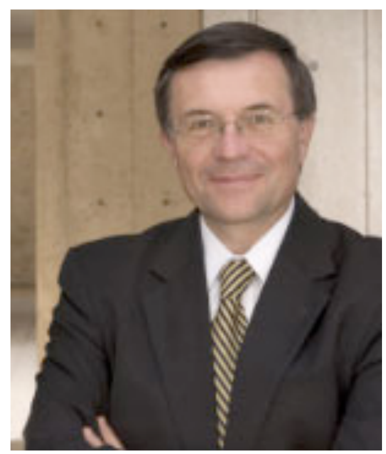
Hopfield Networks

Hopfield Networks

- Proposed in 1982 by John Hopfield: Professor at Princeton, Caltech, now Princeton
- According to Terry Sejnowski (then Hopfield's graduate student), Hopfield nets may have been suggested by Sejnowski himself.

John Hopfield



Terry Sejnowski

Approaches to Hopfield Nets

- Recurrent neural nets without sequential input, or
- Extend **linear associative memory** ideas by adding cyclic connections, or
- Special case of Bart Kosko's BAM (Bi-Directional Associative Memory, proposed later), or
- Derive from Cohen-Grossberg theorem (not covered yet).

Still Earlier Work

- Earlier work in 1977 by Anderson, et al. in a cognitive psychology journal called "Brain State in a Box" (BSB) **anticipated** the Hopfield model of 1982.

- It was a slightly more complex model, and used satlins as the activation function.

- The state-space is continuous inside a hypercube (the "box").

- They provided both Hopfield's method of setting weights (outer product) and an iterative learning method.

Hopfield Nets

- Generally considered to be **fixed-weight** models; **they don't learn**.
- However, one way to get the weights is through the supervised **Hebbian** outer-product summation as used in the Linear Associative Model.
- Some insensitivity to noise or network damage.
- Some **extensions** do learn: e.g. Boltzmann machine.

Applications

- Associative or content-addressable memory.
- Model of memory as a dynamical system.
- A technique for finding solutions to certain **optimization** problems.
- The *practical* applications do not seem so plentiful.

Commentary from James Anderson’s book (1995)

- “Theoretical physicists are an unusual lot, acting like gunslingers in the old West, anxious to prove themselves against a really good problem. And there aren’t that many really good problems that might be solvable.
- As soon as Hopfield pointed out the connection between a new and important problem (network models of brain function) and an old and well-studied problem (the Ising model), many physicists rode into town, so to speak, with the intention of shooting the problem full of holes and then, the brain understood, riding off into the sunset looking for a newer, tougher problem. (Who was that masked physicist?)”.

Commentary from James Anderson's book (1995)

- "Hopfield [1982] made the portentous comment: 'This case is isomorphic with an Ising model,' thereby allowing a deluge of physical theory (and physicists) to enter neural network modeling. This flood of new participants transformed the field.

- In 1974 Little and Shaw made a similar identification of neural network dynamics with the Ising model, but for whatever reason, their idea was not widely picked up at the time".

Commentary from James Anderson's book (1995)

- "Unfortunately, the problem of brain function turned out to be more difficult than expected, and it is still unsolved, although a number of interesting results about Hopfield nets were proved.

- At present, many of the traveling theoreticians have traveled on".

Comment in Hertz, Krogh, and Palmer, 1991

- “Gerard Toulouse has called Hopfield’s use of symmetric connections a ‘clever step backwards from biological realism’. The cleverness arises from the existence of an energy function”.

- Toulouse, et al. *Spin glass model of learning by selection*. Proc. of the National Academy of Sciences, 83, 1695-1698, 1986.

Hopfield Memory

- As with the Linear Associative Memory, the “stored patterns” are represented by the **weights**.

- To be effective, the patterns should be reasonably **orthogonal**.

Model Variants

- Basic: Discrete state, discrete time, asynchronous
- Same as basic, but synchronous
- Continuous state, discrete time
- Continuous state, continuous time

Basic Model

- N neurons, fully connected in a cyclic fashion:
  - Values are +1, -1.
  - Each neuron has a weighted input from all **other** neurons.
  - Weights are **symmetric**: $w_{ij} = w_{ji}$
    and self-weights $= w_{ii} = 0$
  - Activation function on each neuron i is

$$f(net) = sgn(net) = \begin{cases} 1 \text{ if } net > 0 \\ -1 \text{ if } net < 0 \end{cases} \qquad (net_i = \Sigma w_{ij}\, x_j)$$

  - If net = 0, then the output is the same as before, **by convention**.

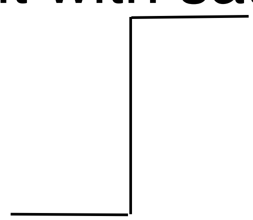
Thresholds/Biases

- There are no separate thresholds or biases.

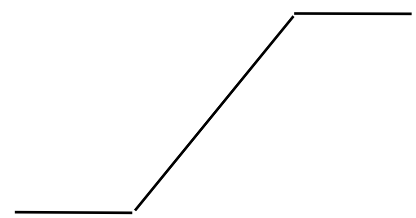
- However, these could be represented by units that have all weights = 0 and thus never change their output.

