{cases}$$



- Hebbian learning (correlation learning, outer product)

$\mathbf{W}=\mathbf{W}^1+\mathbf{W}^2+\ldots+\mathbf{W}^m$

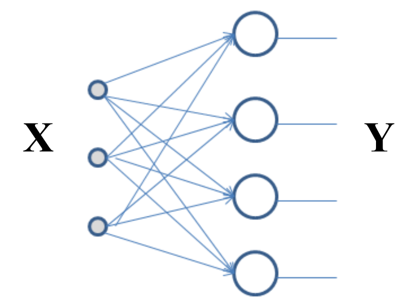
$\mathbf{W}^k= [w_{ij}]=[x_j^k \ y_i^k]$ (*outer product*)



Associative learning in a single layer network

- Bipolar coding {-1, 1} with sign transform:

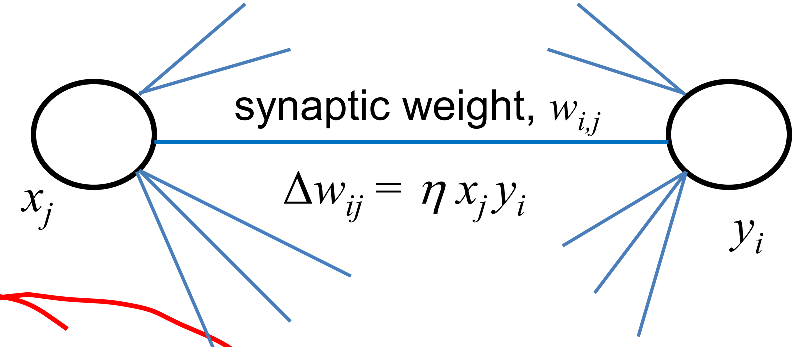
$$\mathrm{sgn}(x) = \begin{cases} 1 \;, x \geq 0 \\ -1 \,, x < 0 \end{cases}$$



- Hebbian learning (correlation learning, outer product)

$$\mathbf{W} = \mathbf{W}^1 + \mathbf{W}^2 + \ldots + \mathbf{W}^m$$

$$\mathbf{W}^k = [w_{ij}] = [x_j^k \; y_i^k] \quad \textit{(outer product)}$$



$$\vec{y}_p(\vec{x}_p^{\mathrm{T}} \vec{x}_p) + \sum_{p \neq k}^{m} \vec{y}_k (\vec{x}_p^{\mathrm{T}} \vec{x}_k)$$

crosstalk

Hebbian learning for associative memory

- Autoassociative case

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

$$\operatorname{sgn}(\mathbf{W}\vec{x}) = \vec{x}, \quad \operatorname{sgn}(\mathbf{XW}) = \mathbf{X}$$

Essentially, $\vec{x}$ are the eigenvectors of nonlinear sgn operation so the idea is to find $\mathbf{W}$ for which sgn($\mathbf{XW}$) has these patterns as eigenvectors, <u>but we do not want $\mathbf{W} = \mathbf{I}$ as a trivial solution of $\operatorname{sgn}(\mathbf{XW}) = \mathbf{X}$</u>

Hebbian learning for associative memory

- Autoassociative case

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

$$\mathrm{sgn}(\mathbf{W}\vec{x}) = \vec{x}, \quad \mathrm{sgn}(\mathbf{X}\mathbf{W}) = \mathbf{X}$$

Essentially, $\vec{x}$ are the eigenvectors of nonlinear sgn operation so the idea is to find $\mathbf{W}$ for which sgn($\mathbf{XW}$) has these patterns as eigenvectors,
<u>but we do not want $\mathbf{W}$ = $\mathbf{I}$ as a trivial solution of $\mathrm{sgn}(\mathbf{XW}) = \mathbf{X}$</u>

$$\text{for } \mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}, \; \mathrm{sgn}(\mathbf{X}\mathbf{W}) = \mathrm{sgn}(\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{X})$$

Hebbian learning for associative memory

- Autoassociative case

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

$$\operatorname{sgn}(\mathbf{W}\vec{x}) = \vec{x}, \quad \operatorname{sgn}(\mathbf{XW}) = \mathbf{X}$$

Essentially, $\vec{x}$ are the eigenvectors of nonlinear sgn operation so the idea is to find $\mathbf{W}$ for which sgn($\mathbf{XW}$) has these patterns as eigenvectors,
<u>but we do not want $\mathbf{W}$ = $\mathbf{I}$ as a trivial solution of $\operatorname{sgn}(\mathbf{XW}) = \mathbf{X}$</u>

$$\text{for } \mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}, \ \operatorname{sgn}(\mathbf{XW}) = \operatorname{sgn}(\mathbf{X}\underbrace{\mathbf{X}^{\mathrm{T}}\mathbf{X}})$$

For orthogonal $\mathbf{X}$ (or nearly),
$\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is a scaled identity $\mathbf{I}$ matrix

Hebbian learning for associative memory

- Autoassociative case

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

$$\operatorname{sgn}(\mathbf{W}\vec{x}) = \vec{x}, \quad \operatorname{sgn}(\mathbf{X}\mathbf{W}) = \mathbf{X}$$

Essentially, $\vec{x}$ are the eigenvectors of nonlinear sgn operation so the idea is to find $\mathbf{W}$ for which sgn($\mathbf{XW}$) has these patterns as eigenvectors,

<u>but we do not want $\mathbf{W}$ = $\mathbf{I}$ as a trivial solution of $\operatorname{sgn}(\mathbf{X}\mathbf{W}) = \mathbf{X}$</u>

$$\text{for } \mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}, \ \operatorname{sgn}(\mathbf{X}\mathbf{W}) = \operatorname{sgn}(\mathbf{X}\underbrace{\mathbf{X}^{\mathrm{T}}\mathbf{X}}) = \operatorname{sgn}(\mathbf{X}) = \mathbf{X}$$

For orthogonal $\mathbf{X}$ (or nearly),
$\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is a scaled identity $\mathbf{I}$ matrix

Hebbian learning for associative memory

- Autoassociative case

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

$$\mathrm{sgn}(\mathbf{W}\vec{x}) = \vec{x}, \quad \mathrm{sgn}(\mathbf{X}\mathbf{W}) = \mathbf{X}$$

Essentially, $\vec{x}$ are the eigenvectors of nonlinear sgn operation so the idea is to find $\mathbf{W}$ for which sgn($\mathbf{XW}$) has these patterns as eigenvectors

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

From a geometrical perspective:

$\mathbf{W}$ describes *non-orthogonal* projection on the subspace spanned by $\vec{x}$.

Pseudoinverse-based learning

- For a linear associator $\mathbf{XW} = \mathbf{Y}$

If **W** is rectangular, we are looking to minimise $\left\|\mathbf{XW} - \mathbf{Y}\right\|$

Pseudoinverse:

$$\mathbf{X}^{+} = \min_{\mathbf{W}} \left\|\mathbf{XW} - \mathbf{Y}\right\|$$

Pseudoinverse-based learning

- For a linear associator $\mathbf{XW} = \mathbf{Y}$

  If **W** is rectangular, we are looking to minimise $\left\|\mathbf{XW} - \mathbf{Y}\right\|$

  Pseudoinverse:

$$\mathbf{X}^{+} = \min_{\mathbf{W}} \left\|\mathbf{XW} - \mathbf{Y}\right\|$$

- Pseudoinverse for learning $\mathbf{W}$

$$\mathbf{W} = \mathbf{X}^{+}\mathbf{Y} \quad \Rightarrow \min \mathbf{E} = \left\|\mathbf{XW} - \mathbf{Y}\right\|$$

  Minimising E implies minimisation of the error (deviation from perfect recall): $\min \sum_{i} \left\|\mathbf{W}\vec{x}_i - \vec{y}_i\right\|^2$

Pseudoinverse-based learning

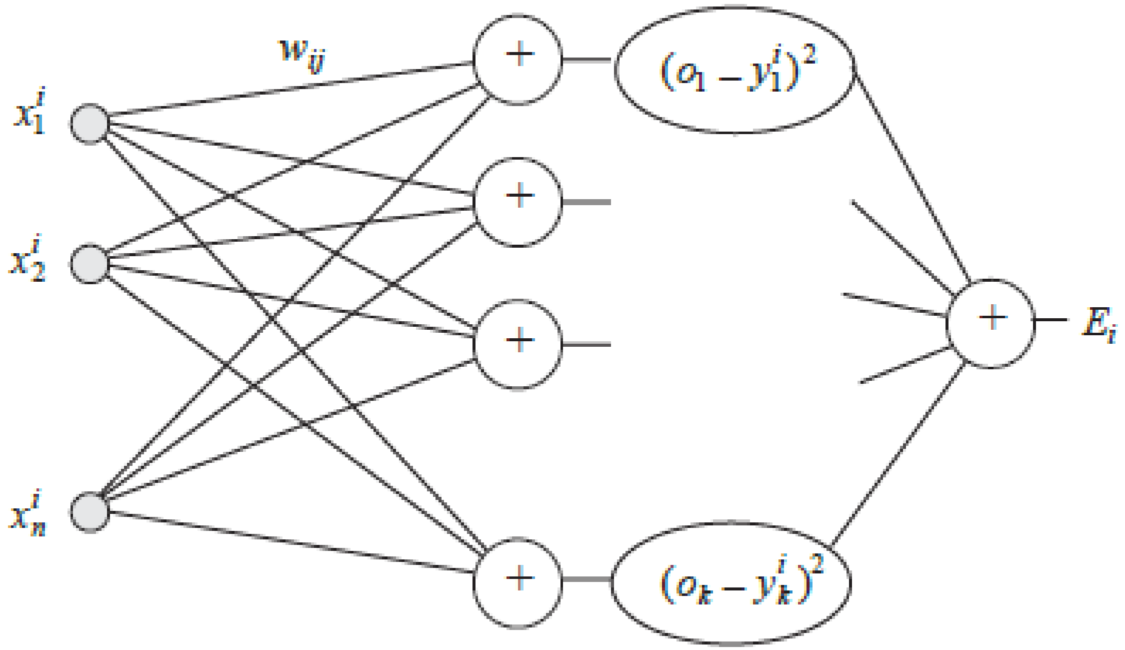
- For a linear associator $\mathbf{XW} = \mathbf{Y}$

If **W** is rectangular, we are looking to minimise $\left\| \mathbf{XW} - \mathbf{Y} \right\|$

Pseudoinverse:

$$\mathbf{X}^{+} = \min_{\mathbf{W}} \left\| \mathbf{XW} - \mathbf{I} \right\| \Rightarrow \mathbf{W} = \mathbf{X}^{+}\mathbf{Y} = \min_{\arg \mathbf{W}} \left\{ \left\| \mathbf{XW} - \mathbf{Y} \right\| \right\}$$

Pseudoinverse-based learning



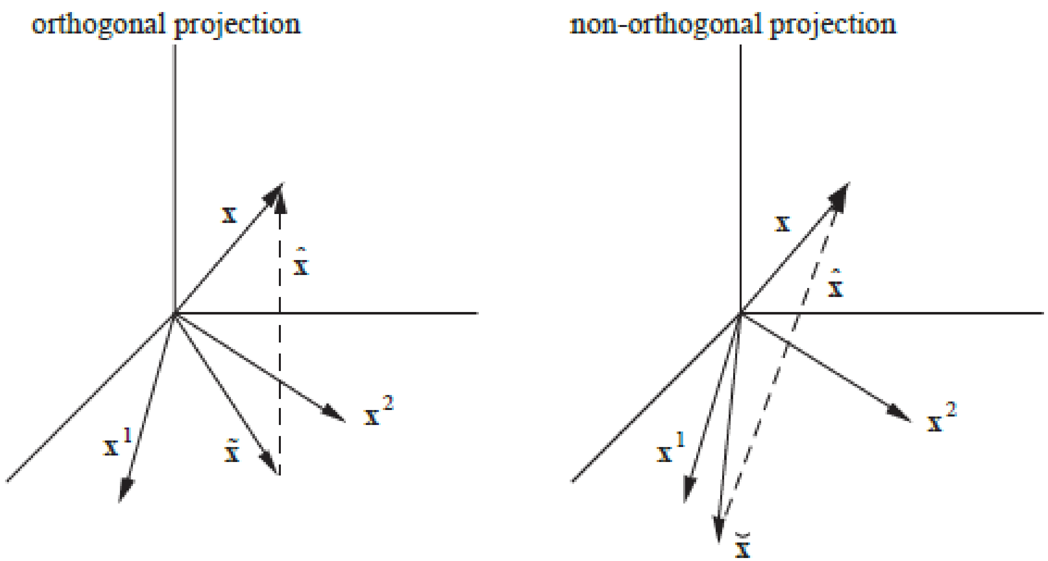
One way to estimate the pseudoinverse is by means of generalized delta rule

Pseudoinverse-based learning

Geometrical interpretation:

$\mathbf{W} = \mathbf{X}^{+}\mathbf{X}$ represents an orthogonal projection on the space spanned by vectors $\vec{x}_i$ that constitute $\mathbf{X}$.

