Hopfield networks

Associative memories

What were you doing on 22 November 1963? (Or 20 January 2009)?

Human memories are content addressable. Given some partial clue, retrieve the whole item. (Does this describe Google?)

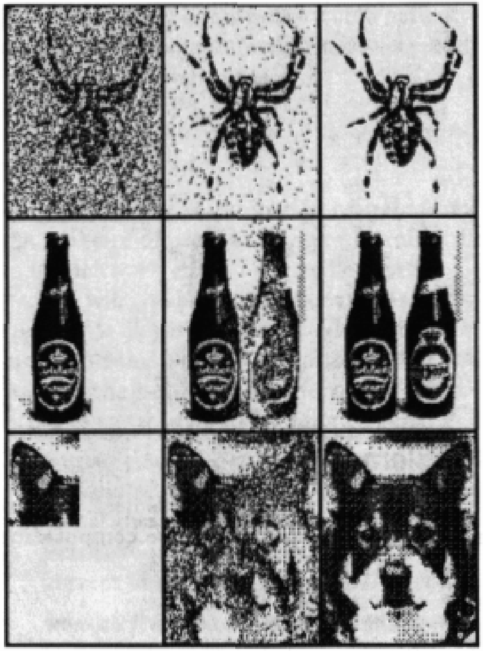
Compare this with direct access computer memories where you have to know the location to find the contents.

Which memory techniques are (a) distributed? (b) fault tolerant?

Types of associative memory:

Autoassociative	Given part of item X (partial, noisy), recall whole item X.
Heteroassociative	Given part of item X (partial, noisy), recall whole item Y.

Basins of attraction



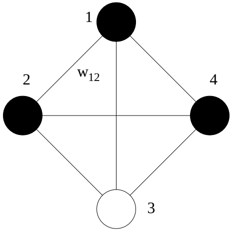
Hopfield (1982): Equations 1

Term	meaning
N	number of units in network
w_{ij}	strength of connection from unit i to unit j
μ	pattern number, $\mu = 1 \ldots p$
x_i^μ	ith component of pattern μ $\{+1, -1\}$
v_i	state of unit i $\{+1, -1\}$
θ_i	threshold value of unit i (often zero).

- Fully-connected network: Each neuron connects to all other neurons; typically $w_{ii} = 0$. Symmetric weights: $w_{ij} = w_{ji}$.

e.g. Store three patterns:

pattern 1	-1	-1	$+1$	-1
pattern 2	-1	$+1$	$+1$	$+1$
pattern 3	$+1$	$+1$	-1	-1



Equations 2

- State of unit *i*:

$$v_i = \text{sgn}(\sum_{j=1}^{N} w_{ij} v_j - \theta_i)$$

where $\text{sgn(x)} = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$

- (Assume weights have been chosen.)
- Start network by setting state of nodes to values of input pattern.
- Asynchronous update: Select one unit at random, and update its state. (Gibbs sampling.)
- Repeat until network is stable (the state of all units in the network is unchanging).
- Synchronous update is unstable . . . (ok for continuous time Hopfield model).

Stability of one pattern

Can the network remember just one pattern?
Assume we are just storing one pattern ($p = 1$) x_i.
For a stable network, $v_i = x_i \quad \forall i$. When the state of a unit is updated, it should be the same as the input pattern:

$$v_i = x_i$$

$$\mathrm{sgn}(\sum_{j=1}^{N} w_{ij} x_j) = x_i$$

What should w_{ij} be to satisfy this constraint? If we "guess" $w_{ij} = x_i x_j$ then:

$$\begin{aligned}\sum_{j=1}^{N} w_{ij} x_j &= (x_i x_1) x_1 + (x_i x_2) x_2 + \ldots + (x_i x_N) x_N \\ &= x_i (x_1^2 + x_2^2 + \ldots + x_N^2) \\ &= N x_i \qquad \text{since } x_i = \pm 1, x_i^2 = 1\end{aligned}$$

And since N is positive, we find that $sgn(\sum_{j=1}^{N} w_{ij} x_j) = x_i$.

Hopfield network: learning rule

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} x_i^\mu x_j^\mu$$

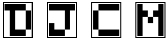
Constant 1/N is not needed for discrete updates.

Here we are measuring pairwise correlations.

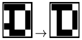
After setting weights as above, enforce $w_{ii} = 0$. Useful to show convergence.
Symmetric weights needed to ensure convergence.

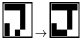
This could be regarded as a **Hebbian** learning rule. More of them later.