Continuous-State Variant

- On the previous slides, sgn is the same as hardlims (symmetric hard-limiter).
- We could allow continuous neuron outputs and replace it with satlins (symmetric saturating limiter)



discrete
hardlims



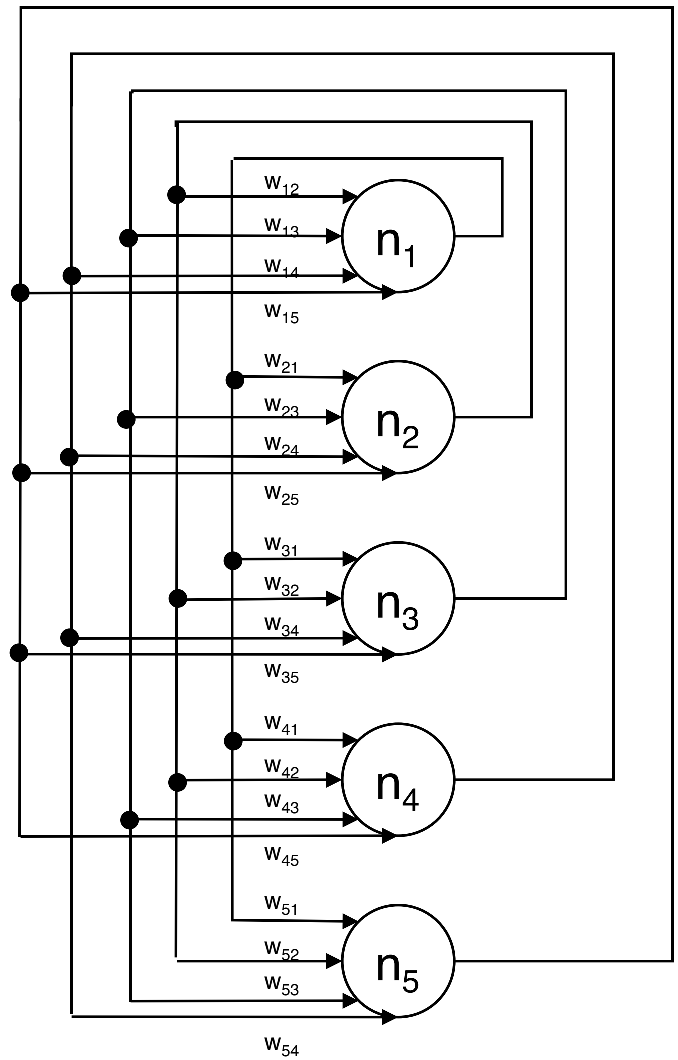
continuous
satlins

- One advantage of the continuous version is that it makes it easier to visualize certain phenomena such as “attractors”.

Hopfield Net

$$\begin{pmatrix} 0 & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & 0 & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & 0 & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & 0 & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & 0 \end{pmatrix}$$

$$w_{ij} = w_{ji}$$



Operation: Asynchronous Version

- Each neuron's output is initially **forced** to a specified value; this is the "input" state.
- Repeat until no change:
  A neuron that has f(net) $\neq$ current output is "fired", changes its output to 1 or -1 according to the definition of f.
- The firable neuron is chosen arbitrarily.
- When and if the network stabilizes, the current state is the "output".

Operation: Synchronous Version

- All **firable** neurons are first identified, then all change their state **simultaneously**.

- While this may be viewed as an expedient, it may create behavioral anomalies, such as **oscillations,** not present in the asynchronous version.

Termination for the Asynchronous Case

- Energy Minimization:
  - For an appropriate definition of “energy”, **each single firing** can be show to **decrease** the energy.
  - Energy is provably bounded from below, thus **cannot decrease forever**; there is a definite minimum.
  - Therefore operation must eventually **terminate**.

Final State

- For **asynchronous** (basic) behavior, a **unique** final state is **not** guaranteed: it could be a **local minimum**.

- For **synchronous** behavior, **if** there is a final state, it could still be a **local minimum** (it is also reachable by asynchronous firing). However, the network could instead **oscillate** forever.

Weights

- Similar to **storing patterns** in the Linear Associative Memory, weights can be computed by summing the **outer product** of the normalized pattern vectors.

- However, after computing the sum of the outer products, the diagonal element are **forced** to 0.