Pseudoinverse-based learning



$\vec{x}_P \mathbf{X}^{+}\mathbf{X}$ gives the projection closest to memory pattern with lowest error deviation (in Euclidean and Hamming sense)

Learning for associative memory networks

Hebbian (outer product) vs pseudoinverse matrix approach

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X} \quad vs \quad \mathbf{W} = \mathbf{X}^{+}\mathbf{Y}$$

Learning for associative memory networks

Hebbian (outer product) vs pseudoinverse matrix approach

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X} \quad vs \quad \mathbf{W} = \mathbf{X}^{+}\mathbf{Y}$$

Fast computations and direct biological interpretation but non-orthogonal projection causing memory recall problems

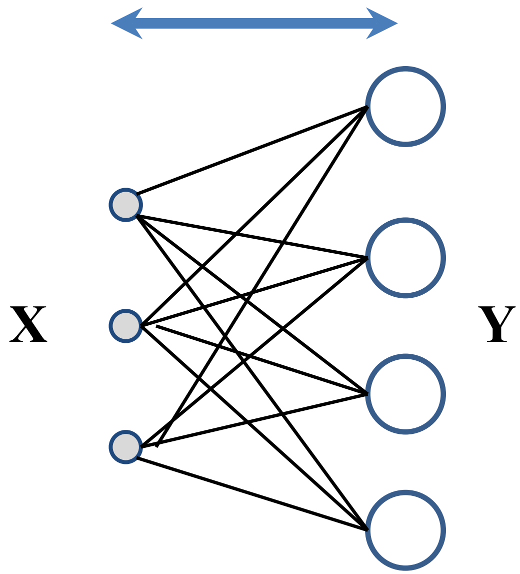
Better reliability and storage capacity (less problems with crosstalk) with orthogonal projections mitigating crosstalk problems when recalling memories

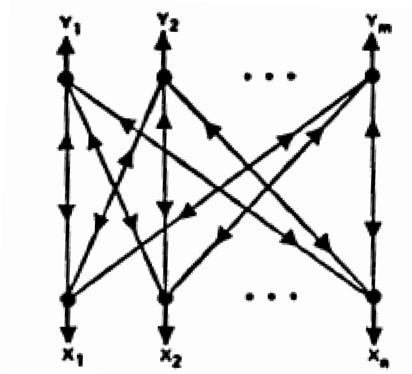
Bidirectional associative memory (resonance)

Builds on the concept of memory networks with feedback (recursive)

- bipolar {-1, 1} coding

- sign activation function





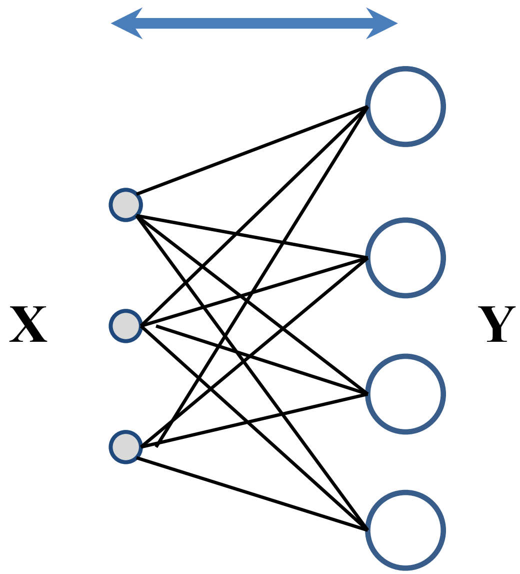
B. Kosko, 1988

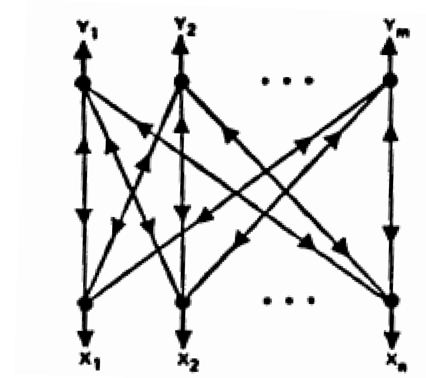
Bidirectional associative memory (resonance)

Builds on the concept of memory networks with feedback (recursive)

- bipolar {-1, 1} coding

- sign activation function





B. Kosko, 1988

Bidirectionality (feedback) imposes extra challenges

- synchronous vs asynchronous update
- different properties depending on updating mode

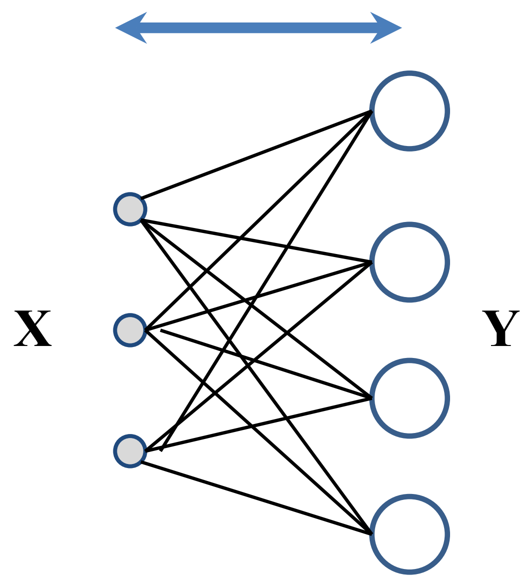
Bidirectional associative memory (resonance)

Builds on the concept of memory networks with feedback (recursive)

- bipolar {-1, 1} coding

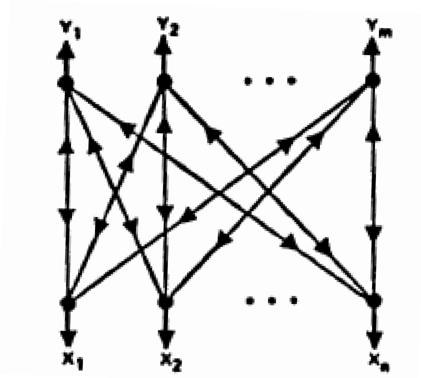
- sign activation function

$$\vec{y}(t) = \mathrm{sgn}(\mathbf{W}\vec{x}(t))$$
$$\vec{x}(t+1) = \mathrm{sgn}(\mathbf{W}\vec{y}(t))$$



Does it converge?

What are stable points?

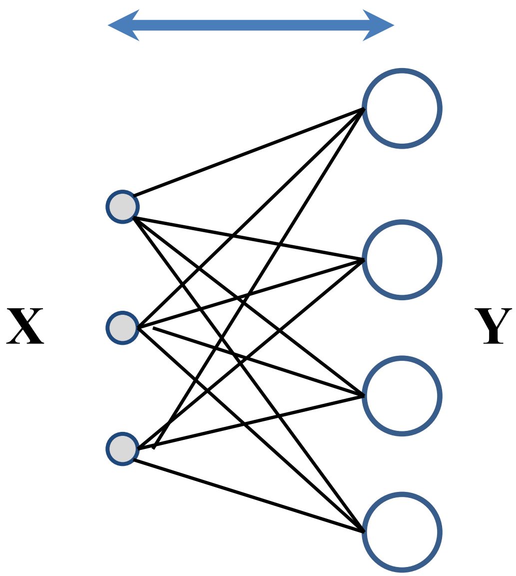


B. Kosko, 1988

Bidirectionality (feedback) imposes extra challenges

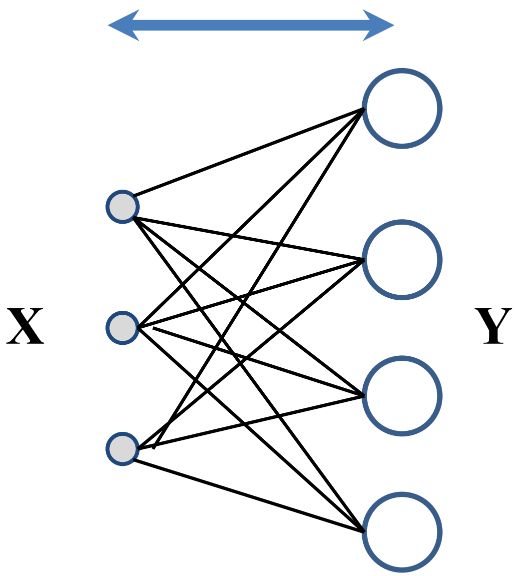
- synchronous vs asynchronous update
- different properties depending on updating mode

Concept of energy in BAM



If $(\vec{x}, \vec{y})$ is a stable point, then nearby points like $(\vec{x}_0, \vec{y}_0)$ should converge.

Concept of energy in BAM

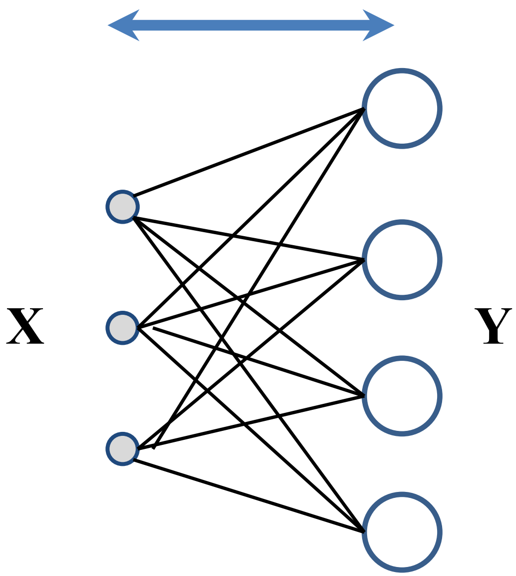


If $(\vec{x},\vec{y})$ is a stable point, then nearby points like $(\vec{x}_0,\vec{y}_0)$ should converge.

$$\vec{y}_0 = \mathbf{W}\vec{x}_0 \text{ , next } \quad \vec{e} = \mathbf{W}^{\mathrm{T}}\vec{y}_0$$

How far is $\vec{e}$ from $\vec{x}_0$?

Concept of energy in BAM



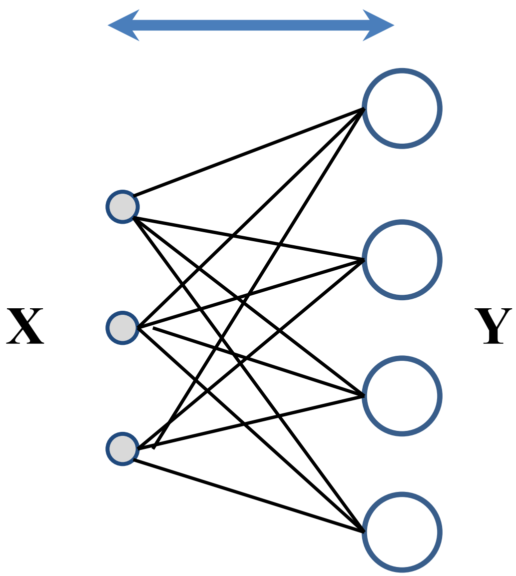
If $(\vec{x}, \vec{y})$ is a stable point, then nearby points like $(\vec{x}_0, \vec{y}_0)$ should converge.

$$\vec{y}_0 = \mathbf{W}\vec{x}_0 \text{ , next } \quad \vec{e} = \mathbf{W}^{\mathrm{T}}\vec{y}_0$$

How far is $\vec{e}$ from $\vec{x}_0$?

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

Concept of energy in BAM



If $(\vec{x}, \vec{y})$ is a stable point, then nearby points like $(\vec{x}_0, \vec{y}_0)$ should converge.

$$\vec{y}_0 = \mathbf{W}\vec{x}_0 \text{ , next } \quad \vec{e} = \mathbf{W}^{\mathrm{T}}\vec{y}_0$$

How far is $\vec{e}$ from $\vec{x}_0$?