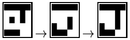
Example (Mackay)

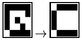

(a)

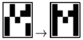
.	0	0	0	0	-2	2	-2	2	2	-2	0	0	0	2	0	0	-2	0	2	2	0	0	-2	-2
0	.	4	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	.	4	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	4	4	.	0	-2	-2	-2	-2	-2	-2	0	-4	0	-2	0	0	-2	0	-2	-2	4	4	2	-2
0	0	0	0	.	2	-2	-2	2	-2	2	-4	0	0	-2	4	-4	-2	0	-2	2	0	0	-2	2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

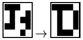

(b)

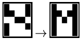

(c)

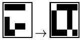

(d)

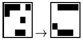

(e)

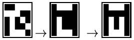

(f)

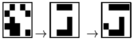

(g)

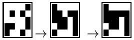

(h)


(i)


(j)

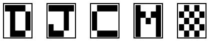

(k)


(l)

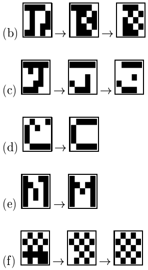

(m)

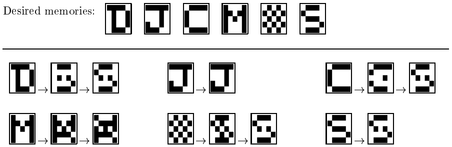
Fault tolerance and capacity (Mackay)

Left: 20/300 connections deleted and extra pattern added.
Right: One further pattern causes more breakdown.

(a)


```
 . -1  1 -1  1  x  x -3  3  x  x -1  1 -1  x -1  1 -3  x  1  3 -1  1  x -1
-1  .  3  5 -1 -1 -3 -1 -3 -1 -3  1  x  1 -3  1 -1 -1 -1 -1 -3  5  3  3 -3
 1  3  .  3  1 -3 -1  x -1 -3 -1 -1  x -1 -1 -1  1 -3  1 -3 -1  3  5  1 -1
-1  5  3  . -1 -1 -3 -1 -3 -1 -3  1 -5  1 -3  1 -1 -1 -1 -1 -3  5  x  3 -3
 1 -1  1 -1  .  1 -1 -3  x  x  3 -5  1 -1 -1  3  x -3  1 -3  3 -1  1 -3  3
 x -1 -3 -1  1  . -1  1 -1  1  3 -1  1 -1 -1  3 -3  1  x  1  x -1 -3  1  3
 x -3 -1 -3 -1 -1  . -1  1  3  1  1  3 -3  5 -3  3 -1 -1  x  1 -3 -1 -1  1
-3 -1  x -1 -3  1 -1  . -1  1 -1  3  1  x -1 -1  1  5  1  1 -1  x -3  1 -1
 3 -3 -1 -3  x -1  1 -1  . -1  1 -3  3  1  1  1 -1 -1  3 -1  5 -3 -1  x  1
 x -1 -3 -1  x  1  3  1 -1  . -1  3  1 -1  3 -1  x  1 -3  5 -1 -1 -3  1 -1
 x -3 -1 -3  3  3  1 -1  1 -1  . -3  3 -3  1  1 -1 -1 -1 -1  1 -3 -1 -1  5
-1  1 -1  1 -5 -1  1  3 -3  3 -3  . -1  1  1 -3  3  x -1  3 -3  1 -1  3 -3
 1  x  x -5  1  1  3  1  3  1  3 -1  . -1  3 -1  1  1  1  1  3 -5 -3 -3  3
-1  1 -1  1 -1 -1 -3  x  1 -1 -3  1 -1  .  x  1 -1  3  3 -1  1  1 -1 -1 -3
 x -3 -1 -3 -1 -1  5 -1  1  3  1  1  3  x  .  x  3 -1 -1  3  1 -3 -1 -1  1
-1  1 -1  1  3  3 -3 -1  1 -1  1 -3 -1  1  x  . -5 -1 -1 -1  1  1 -1 -1  1
 1 -1  1 -1  x -3  3  1 -1  x -1  3  1 -1  3 -5  .  1  1  1 -1 -1  1  1 -1
-3 -1 -3 -1 -3  1 -1  5 -1  1 -1  x  1  3 -1 -1  1  .  1  1 -1 -1 -3  1 -1
 x -1  1 -1  1  x -1  1  3 -3 -1 -1  1  3 -1 -1  1  1  . -3  3 -1  1 -3 -1
 1 -1 -3 -1 -3  1  x  1 -1  5 -1  3  1 -1  3 -1  1  1 -3  .  x -1 -3  1 -1
 3 -3 -1 -3  3  x  1 -1  5 -1  1 -3  3  1  1  1 -1 -1  3  x  . -3 -1 -5  1
-1  5  3  5 -1 -1 -3  x -3 -1 -3  1 -5  1 -3  1 -1 -1 -1 -1 -3  .  3  x -3
 1  3  5  x  1 -3 -1 -3 -1 -3 -1 -1 -3 -1 -1 -1  1 -3  1 -3 -1  3  .  1 -1
 x  3  1  3 -3  1 -1  1  x  1 -1  3 -3 -1 -1 -1  1  1 -3  1 -5  x  1  . -1
-1 -3 -1 -3  3  3  1 -1  1 -1  5 -3  3 -3  1  1 -1 -1 -1 -1  1 -3 -1 -1  .
```



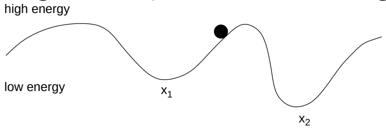


Energy function

- Two units are in low energy state (stable) if states agree with sign of connection weight.

weight	v_i	v_j	energy
$+$	$+1$	$+1$	low
$+$	$+1$	-1	high
$-$	$+1$	$+1$	high
$-$	$+1$	-1	low

- Captured by $e_{ij} = -w_{ij}v_iv_j$.
- Energy of whole system: $E = -1/2 \sum_{ij} w_{ij}v_iv_j$
- System travels through state space until the energy reaches a minimum:



- Each minima ("attractor") corresponds to a stable state (stored pattern).
- Despite recurrency, system always settles in a stable state since energy monotonically decreases.