Working an Example

- Two patterns: (1, -1, 1) and (-1, 1, -1)
- Compute the **outer products**, sum, normalize, and set diagonals to 0:
  - $(1, -1, 1)^T * (1, -1, 1) +$

    $(-1, 1, -1)^T * (-1, 1, -1) =$

$$\begin{pmatrix} 2 & -2 & 2 \\ -2 & 2 & -2 \\ 2 & -2 & 2 \end{pmatrix} \Rightarrow \frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$$

Force Diagonal to 0

$\frac{1}{3}$ = 1/(number of neurons)

Working an Example

- Eight states total: (-1, -1, -1) … (1, 1, 1)
- For each state, compute the possible next states using the firing rule and the weight matrix:

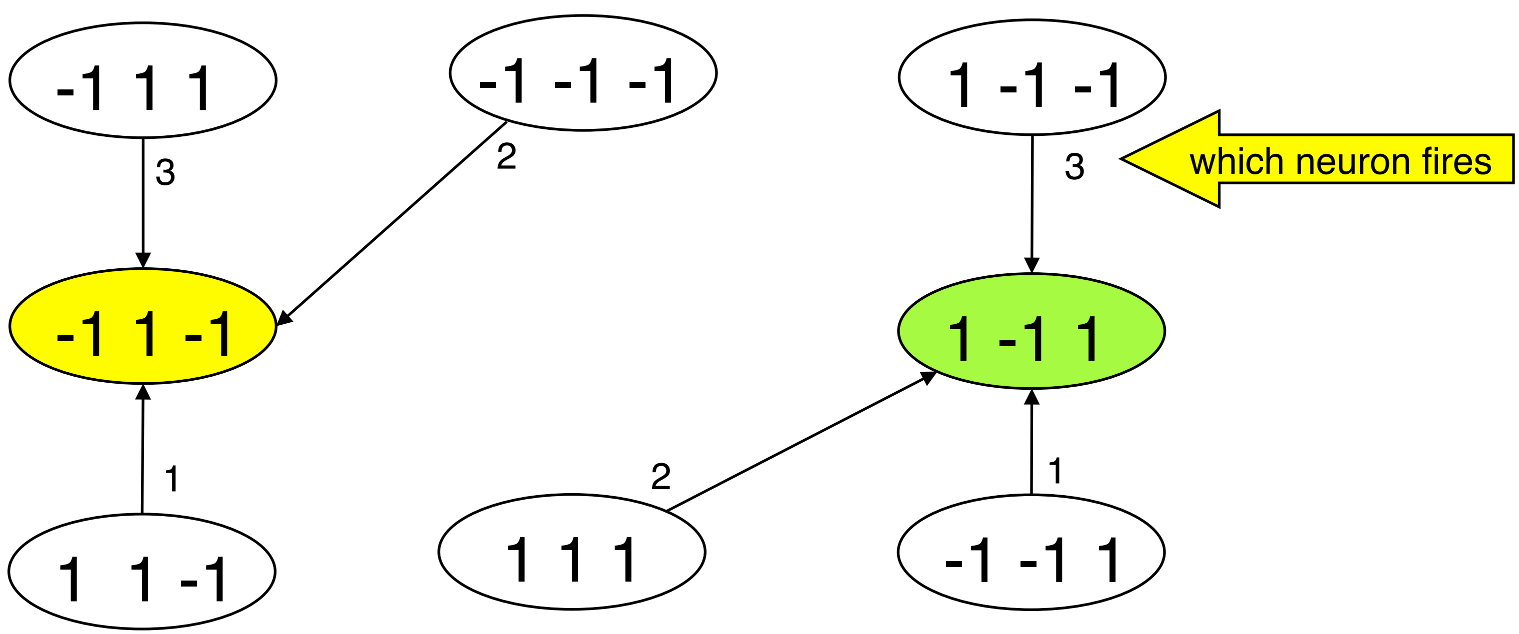
$$\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$$

- Then plot the transitions, noting where the patterns occur.

Working an Example (**asynchronous**)

states as column vectors

$$\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \qquad \begin{pmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$



Working an Example (synchronous)

- states as columns

$$\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \begin{matrix} -1 \\ -1 \\ -1 \end{matrix}\begin{pmatrix} -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