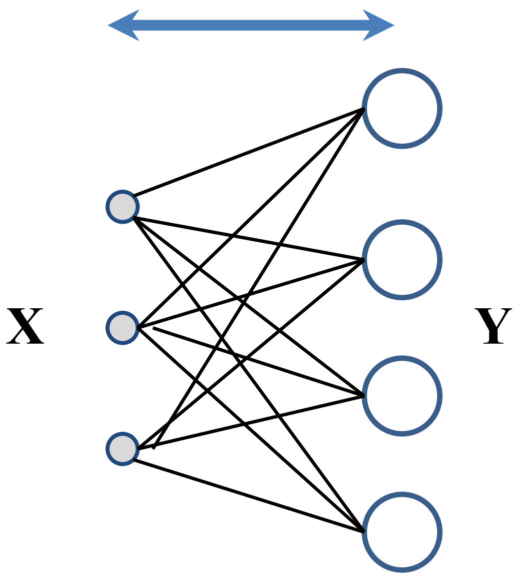
$$E = -\vec{x}_0^{\ \mathrm{T}}\vec{e} = -\vec{x}_0^{\ \mathrm{T}}\mathbf{W}^{\mathrm{T}}\vec{y}_0 = -\vec{y}_0^{\ \mathrm{T}}\mathbf{W}\vec{x}_0$$

For the autoassociative BAM with $\mathbf{W}$, energy in the state $\vec{x}$:

$$E(\vec{x}, \vec{x}) = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x}$$

$$E(\vec{x}) = -\frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}x_i x_j$$

Concept of energy in BAM



If $(\vec{x},\vec{y})$ is a stable point, then nearby points like $(\vec{x}_0,\vec{y}_0)$ should converge.

$$\vec{y}_0 = \mathbf{W}\vec{x}_0 \text{ , next } \quad \vec{e} = \mathbf{W}^{\mathrm{T}}\vec{y}_0$$

How far is $\vec{e}$ from $\vec{x}_0$?

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

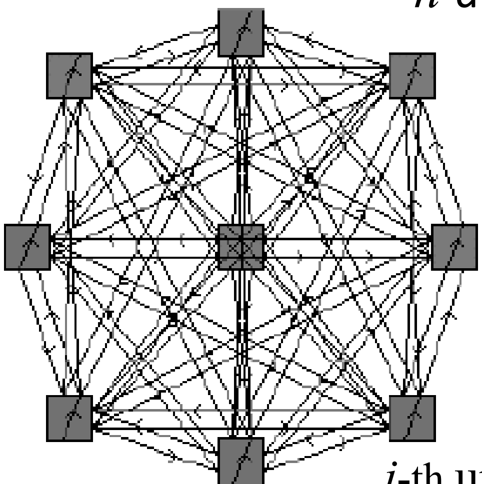
For the autoassociative BAM with $\mathbf{W}$, energy in the state $\vec{x}$:

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

If bias is added

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

Hopfield network

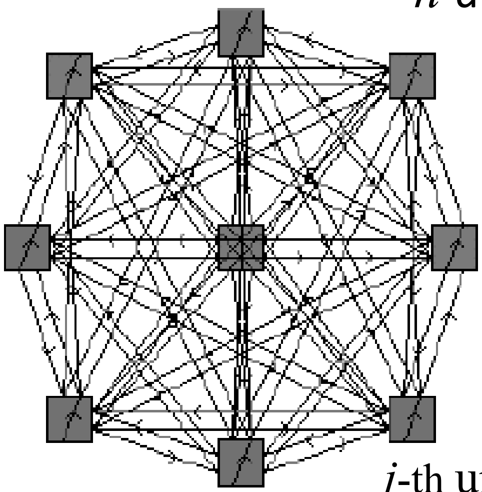


$$\forall_i \, w_{i,i} = 0$$ no self-connections

$$\vec{x}\,' = \operatorname{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

Hopfield network



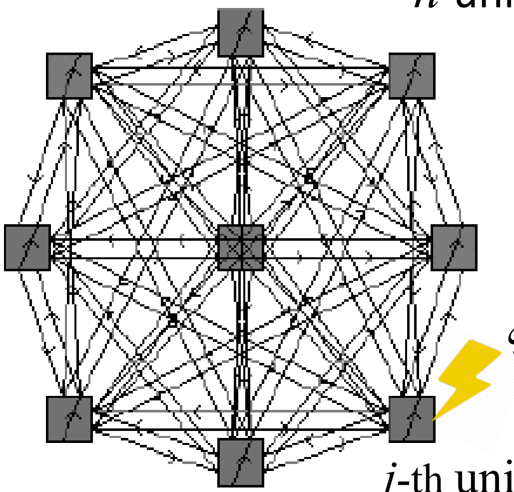
$$\forall_i \, w_{i,i} = 0$$ no self-connections

$$\vec{x}\,' = \operatorname{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

Iterative recall with asynchronous update

Hopfield network



$$\forall_i \, w_{i,i} = 0$$ no self-connections

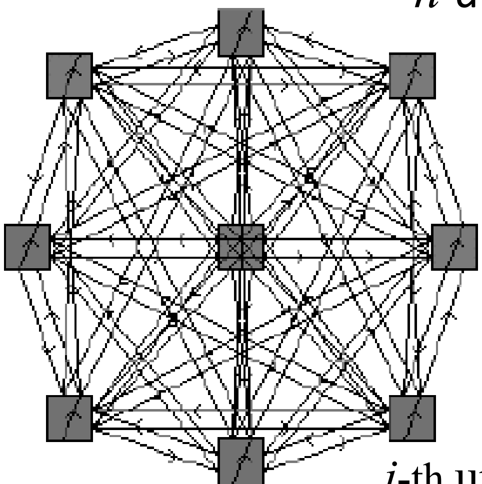
$$\vec{x}\,' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

Iterative recall with asynchronous update

1) Apply input probe $\boldsymbol{\xi}_p = [\xi_{1,p}\,,\ \xi_{2,p}\,, \ldots,\ \xi_{n,p}]$, i.e. $x_j(0) = \xi_{j,p}$

Hopfield network



$$\forall_i \, w_{i,i} = 0$$ no self-connections

$$\vec{x}' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

Iterative recall with asynchronous update

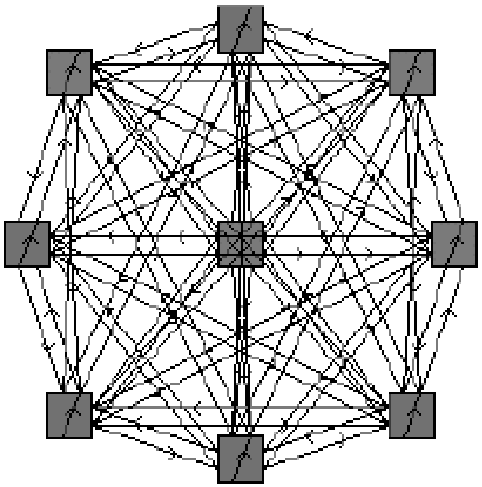
1) Apply input probe $\boldsymbol{\xi_p} = [\xi_{1,p}, \ \xi_{2,p}, \ldots, \ \xi_{n,p}]$, i.e. $x_j(0) = \xi_{j,p}$

2) Iterate *asynchronous* update until convergence (until the state $\boldsymbol{x}$ remains unchanged)

$$x_j(t+1) = \mathrm{sgn}\left(\sum_{i=1}^{n} w_{j,i}\, x_i(t)\right)$$

j=1,..,n is randomly selected one at a time

Hopfield network



$$\forall_i \, w_{i,i} = 0$$ no self-connections

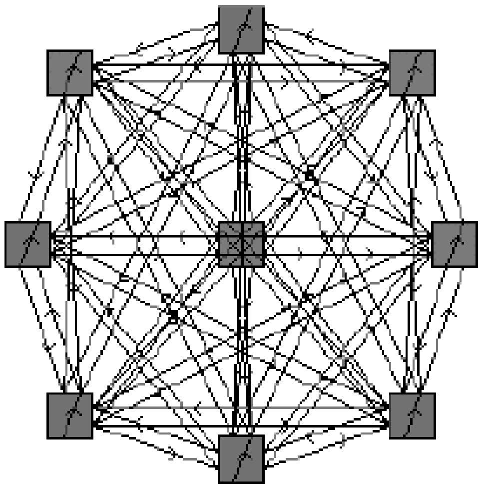
$$\vec{x}\,' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

Update occurs only when the state changes, so.....

$$\Delta E_{x_j \to x_j^*} = -\frac{1}{2}\left(\sum_i^n w_{i,j} x_i \, x_j^* - \sum_i^n w_{i,j} x_i x_j\right) = -\frac{1}{2}\left(x_j^* - x_j\right)\sum_i^n w_{i,j} x_i \le 0$$

Hopfield network



$$\forall_i \, w_{i,i} = 0$$ no self-connections

$$\vec{x}\,' = \operatorname{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

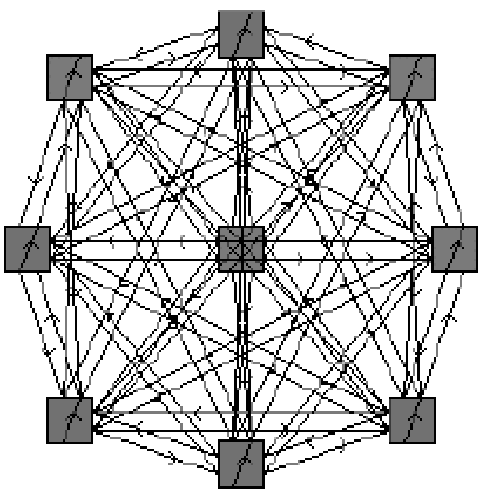
W should be symmetric with diag=0 for convergence

Update occurs only when the state changes, so.....

$$\Delta E_{x_j \to x_j^*} = -\frac{1}{2}\left(\sum_i^n w_{i,j} x_i x_j^* - \sum_i^n w_{i,j} x_i x_j\right) = -\frac{1}{2}\left(x_j^* - x_j\right)\sum_i^n w_{i,j} x_i \le 0$$

towards lower energy – convergence!

Hopfield network



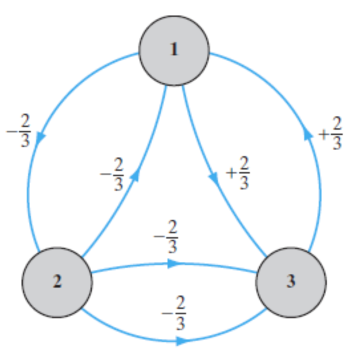
$$\forall_i \, w_{i,i} = 0$$ no self-connections

$$\vec{x}\,' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

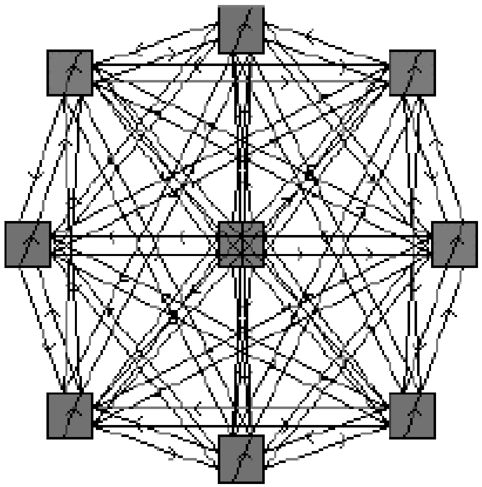
$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j + \sum_{i=1}^{n}\theta_i x_i$$

$\mathbf{W}$ should be symmetric with diag=0 for convergence

How many states are candidates for fixed states?