Energy minimised (1)

E is an energy function that always decreases to a minimum.

Proof: When updating the state of a given unit k, separate energy function into terms that depend on k and those that do not:

$$\begin{aligned} E &= -\frac{1}{2}\sum_{i\neq k, j\neq k} w_{ij}v_iv_j - \frac{1}{2}\sum_{j\neq k} w_{kj}v_kv_j - \frac{1}{2}\sum_{i\neq k} w_{ik}v_iv_k \\ &= -\frac{1}{2}\sum_{i\neq k, j\neq k} w_{ij}v_iv_j - \sum_{i\neq k} w_{ik}v_iv_k \end{aligned}$$

Let first term be constant (C) since it does not involve k:

$$E = C - v_k \sum_{i\neq k} w_{ik}v_i$$

$$E = C - v_k a_k \qquad (a_k \text{ is the total input to unit } k)$$

Energy minimised (2)

After unit k is updated, new state = $v_k^{'}$, new energy $= E^{'}$

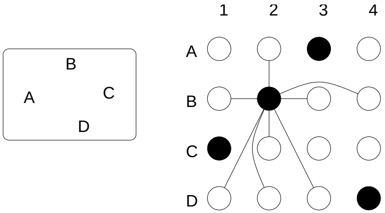
$$E^{'} = C - v_k^{'} a_k$$
$$\Delta E = E^{'} - E = a_k (v_k - v_k^{'})$$

a_k	v_k	$v_k^{'}$	ΔE
$+$	-1		
$+$	$+1$		
$-$	-1		
$-$	$+1$		

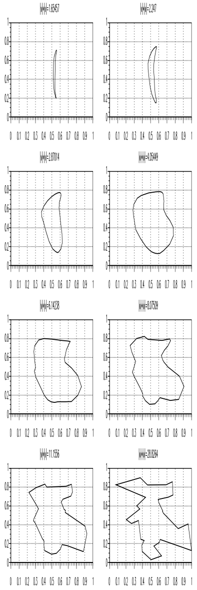
$\Delta E \leq 0$. Therefore E always stays the same or decreases.

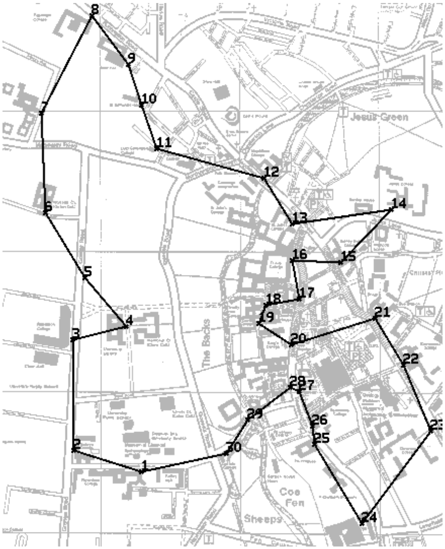
Example: Travelling salesman problem (TSP)

- Related to “N-Rooks” and “N-Queens” chess problems.
- Hopfield and Tank (1985) used network to solve TSP. Example use of “relaxation network”.
- Analogue rather than binary states $v_i \in [-1, 1] : v_j = \tanh(\sum_i w_ijv_i)$.
- Weights encode knowledge of problem:
    1. Cannot visit two cities at once (inhibition within column)
    2. Cannot visit a city more than once (inhibition within row)
    3. Inhibition between neighbouring columns proportional to distance between cities.
- Finds valid tours. Optimal tour hard to find!



TSP, Cambridge style (Mackay, orig. Aiyer)


(a)



(b)

Limitations: capacity and alternative states

- Empirical observations from Hopfield (1982) suggested about 0.15N patterns could be recalled before recall was severely impaired.
- Theoretical limits (assuming random inputs) suggest around 0.138N (Hertz, Krogh & Palmer, 1991).
- **Palimpsest** memories: erase old memories to store new memories. Clip weights or exponential decay of old weights.
- Can use supervised learning algorithms to improve capacity of Hopfield network.
- “spurious-states” as well as input patterns:
    1. Inverse of pattern also stable.
       e.g. $[+1+1-1] \Leftrightarrow [-1-1+1]$
       Detect inverse by adding sign bit.
    2. Mixture states: combination of an odd number of input patterns.
    3. “Spin glass states” not correlated with any original pattern (when p is large).