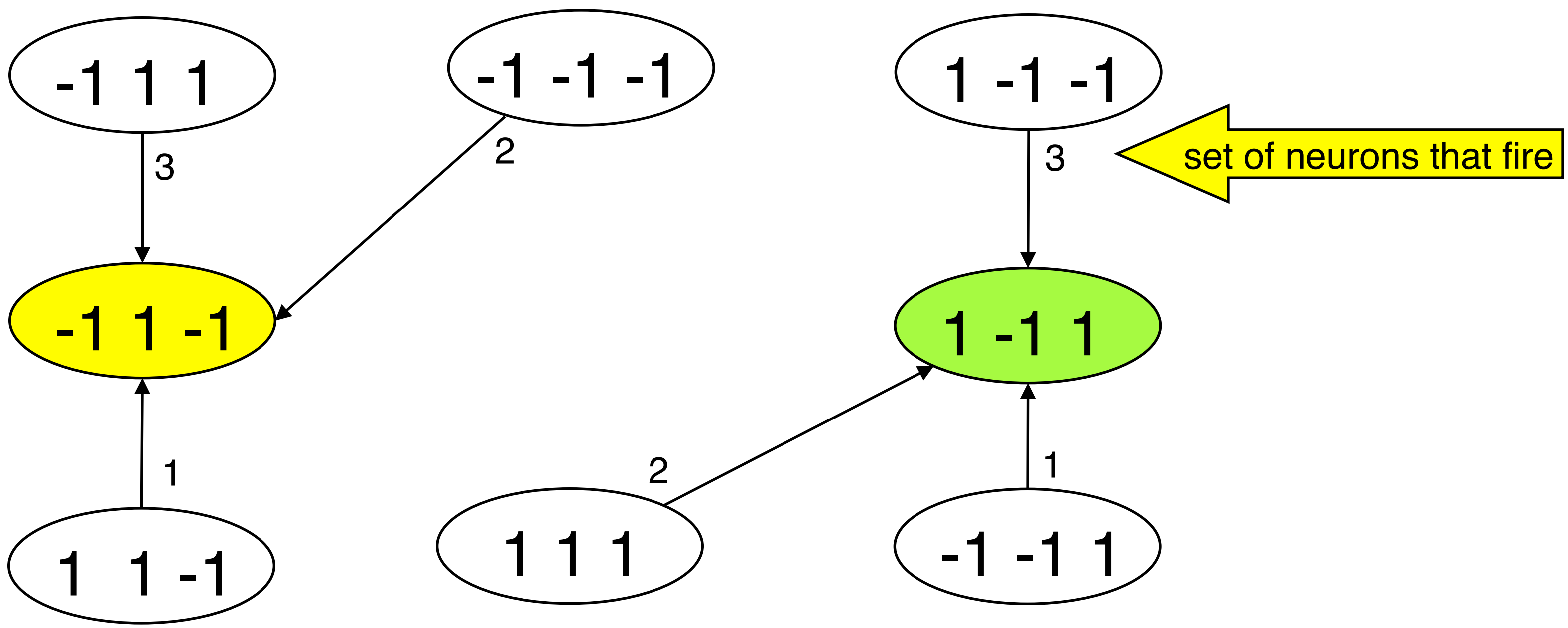
$$= \frac{1}{3}\begin{pmatrix} 0 & 4 & 0 & 0 & 0 & 4 & -4 & 0 \\ 4 & 0 & 4 & 0 & 0 & -4 & 0 & -4 \\ 0 & 0 & -4 & -4 & 4 & 4 & 0 & 0 \end{pmatrix}$$

- next states =

$$\begin{matrix} -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \end{matrix}$$

Working an Example (synchronous)

- next states = $$\begin{array}{r} \\ 1 \\ -1 \end{array}\begin{pmatrix} -1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & -1 & \\ 1 & -1 & -1 & -1 & 1 & -1 & 1 & \end{pmatrix}$$



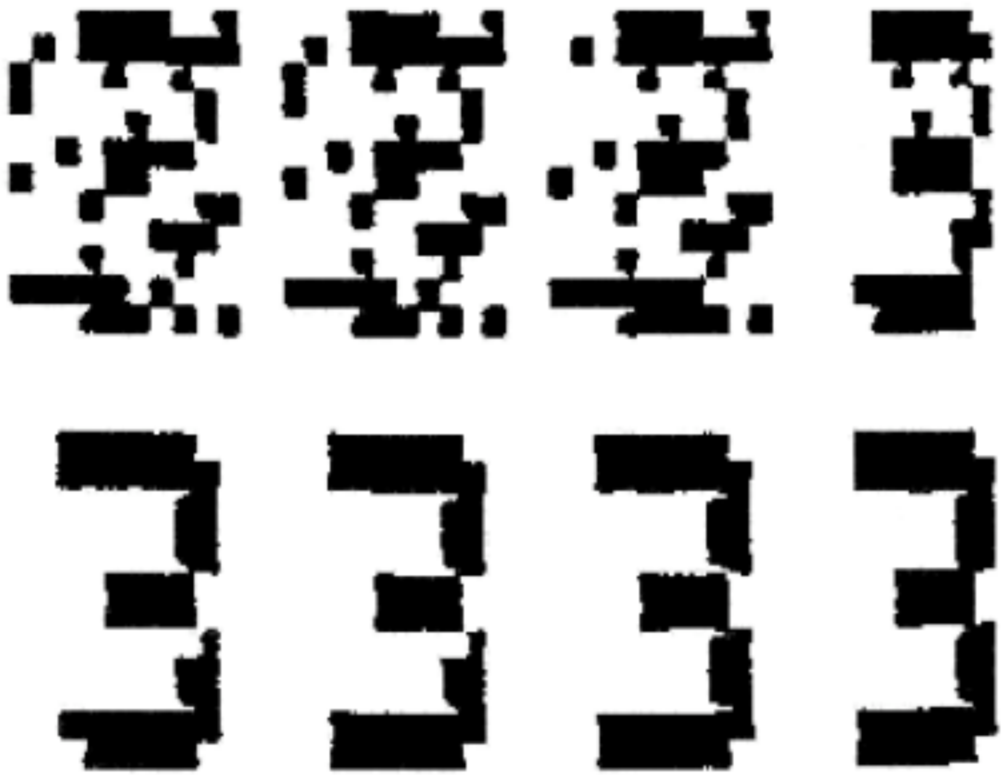
Comparison

- In this example, the asynchronous and synchronous behaviors worked out to be the same.
- This **won't** always be the case.
- Firing a neuron in the asynchronous *could* **disable** one of the neurons that would have fired simultaneously in the synchronous case.
- Conceivably, the synchronous case could therefore have **cycles** in its behavior.
- See if you can find an example.