Hopfield network

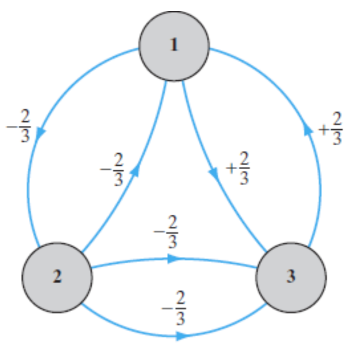


$$\forall_i \, w_{i,i} = 0 \qquad \text{no self-connections}$$

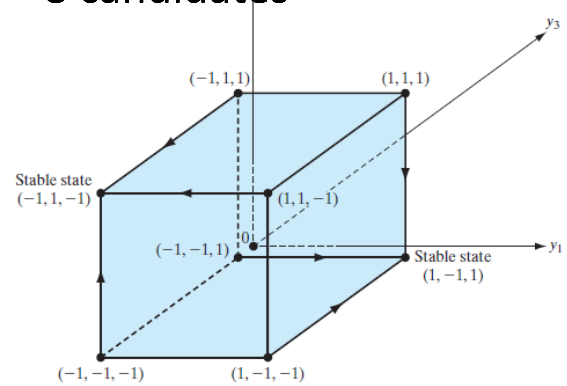
$$\vec{x}\,' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

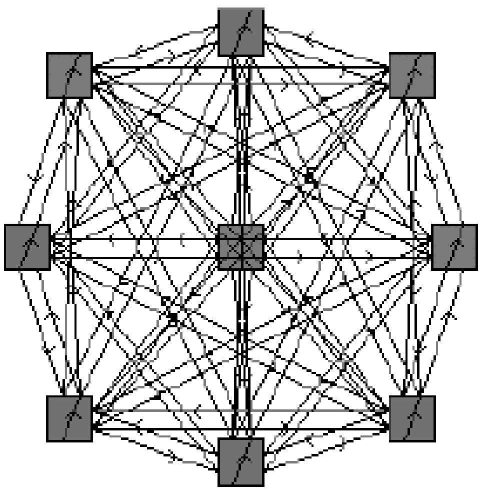
$\mathbf{W}$ should be symmetric with diag=0 for convergence



8 candidates



Hopfield network

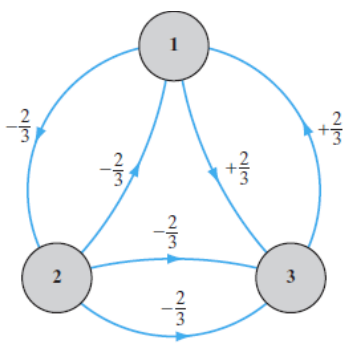


$$\forall_i \, w_{i,i} = 0$$ no self-connections

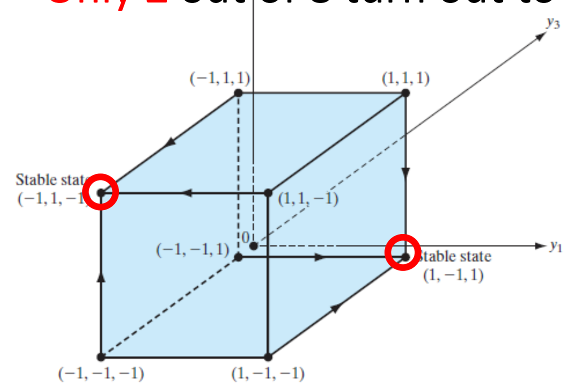
$$\vec{x}' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

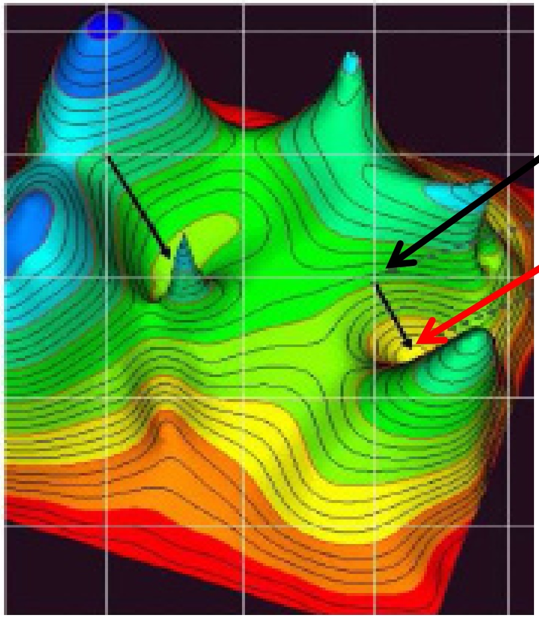
$\mathbf{W}$ should be symmetric with diag=0 for convergence



Only **2** out of 8 turn out to be stable!



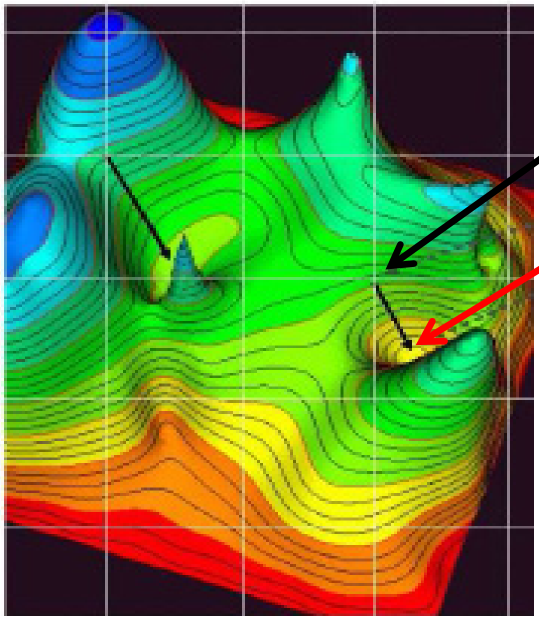
Attractor dynamics



Memory cue
(within the basin of attractor)

Memory state
(local energy minimum, stable point, attractor)

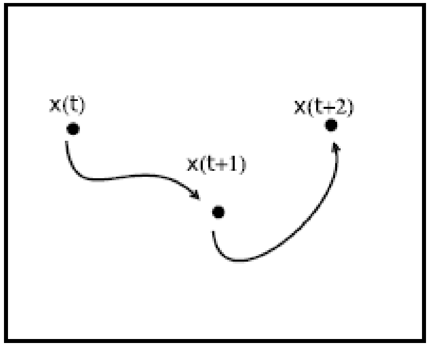
Attractor dynamics



Memory cue
(within the basin of attractor)

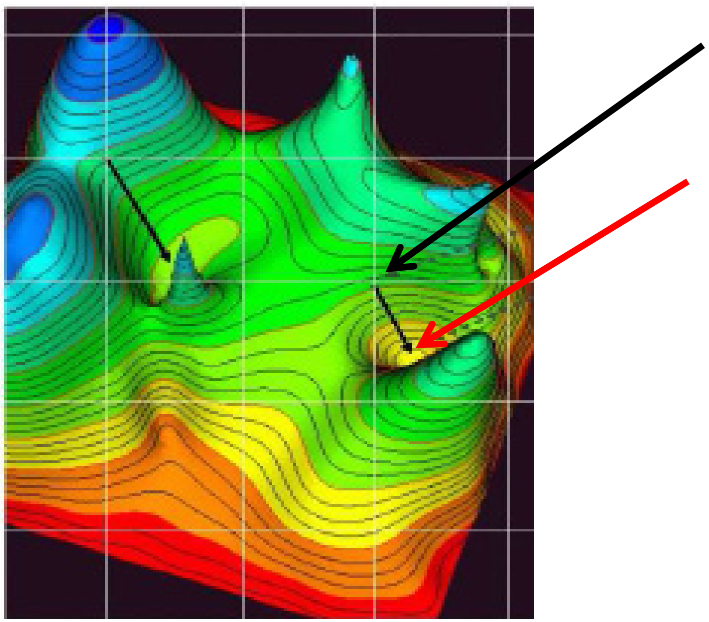
Memory state
(local energy minimum,
stable point, **fixed-point attractor**)

Dynamics travelling in the energy landscape and attracted to the <u>energy minimum</u>



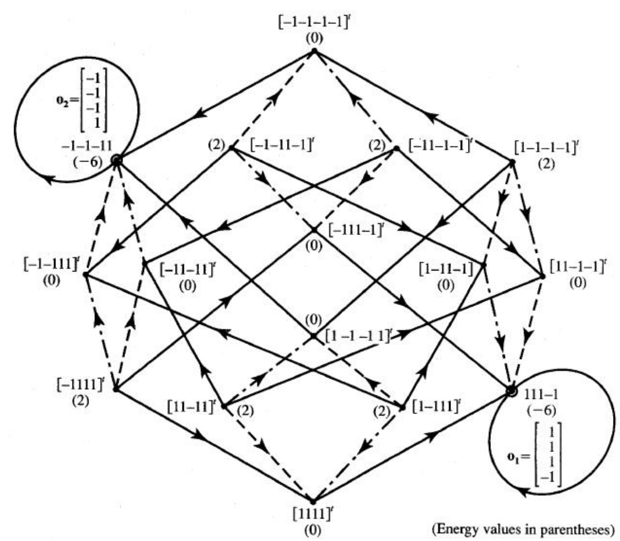
In *discrete* Hopfield network,
the energy landscape is discrete!

Attractor dynamics



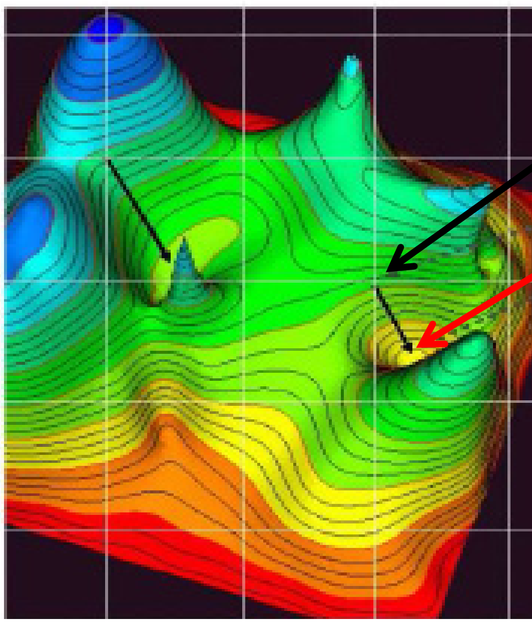
Memory cue
(within the basin of attractor)

Memory state
(local energy minimum,
stable point, **fixed-point attractor**)



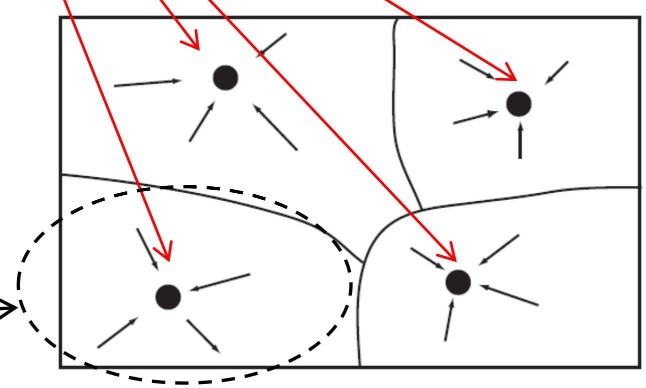
In *discrete* Hopfield network,
the energy landscape is discrete!

Attractor dynamics



Memory cue
(within the basin of attractor)

Memory state
(local energy minimum,
stable point, **fixed-point attractor**)



Around each fixed point (attractor), there is a ***basin of attraction***

How do we learn memories for storage?

Hopfield network as a content addressable memory

A set of memory patterns $\{\xi_1, \xi_2, \ldots \xi_M\}$ to be learnt.

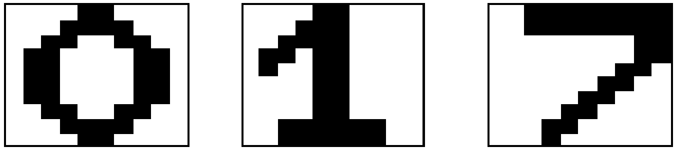
$$\boldsymbol{\xi_k} = [\xi_{k,1}\ ,\ \xi_{k,2}\ , \ldots\ \xi_{k,n}\ ],\ \ k=1,.., M$$

Outer product rule (Hebbian-like learning) is used to compute **W**:

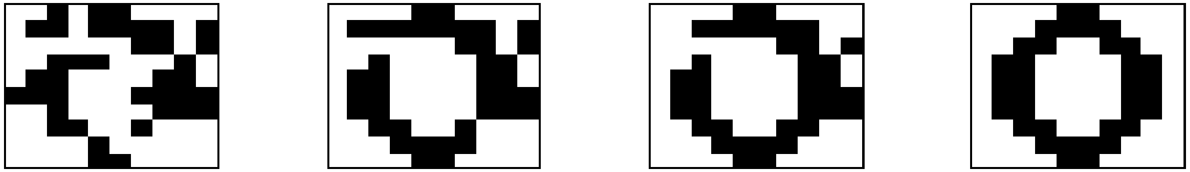
$$w_{j,i} = \begin{cases} \dfrac{1}{n}\displaystyle\sum_{k=1}^{M} \xi_{k,j} \cdot \xi_{k,i}, & j \neq i \\ 0, & j = i \end{cases}$$

Pattern storage and recall example

- The following patterns ξ^1, ξ^2, ξ^3 were stored in the weight matrix W:

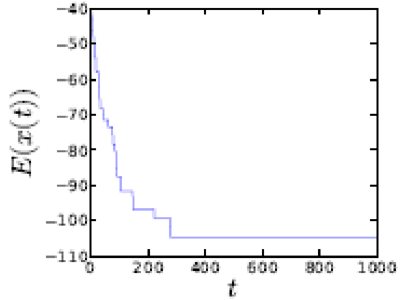


- Four snapshots of the state evolution $x(t)$:



$t = 0$ $t = 50$ $t = 100$ $t = 300$

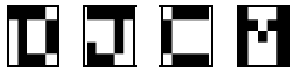
- Evolution of the energy $E(x(t))$:



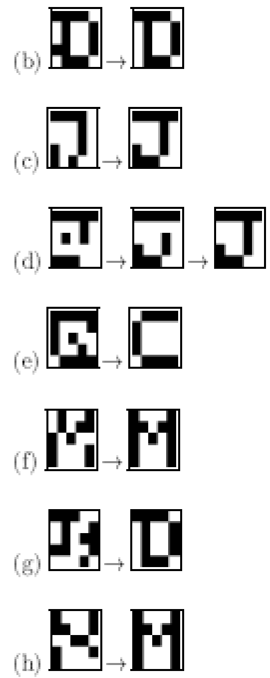
adapted from L. Busing (TU Graz)

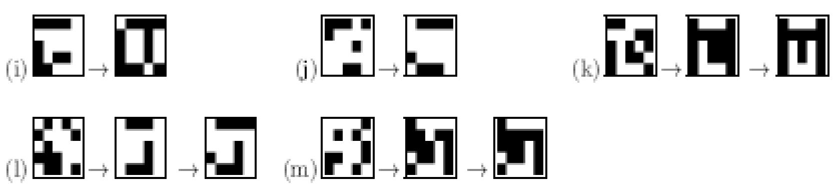
Pattern storage and recall example

(a)



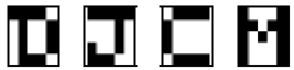
.	0	0	0	0	-2	2	-2	2	2	-2	0	0	0	2	0	0	-2	0	2	2	0	0	-2	-2
0	.	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	.	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	4	.	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	0	0	0	.	2	-2	-2	2	-2	2	-4	0	0	-2	4	-4	-2	0	-2	2	0	0	-2	2
-2	-2	-2	-2	2	.	0	0	0	0	4	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	4
2	-2	-2	-2	-2	0	.	0	0	4	0	2	2	-2	4	-2	2	0	-2	4	0	-2	-2	0	0
-2	-2	-2	-2	-2	0	0	.	0	0	0	2	2	2	0	-2	2	4	2	0	0	-2	-2	0	0
2	-2	-2	-2	2	0	0	0	.	0	0	-2	2	2	0	2	-2	0	2	0	4	-2	-2	-4	0
2	-2	-2	-2	-2	0	4	0	0	.	0	2	2	-2	4	-2	2	0	-2	4	0	-2	-2	0	0
-2	-2	-2	-2	2	4	0	0	0	0	.	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	4
0	0	0	0	-4	-2	2	2	-2	2	-2	.	0	0	2	-4	4	2	0	2	-2	0	0	2	-2
0	-4	-4	-4	0	2	2	2	2	2	2	0	.	0	2	0	0	2	0	2	2	-4	-4	-2	2
0	0	0	0	0	-2	-2	2	2	-2	-2	0	0	.	-2	0	0	2	4	-2	2	0	0	-2	-2
2	-2	-2	-2	-2	0	4	0	0	4	0	2	2	-2	.	-2	2	0	-2	4	0	-2	-2	0	0
0	0	0	0	4	2	-2	-2	2	-2	2	-4	0	0	-2	.	-4	-2	0	-2	2	0	0	-2	2
0	0	0	0	-4	-2	2	2	-2	2	-2	4	0	0	2	-4	.	2	0	2	-2	0	0	2	-2
-2	-2	-2	-2	-2	0	0	4	0	0	0	2	2	2	0	-2	2	.	2	0	0	-2	-2	0	0
0	0	0	0	0	-2	-2	2	2	-2	-2	0	0	4	-2	0	0	2	.	-2	2	0	0	-2	-2
2	-2	-2	-2	-2	0	4	0	0	4	0	2	2	-2	4	-2	2	0	-2	.	0	-2	-2	0	0
2	-2	-2	-2	2	0	0	0	4	0	0	-2	2	2	0	2	-2	0	2	0	.	-2	-2	-4	0
0	4	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	.	4	2	-2
0	4	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	.	2	-2
-2	2	2	2	-2	0	0	0	-4	0	0	2	-2	-2	0	-2	2	0	-2	0	-4	2	2	.	0
-2	-2	-2	-2	2	4	0	0	0	0	4	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	.



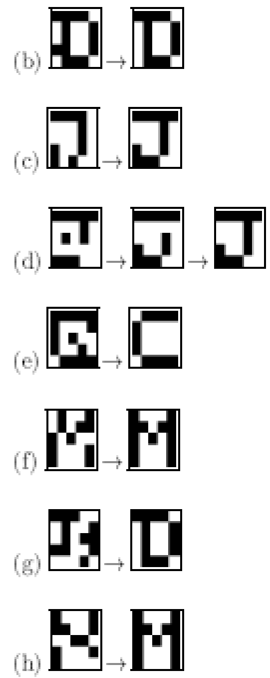


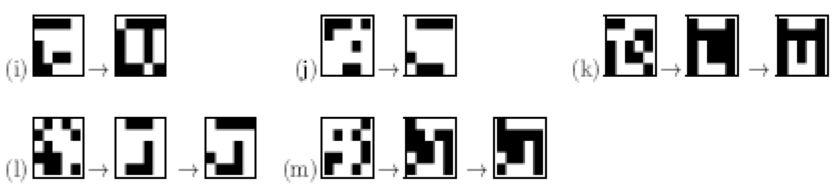
adapted from McKay

Pattern storage and recall example

(a) 

.	0	0	0	0	-2	2	-2	2	2	-2	0	0	0	2	0	0	-2	0	2	2	0	0	-2	-2
0	.	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	.	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	4	.	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	0	0	0	.	2	-2	-2	2	-2	2	-4	0	0	-2	4	-4	-2	0	-2	2	0	0	-2	2
-2	-2	-2	-2	2	.	0	0	0	0	4	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	4
2	-2	-2	-2	-2	0	.	0	0	4	0	2	2	-2	4	-2	2	0	-2	4	0	-2	-2	0	0
-2	-2	-2	-2	-2	0	0	.	0	0	0	2	2	2	0	-2	2	4	2	0	0	-2	-2	0	0
2	-2	-2	-2	2	0	0	0	.	0	0	-2	2	2	0	2	-2	0	2	0	4	-2	-2	-4	0
2	-2	-2	-2	-2	0	4	0	0	.	0	2	2	-2	4	-2	2	0	-2	4	0	-2	-2	0	0
-2	-2	-2	-2	2	4	0	0	0	0	.	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	4
0	0	0	0	-4	-2	2	2	-2	2	-2	.	0	0	2	-4	4	2	0	2	-2	0	0	2	-2
0	-4	-4	-4	0	2	2	2	2	2	2	0	.	0	2	0	0	2	0	2	2	-4	-4	-2	2
0	0	0	0	0	-2	-2	2	2	-2	-2	0	0	.	-2	0	0	2	4	-2	2	0	0	-2	-2
2	-2	-2	-2	-2	0	4	0	0	4	0	2	2	-2	.	-2	2	0	-2	4	0	-2	-2	0	0
0	0	0	0	4	2	-2	-2	2	-2	2	-4	0	0	-2	.	-4	-2	0	-2	2	0	0	-2	2
0	0	0	0	-4	-2	2	2	-2	2	-2	4	0	0	2	-4	.	2	0	2	-2	0	0	2	-2
-2	-2	-2	-2	-2	0	0	4	0	0	0	2	2	2	0	-2	2	.	2	0	0	-2	-2	0	0
0	0	0	0	0	-2	-2	2	2	-2	-2	0	0	4	-2	0	0	2	.	-2	2	0	0	-2	-2
2	-2	-2	-2	-2	0	4	0	0	4	0	2	2	-2	4	-2	2	0	-2	.	0	-2	-2	0	0
2	-2	-2	-2	2	0	0	0	4	0	0	-2	2	2	0	2	-2	0	2	0	.	-2	-2	-4	0
0	4	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	.	4	2	-2
0	4	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	.	2	-2
-2	2	2	2	-2	0	0	0	-4	0	0	2	-2	-2	0	-2	2	0	-2	0	-4	2	2	.	0
-2	-2	-2	-2	2	4	0	0	0	0	4	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	.



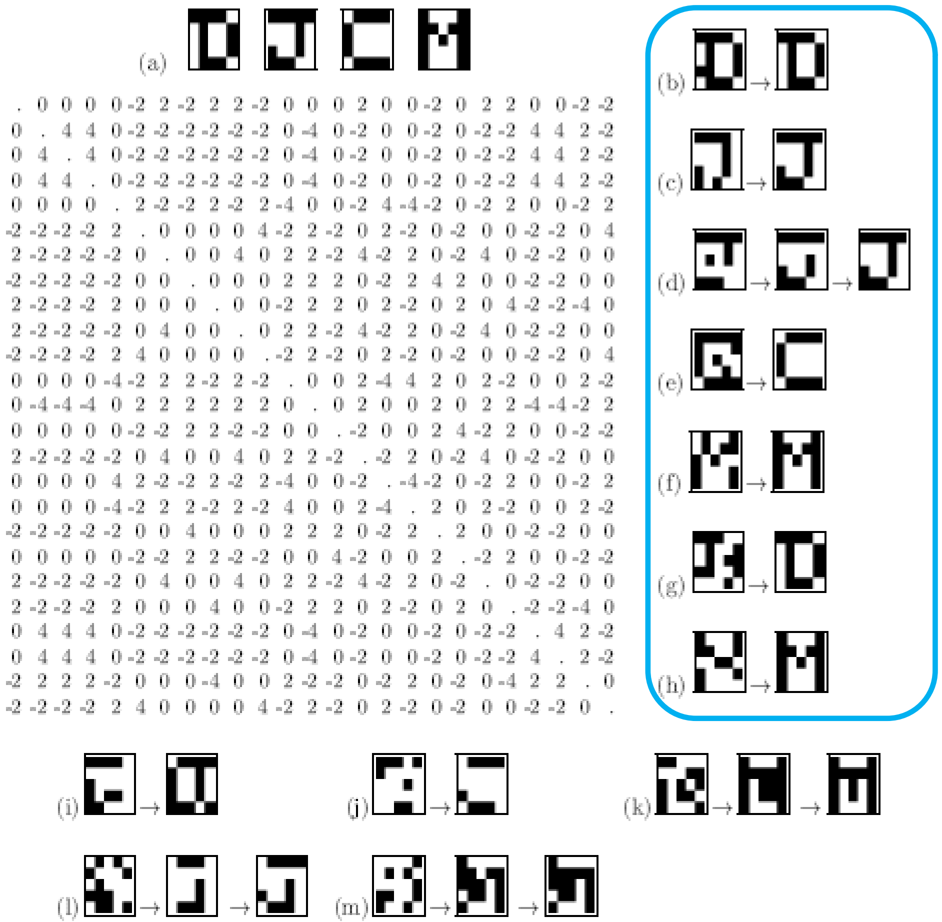


Common problems

1. Corruption of individual bits.
2. Lack of encoded memory or a very small basin of attraction.
3. Appearance of spurious additional memories.

adapted from McKay

Pattern storage and recall example

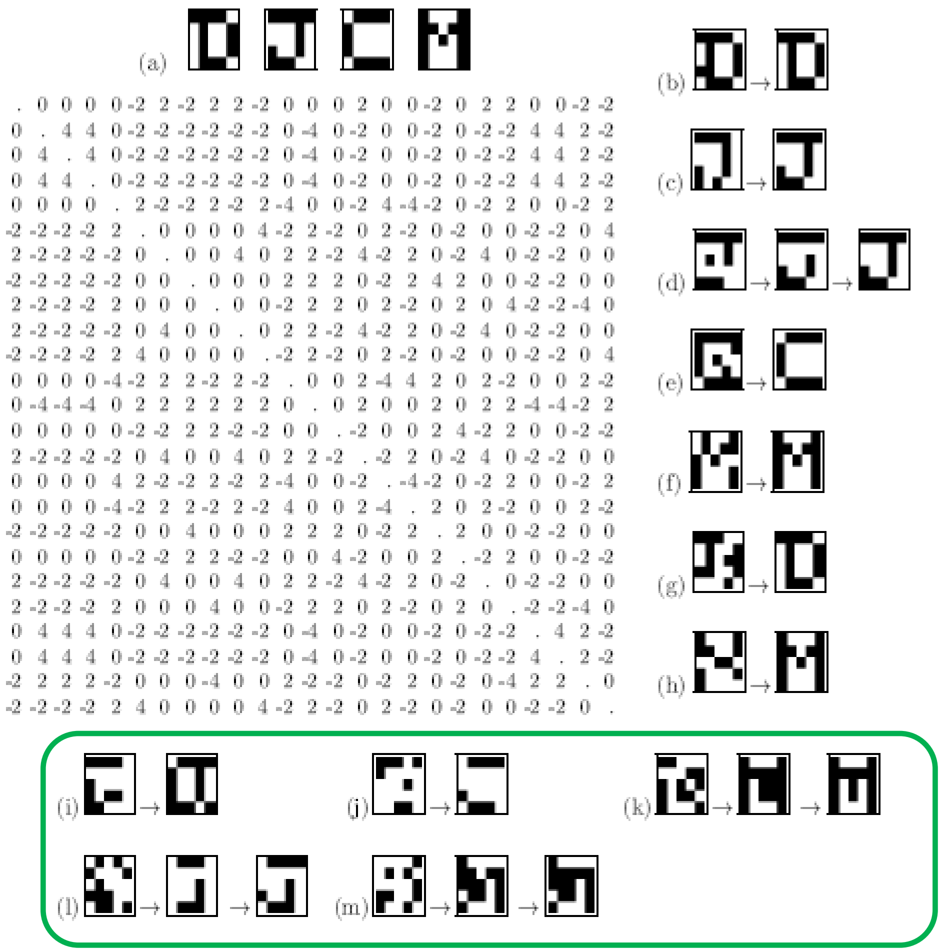


Common problems

1. Corruption of individual bits.
2. Lack of encoded memory or a very small basin of attraction.
3. Appearance of spurious additional memories.

adapted from McKay

Pattern storage and recall example



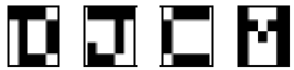
Common problems

1. Corruption of individual bits.
2. Lack of encoded memory or a very small basin of attraction.
3. Appearance of spurious additional memories.