Showing Hopfield State Transitions

A) EIGHT EXEMPLAR PATTERNS

Showing Hopfield State Transitions



B) OUTPUT PATTERNS FOR NOISY "3" INPUT

Images from Hopfield's Paper

(130x180 pixels)

Hopfield Capacity

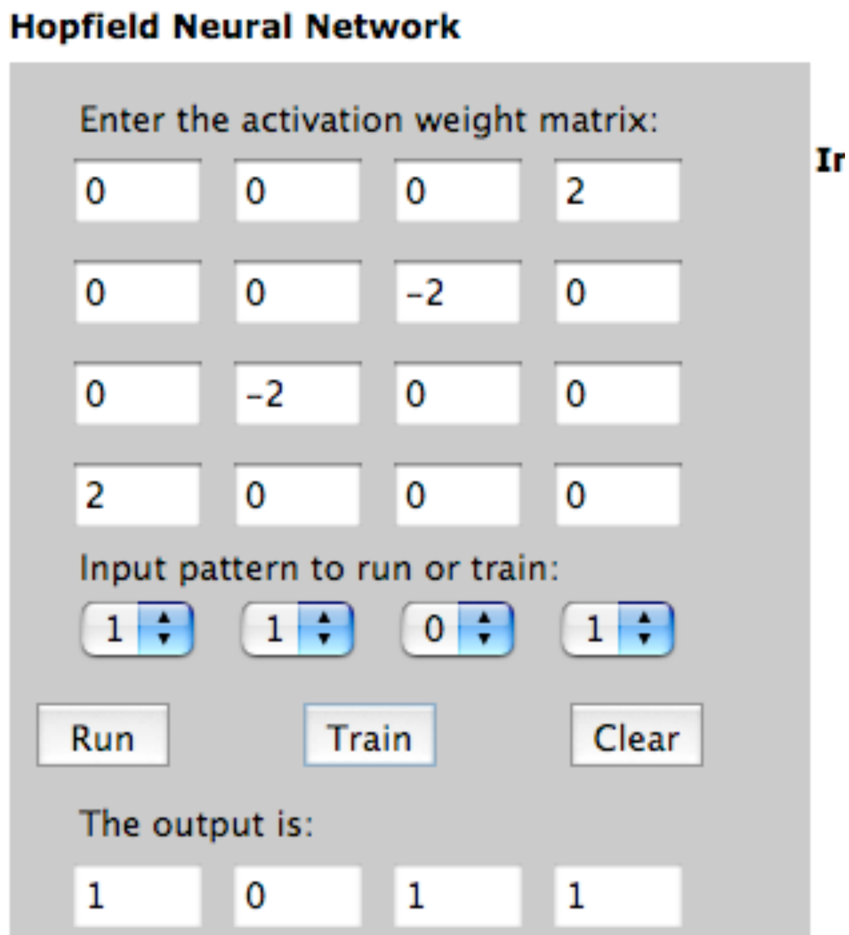
- From Haykin's book, for storage with at most .01 probability of error, asymptotically we can store ***at most*** N/(4 ln N) **patterns** with N neurons.

N	N/(4 ln N)
100	5
1000	36
10000	271
100000	2171
1000000	18096

- However, the number of states for N neurons is 2^N, and the hardware cost is $O(N^2)$, since there are N weights per neuron.

Hopfield Demos

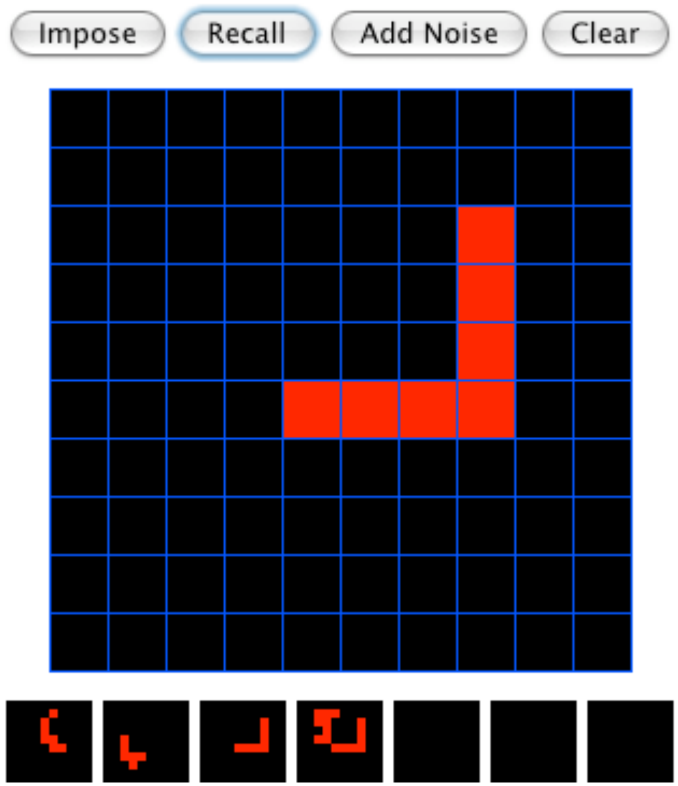
http://www.heatonresearch.com/articles/61/page1.html



Hopfield Demos

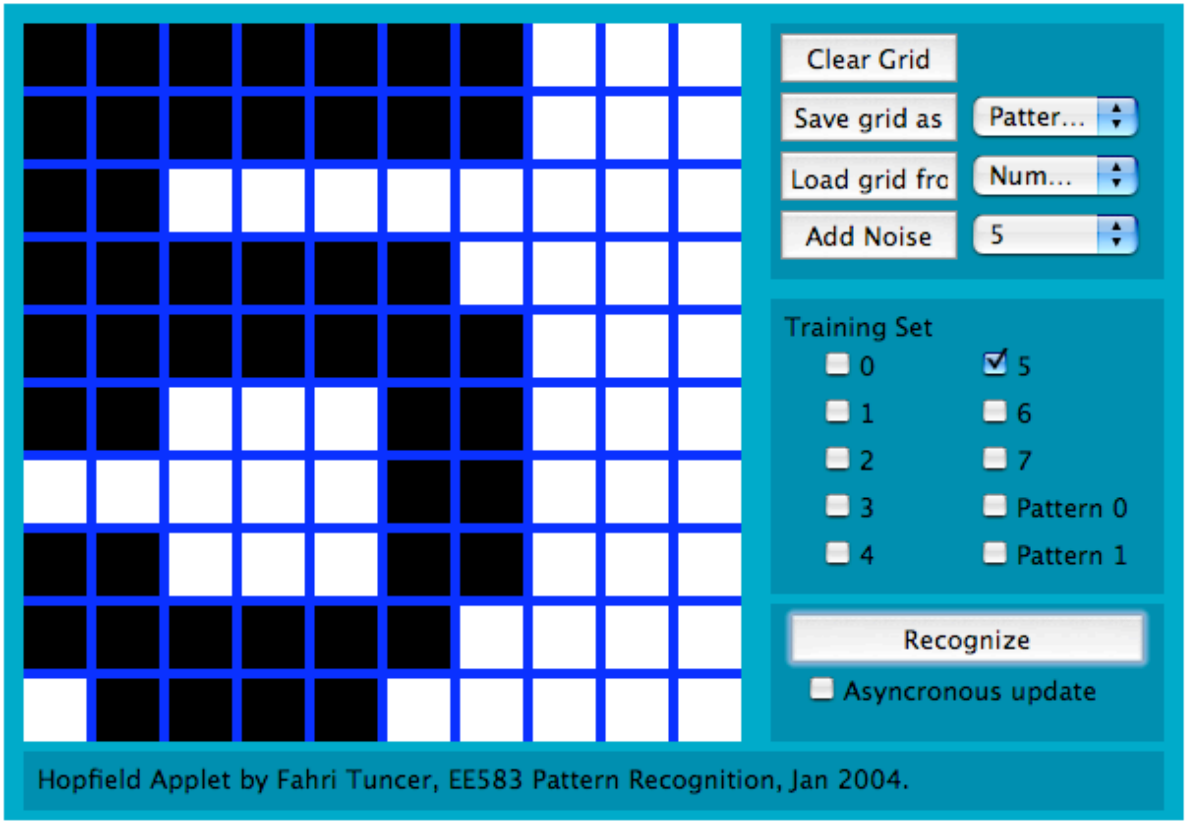
http://suhep.phy.syr.edu/courses/modules/MM/sim/hopfield.html

This demo remembers the bit-map patterns that have been imposed, For testing purposes.