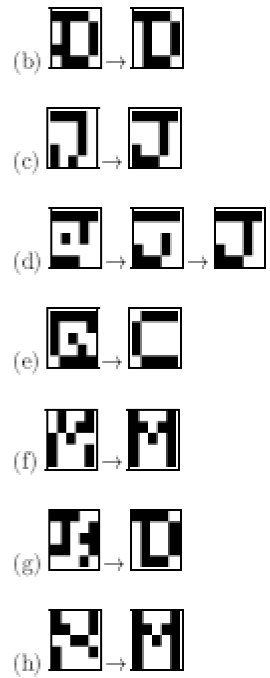
Spurious states often arise out of degenerate eigenvectors.

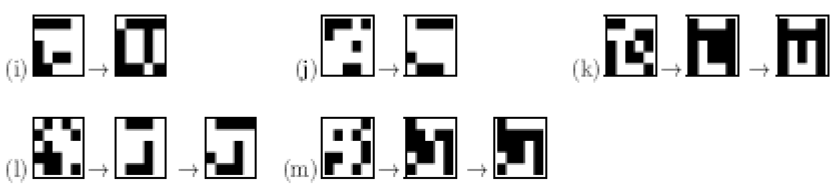
adapted from McKay

Pattern storage and recall example



.	0	0	0	0	-2	2	-2	2	2	-2	0	0	0	2	0	0	-2	0	2	2	0	0	-2	-2
0	.	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	.	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	4	.	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	0	0	0	.	2	-2	-2	2	-2	2	-4	0	0	-2	4	-4	-2	0	-2	2	0	0	-2	2
-2	-2	-2	-2	2	.	0	0	0	0	4	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	4
2	-2	-2	-2	-2	0	.	0	0	4	0	2	2	-2	4	-2	2	0	-2	4	0	-2	-2	0	0
-2	-2	-2	-2	-2	0	0	.	0	0	0	2	2	2	0	-2	2	4	2	0	0	-2	-2	0	0
2	-2	-2	-2	2	0	0	0	.	0	0	-2	2	2	0	2	-2	0	2	0	4	-2	-2	-4	0
2	-2	-2	-2	-2	0	4	0	0	.	0	2	2	-2	4	-2	2	0	-2	4	0	-2	-2	0	0
-2	-2	-2	-2	2	4	0	0	0	0	.	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	4
0	0	0	0	-4	-2	2	2	-2	2	-2	.	0	0	2	-4	4	2	0	2	-2	0	0	2	-2
0	-4	-4	-4	0	2	2	2	2	2	2	0	.	0	2	0	0	2	0	2	2	-4	-4	-2	2
0	0	0	0	0	-2	-2	2	2	-2	-2	0	0	.	-2	0	0	2	4	-2	2	0	0	-2	-2
2	-2	-2	-2	-2	0	4	0	0	4	0	2	2	-2	.	-2	2	0	-2	4	0	-2	-2	0	0
0	0	0	0	4	2	-2	-2	2	-2	2	-4	0	0	-2	.	-4	-2	0	-2	2	0	0	-2	2
0	0	0	0	-4	-2	2	2	-2	2	-2	4	0	0	2	-4	.	2	0	2	-2	0	0	2	-2
-2	-2	-2	-2	-2	0	0	4	0	0	0	2	2	2	0	-2	2	.	2	0	0	-2	-2	0	0
0	0	0	0	0	-2	-2	2	2	-2	-2	0	0	4	-2	0	0	2	.	-2	2	0	0	-2	-2
2	-2	-2	-2	-2	0	4	0	0	4	0	2	2	-2	4	-2	2	0	-2	.	0	-2	-2	0	0
2	-2	-2	-2	2	0	0	0	4	0	0	-2	2	2	0	2	-2	0	2	0	.	-2	-2	-4	0
0	4	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	.	4	2	-2
0	4	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	.	2	-2
-2	2	2	2	-2	0	0	0	-4	0	0	2	-2	-2	0	-2	2	0	-2	0	-4	2	2	.	0
-2	-2	-2	-2	2	4	0	0	0	0	4	-2	2	-2	0	2	-2	0	-2	0	0	-2	-2	0	.





Common problems

1. Corruption of individual bits.
2. Lack of encoded memory or a very small basin of attraction.
3. Appearance of spurious additional memories.

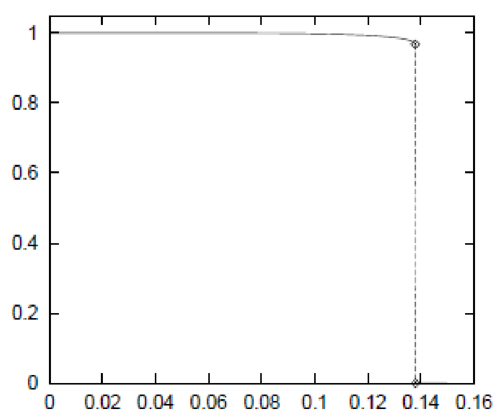
Generally, Hopfield network is robust to noise, data corruption and “brain damage” (zeroed subset of weights).

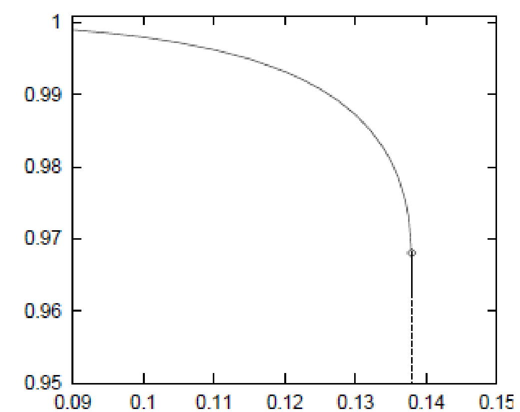
adapted from McKay

Memory capacity

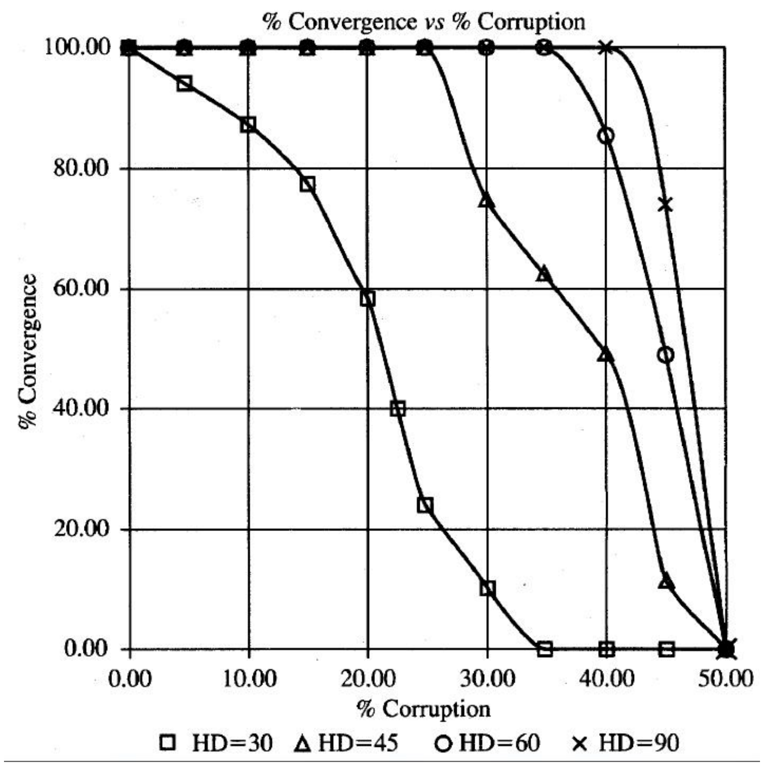
- Cross-talk between memory patterns is key to limited capacity
- Memory capacity is usually tested on independent random patterns
  - Hopfield network can store roughly M<=0.138 n of such random patterns (sharp discontinuity)
  - for large M/n, unstable bits may unfold into an avalanche effect
- To guarantee stability of all patterns with high probability, we must ensure

$$M \leq \frac{n}{4 \ln n}$$



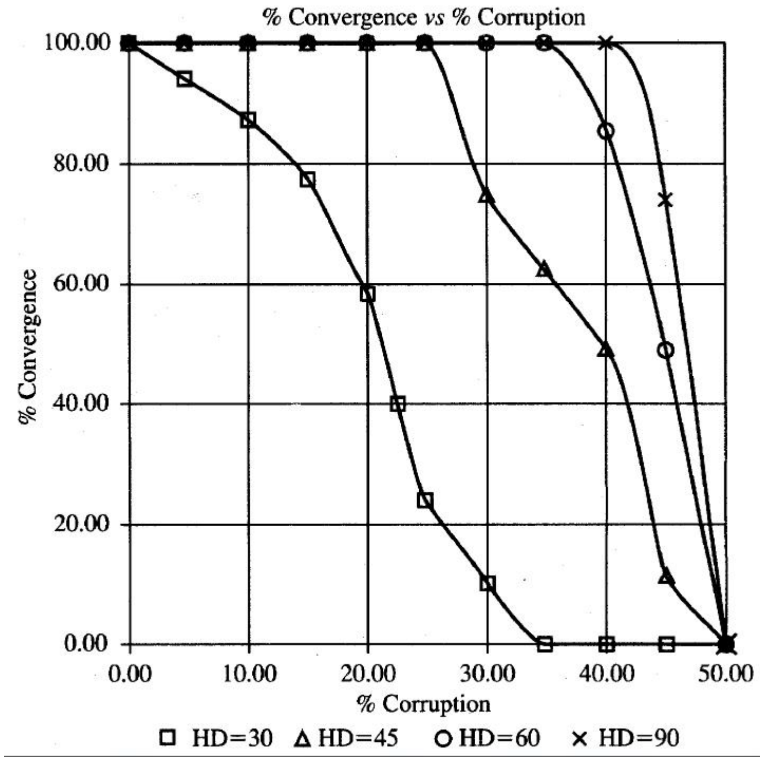


Catastrophic forgetting effect



Convergence rate is defined based on the convergence criterion, often expressed as the upper bound on *Hamming distance*.