Hopfield Demos

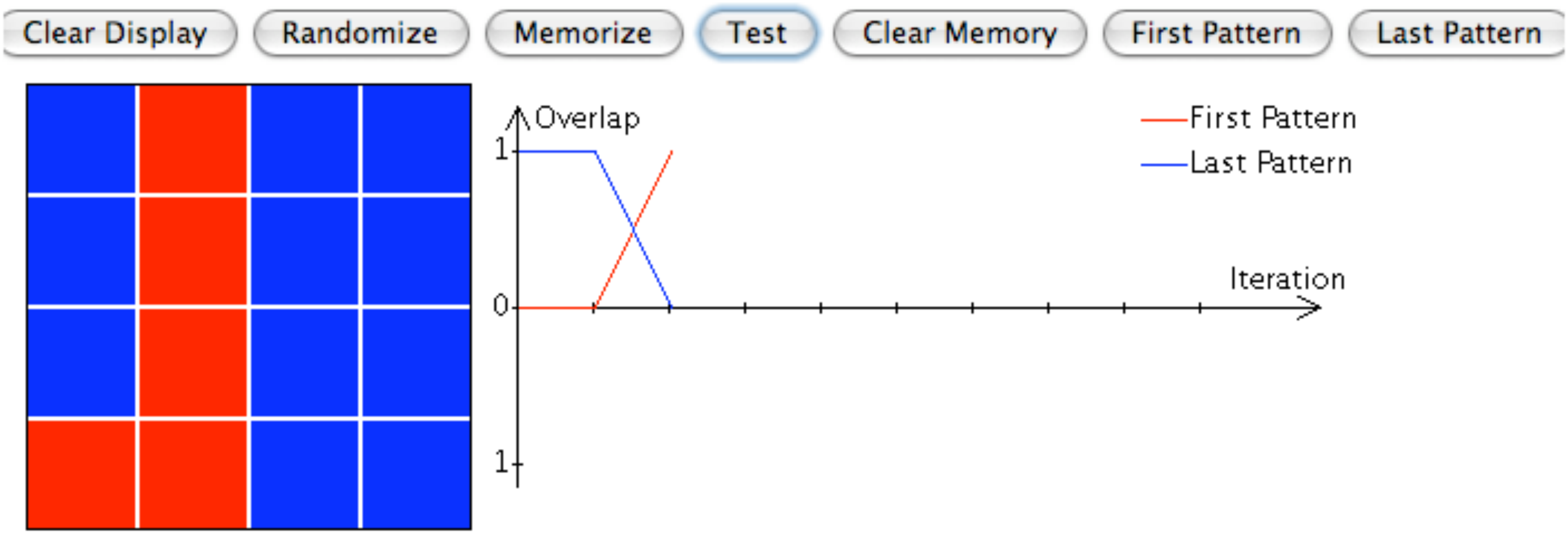
http://www.eee.metu.edu.tr/~alatan/Courses/Demo/Hopfield.htm



Hopfield Demos

http://lcn.epfl.ch/tutorial/english/hopfield/html/index.html

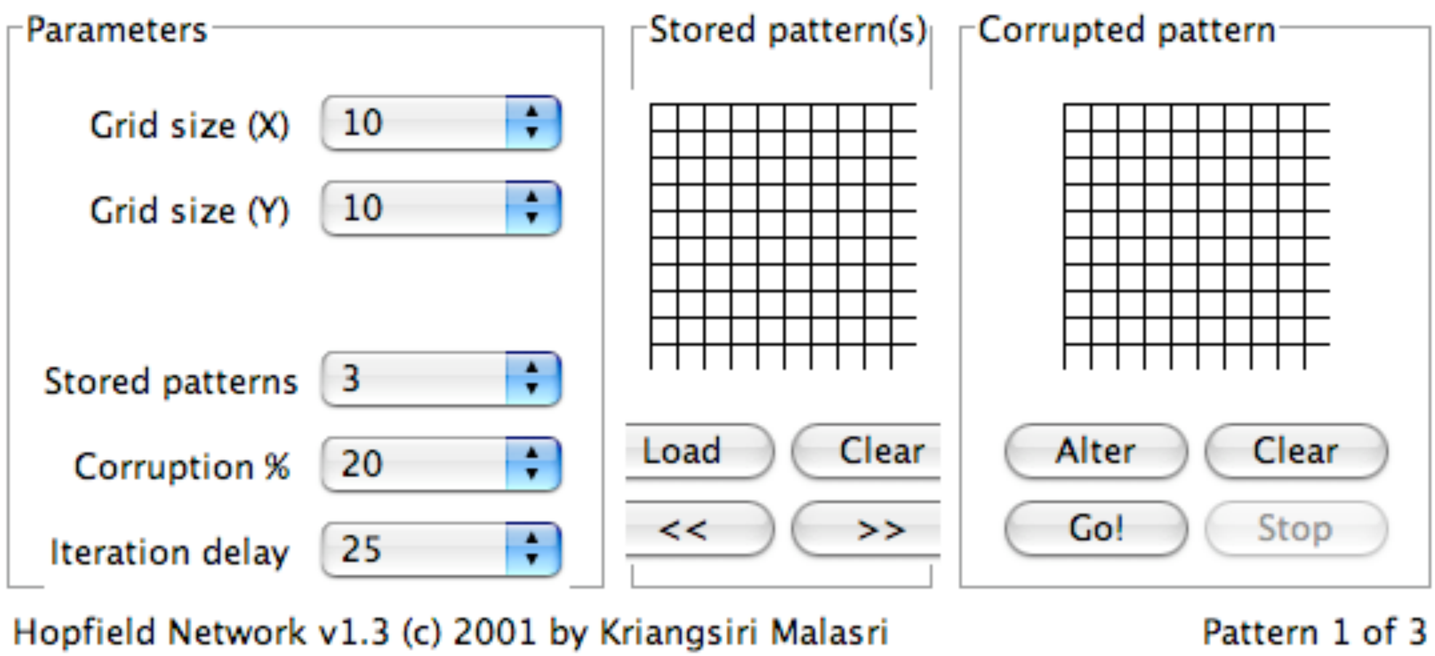
This demo provides an assessment of pattern orthogonality.