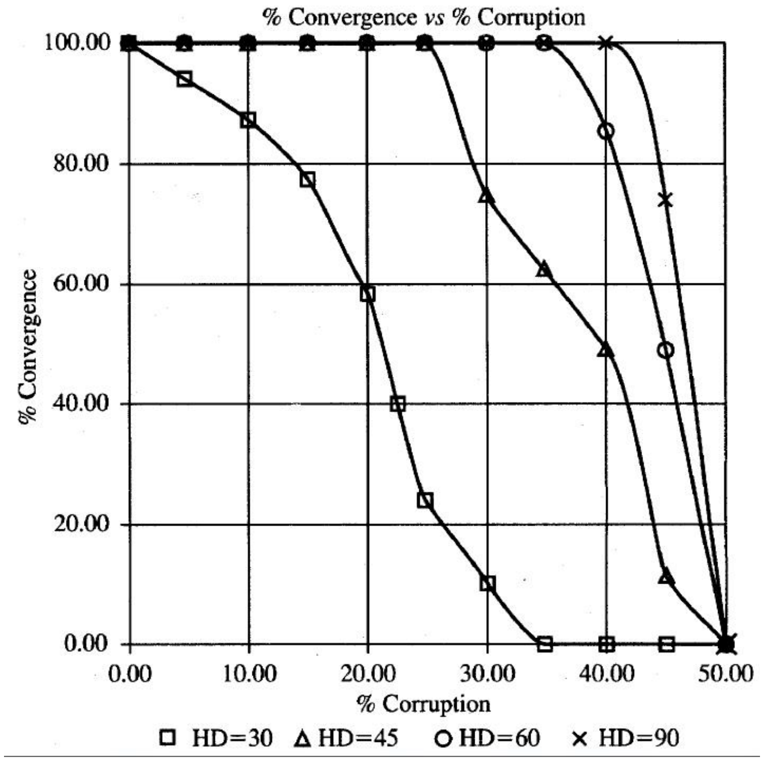
Catastrophic forgetting effect



Convergence rate is defined based on the convergence criterion, often expressed as the upper bound on *Hamming distance*.

Network properties are not robust for synchronous updates.

Catastrophic forgetting effect



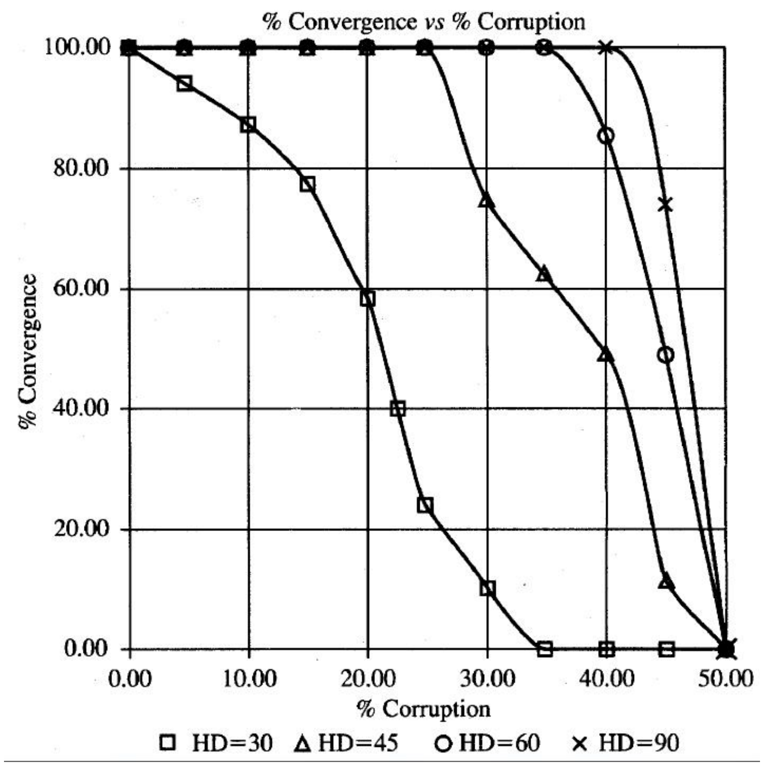
Convergence rate is defined based on the convergence criterion, often expressed as the upper bound on *Hamming distance*.

Network properties are not robust for synchronous updates.

Also, problems for continuous networks.

$$a_i = \sum_j w_{ij} x_j \qquad x_i = \tanh(a_i).$$

Catastrophic forgetting effect



Convergence rate is defined based on the convergence criterion, often expressed as the upper bound on *Hamming distance*.

Network properties are not robust for synchronous updates.

Also, problems for continuous networks.

$$a_i = \sum_j w_{ij} x_j \qquad x_i = \tanh(a_i).$$

Better behaviour for continuous continuous –time Hopfield network

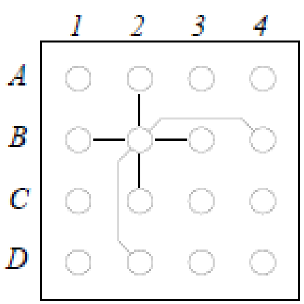
$$a_i(t) = \sum_j w_{ij} x_j(t). \qquad \frac{\mathrm{d}}{\mathrm{d}t} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)),$$

Hopfield networks for optimisation problems

- Hopfield network's dynamics minimises an energy function
- Some optimisation problems could be mapped to the quadratic energy function (particularly constrain satisfaction problems(CSPs))

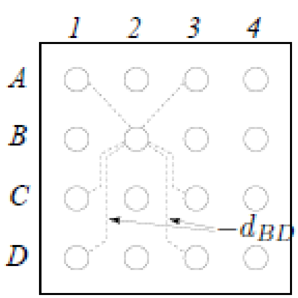
Hopfield networks for optimisation problems

- Hopfield network's dynamics minimises an energy function
- Some optimisation problems could be mapped to the quadratic energy function (particularly constrain satisfaction problems(CSPs))
- Travelling salesman problem (TSP) as a classic CSP problem



validity

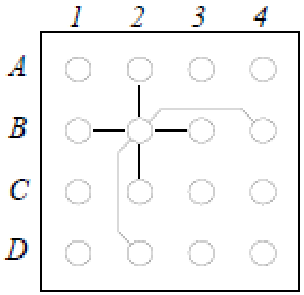
distances



$$E = \underbrace{\frac{1}{2}\sum_{i,j,k}^{n} d_{ij} x_{ik} x_{j,k+1}}_{\text{sum of distances}} + \underbrace{\frac{\gamma}{2}\Big(\sum_{j=1}^{n}\big(\sum_{i=1}^{n} x_{ij} - 1\big)^2 + \sum_{i=1}^{n}\big(\sum_{j=1}^{n} x_{ij} - 1\big)^2\Big)}_{\text{validity: single 1s in each column and row}}$$

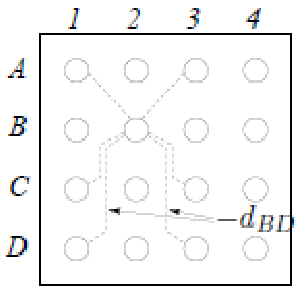
Hopfield networks for optimisation problems

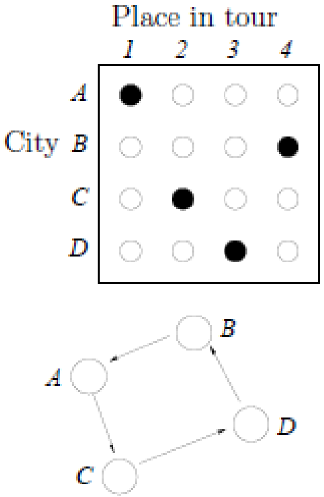
- Hopfield network's dynamics minimises an energy function
- Some optimisation problems could be mapped to the quadratic energy function (particularly constrain satisfaction problems(CSPs))
- Travelling salesman problem (TSP) as a classic CSP problem

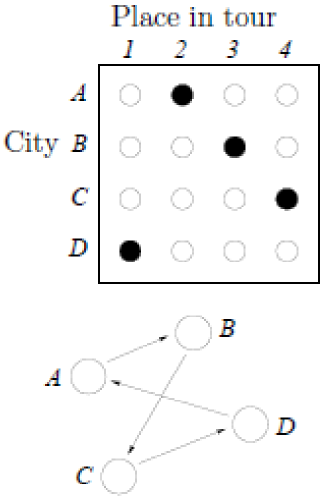


validity

distances







Hopfield networks

In summary

- Hopfield network is a nice model for memory with biological features including Hebbian learning
- It is a very simple, stable and mathematically tractable model
- It has limited capacity however
- It does not allow for storing time series
- The attractor dynamics is limited to fixed points

From Hopfield networks to Boltzmann machines

- Continuous Hopfield network

$$x_i = \frac{1}{1+e^{-a}} \quad \text{instead of} \quad x_i = \operatorname{sgn}(a)$$

- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases} \qquad p = \frac{1}{1+e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}$$

T is a positive temperature const.

From Hopfield networks to Boltzmann machines

- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases}$$

$$p = \frac{1}{1 + e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}$$

$$p(v) = \frac{1}{1 + e^{-v}} \quad \text{where} \quad v = \frac{1}{T}\sum_j w_{i,j} x_j$$

T controls the level of randomness

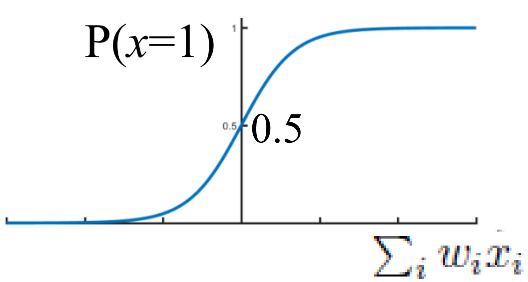
From Hopfield networks to Boltzmann machines

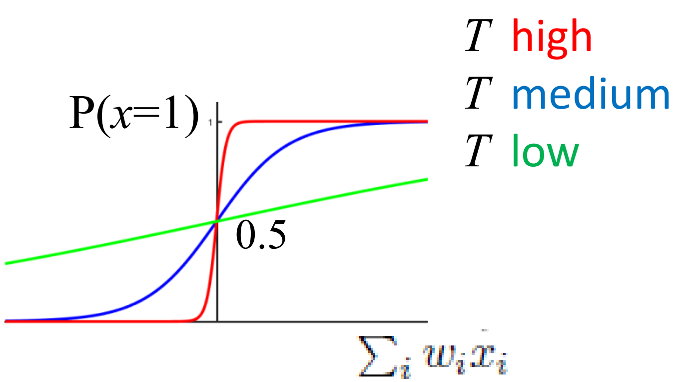
- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases}$$

$$p = \frac{1}{1 + e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}$$

$$p(v) = \frac{1}{1 + e^{-v}} \quad \text{where} \quad v = \frac{1}{T}\sum_j w_{i,j} x_j$$





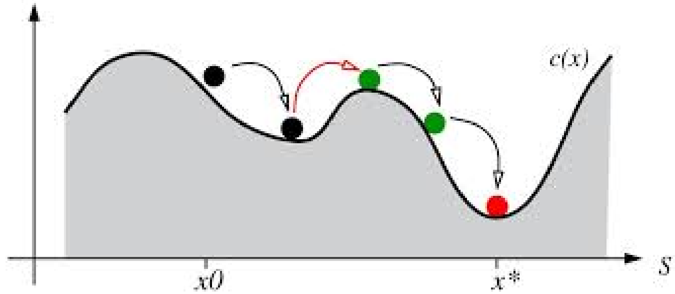
From Hopfield networks to Boltzmann machines

- Stochastic component

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ -1, & \text{with probability } 1\text{-}p_i \end{cases}$$

$$\boxed{p = \frac{1}{1 + e^{-\frac{1}{T}\sum_j w_{i,j} x_j}}}$$

Analogy to simulated annealing (relaxation technique common in metallurgy)

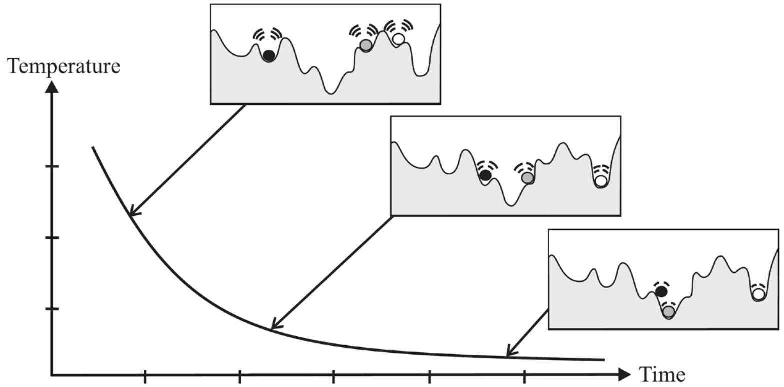


$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

"When optimising a large complex system with many degrees of freedom, instead of always going downhill, try to go downhill most of the time"

Simulated annealing to reach the global energy min

The critical role of temperature T.

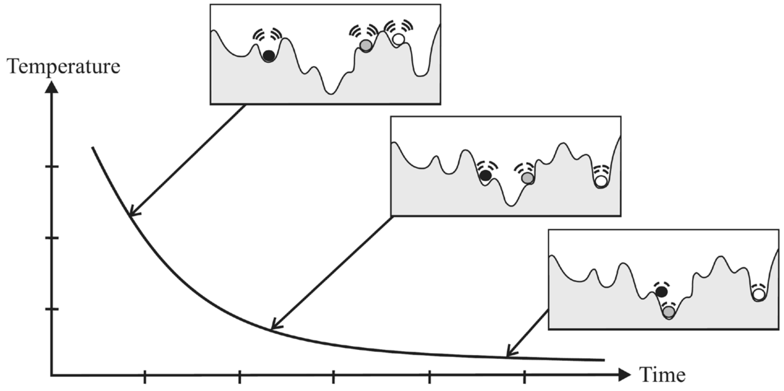


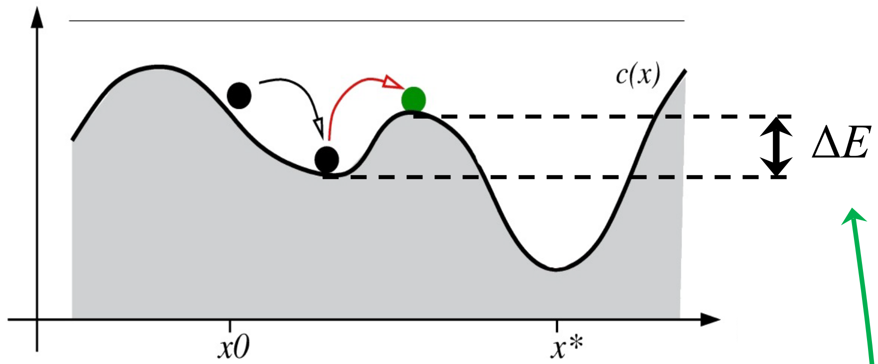
T decreases over time

$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

Simulated annealing to reach the global energy min

The critical role of temperature T.





$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

From Hopfield networks to Boltzmann machines

- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j$$

- The key difference is a stochastic nature of transitions

from state $s1$ to $s2$:

$$p_{s1\to s2} = \frac{1}{1+e^{(E_2-E_1)/T}} = \frac{1}{1+e^{\Delta E/T}}$$

From Hopfield networks to Boltzmann machines

- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j$$

- The key difference is a stochastic nature of transitions

Boltzmann (Gibbs) distribution defines the probability, p_i, that a system assumes the energy level E_i during thermal equilibrium:

$$p_i = \frac{e^{-E_i/T}}{\underbrace{\sum_{j}^{m} e^{-E_j/T}}_{Z}}$$

From Hopfield networks to Boltzmann machines

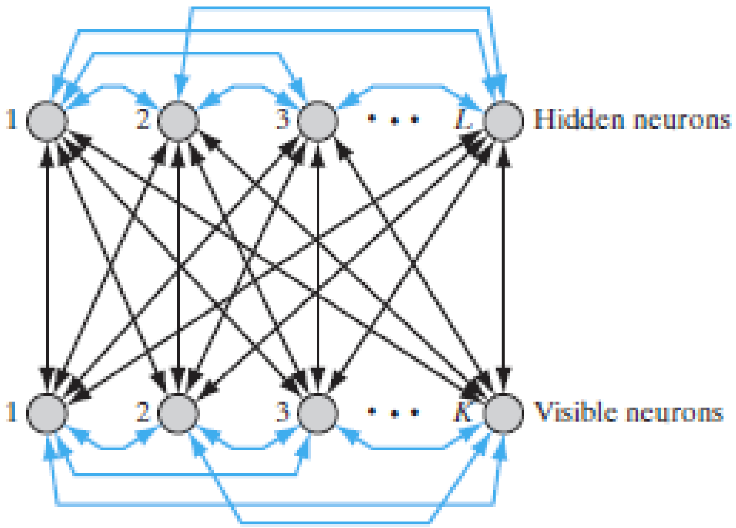
- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j$$

- Given a set of examples $\{\vec{x}_i\}_1^m$ the idea is to adjust $\mathbf{W}$ to describe data distribution (well matched to these examples)

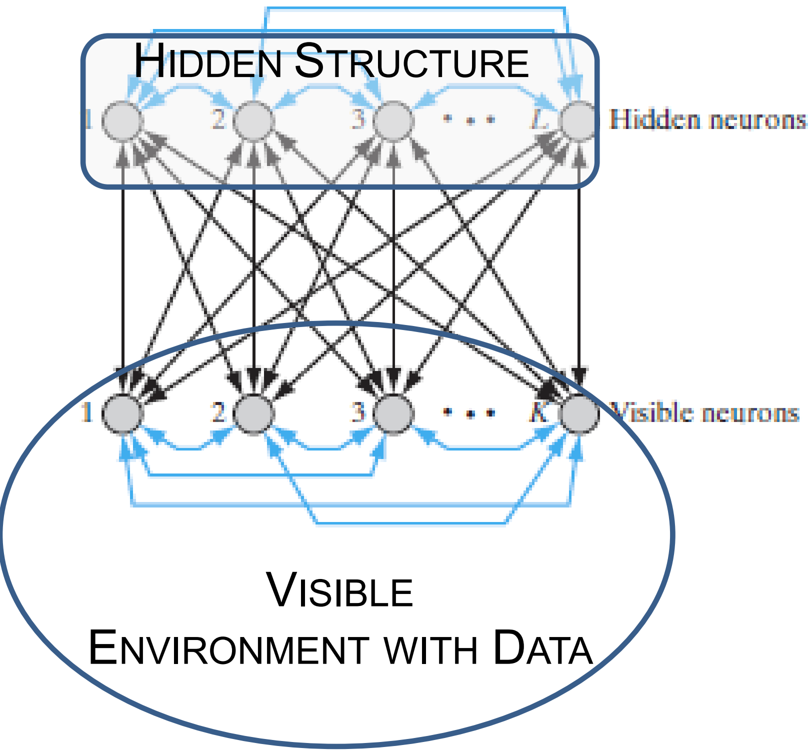
$$P(\vec{x}\,|\,\mathbf{W}) = \frac{e^{-E}}{Z} = \frac{1}{Z(\mathbf{W})}\exp\left(\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x}\right)$$

Hidden and visible units



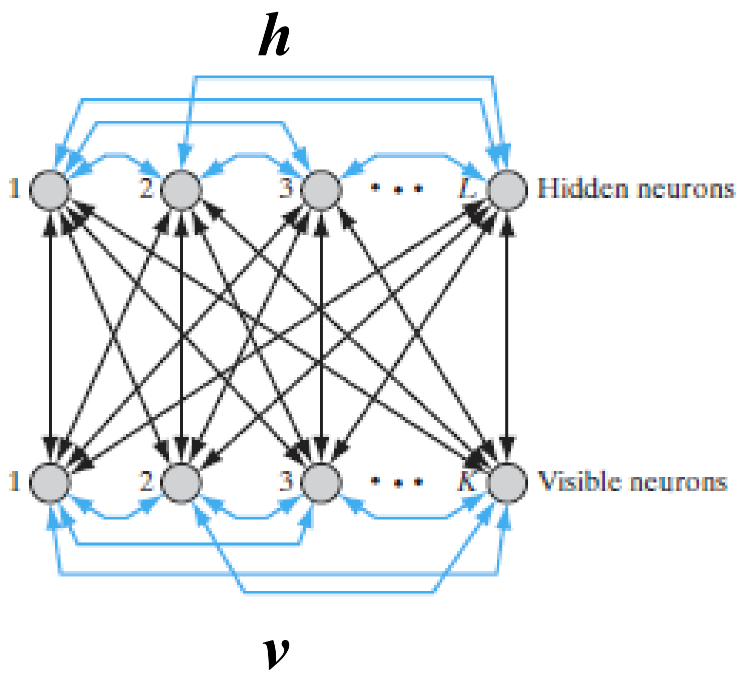
- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors

Hidden and visible units



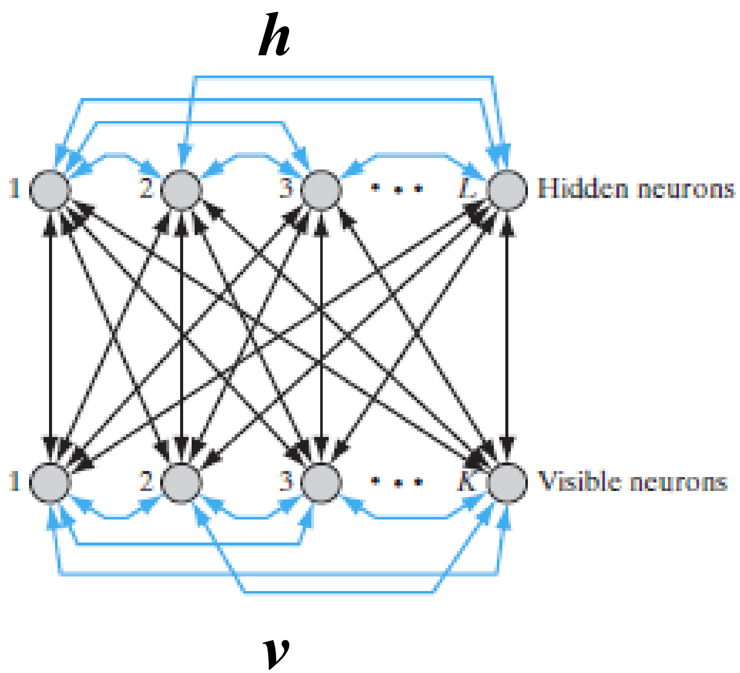
- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors

Hidden and visible units



- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors

Hidden and visible units



$$\boldsymbol{v}^{(p)} = \boldsymbol{x}^{(p)}$$

$$\Downarrow$$

$$\boldsymbol{y}^{(p)} = [\boldsymbol{x}^{(p)}, \boldsymbol{h}]$$

- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors
- Modelling a probability distribution (and hidden representation) by clamping patterns onto the visible units $\boldsymbol{v}^{(p_i)} = \boldsymbol{x}^{(p_i)}$

Boltzmann learning

- The primary goal is to correctly model input patterns according to Boltzmann distribution
  - each input pattern is assumed to last long enough (it might have to be clamped for long) for the network to reach thermal equilibrium (converge) at temperature T
  - to reduce this time, simulated annealing is used with a sequence decreasing temperatures (from “hot” to “cold”)
- Essentially, hidden units learn probabilistically representation of data (seen through visible units)

Boltzmann learning

- The idea is to maximise log-likelihood, $L(\mathbf{W})=\log(P(\mathbf{X})|\mathbf{W})$

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{W})}{\partial w_{ji}} = \eta(\rho_{j,i}^{+} - \rho_{j,i}^{-}), \quad \eta = \varepsilon / T$$

Boltzmann learning

- The idea is to maximise log-likelihood, $L(\mathbf{W})=\log(P(\mathbf{X})|\mathbf{W})$

$$\Delta w_{ji} = \varepsilon \frac{\partial L(\mathbf{W})}{\partial w_{ji}} = \eta(\rho^{+}_{j,i} - \rho^{-}_{j,i}), \qquad \eta = \varepsilon / T$$

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \underbrace{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}} - \underbrace{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}} \right\}$$

positive phase (awake), with clamping, $\boldsymbol{v}^{(p)}=\boldsymbol{x}^{(p)}$

negative phase (sleep) free running

$$\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} = \sum_p \sum_h P\left(\boldsymbol{h} \mid \boldsymbol{v} = \boldsymbol{x}^{(p)}\right) y_i y_j$$

$$\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} = \sum_p \sum_y P(\boldsymbol{y})\, y_i y_j$$

Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W} \right) = \sum_p \left\{ \left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})} - \left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})} \right\}$$

Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}} - \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}} - \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\mathrm{data}} - \left\langle y_i, y_j \right\rangle_{\mathrm{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)

$$\left\langle y_i, y_j \right\rangle_{data} \longleftarrow$$

Expected value at thermal equilibrium

- **Negative** phase involves updating all the units (can be very slow)

$$\left\langle y_i, y_j \right\rangle_{\mathrm{model}}$$

Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}} - \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)

Expected value at thermal equilibrium

- **Nega**

Thermal equilibrium <u>does not</u> imply only that the system settles down into the lowest energy state.

It is about the convergence of probability distribution over different configurations.

Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}} - \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)

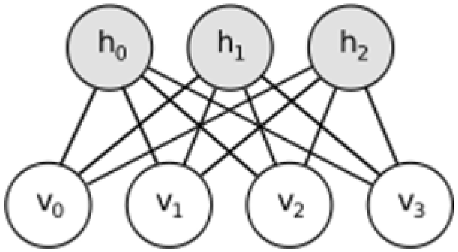
  "Hebbian learning" $\left\langle y_i, y_j \right\rangle_{data}$

- **Negative** phase involves updating all the units (can be very slow)

  "Hebbian forgetting" $\left\langle y_i, y_j \right\rangle_{\text{model}}$ prevent from learning false, spontaneously generated states

Next step to decrease the complexity

- Restricted Boltzmann machines



- Deep belief networks