Hopfield Demos

http://www.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html

This demo shows updating in progress.



Proving that an Asynchronous Hopfield Net Terminates

- Define an **energy function**:

$$E(y_1, y_2, \ldots, y_n) = -\Sigma\, \Sigma\, w_{ij}\, y_i y_j$$

where $(y_1, y_2, \ldots, y_n)$ is the vector of neuron outputs, w_{ij} is the weight from neuron j to neuron i, and the double sum is over i and j.

- Remember that w is symmetric ($w_{ij} = w_{ji}$) and diagonal terms are 0.

Proving that an Asynchronous Hopfield Net Terminates

- Observation: The energy function is bounded from below.

- **Claim**: Firing any transition *decreases* the value of the energy function.

  $$E(y_1, y_2, \ldots, y_n) = -\Sigma\, \Sigma\, w_{ij}\, y_i y_j$$

- Therefore the net cannot fire forever.

- Note: This function might not be the *only* function with the desired properties.

Proof of Claim

- When a neuron i fires, the *increase* (new-old) in energy is entirely due to the contribution of y_i to $\Sigma\, w_{ij}\, y_i y_j$. Since w is symmetric, the amount of this increase is $-\Sigma w_{ij} y_i' y_j \;-\; -\Sigma w_{ij} y_i y_j$ where y_i' represents the new value of y_i and the sum is over $i \neq j$ only.
- Since neuron i *changes*, $y_i' = -y_i$, so the energy increase is

  $$2\Sigma\, w_{ij} y_i y_j = 2y_i \Sigma\, w_{ij} y_j$$

  where the right-hand summation is over j, where $j \neq i$ only.

Proof of Claim

- The energy *increase* is
  $2y_i\Sigma\, w_{ij}y_j$

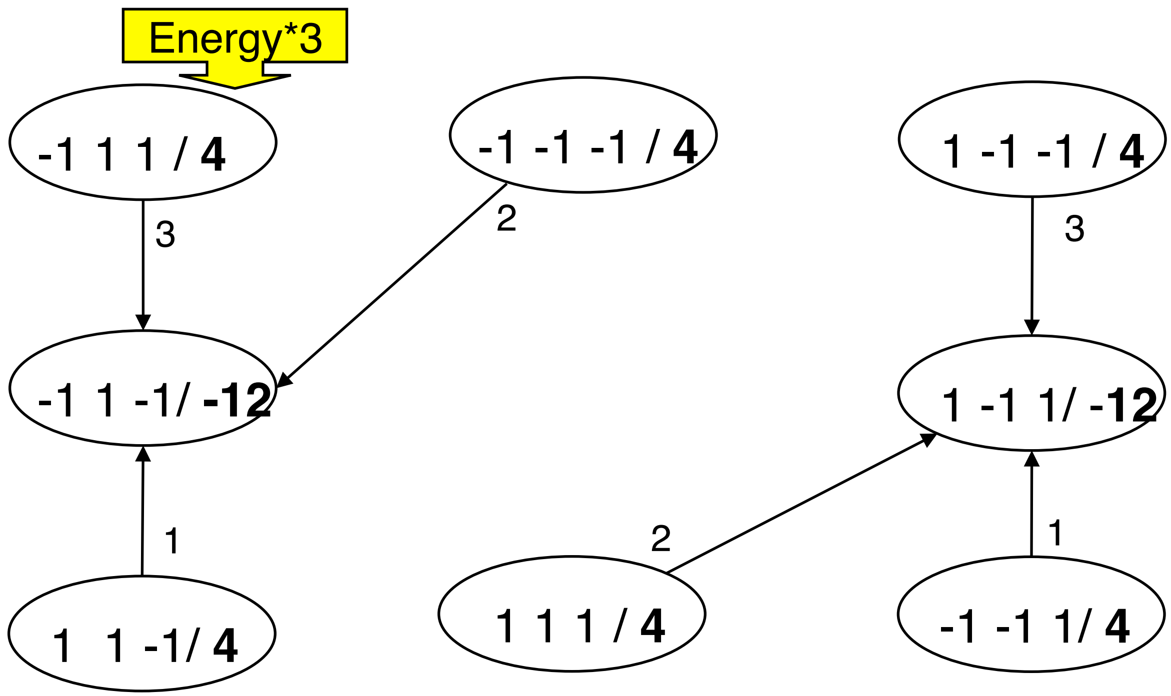
- **If $\mathbf{y_i}$ = 1**: ($y'_i = -1$), then we must have $\Sigma\, w_{ij}y_j < 0$ in order to activate the neuron, so the increase is 2*1*(negative) which is *negative*.

- **If $\mathbf{y_i}$ = -1**: ($y_i' = 1$), then we must have $\Sigma\, w_{ij}y_j > 0$ in order to activate the neuron, so the increase is 2*(-1)*(postive), which is negative.

- So there is a net energy **decrease** either way.

Checking Energy

- $$\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$$



Note on Synchronous Firing

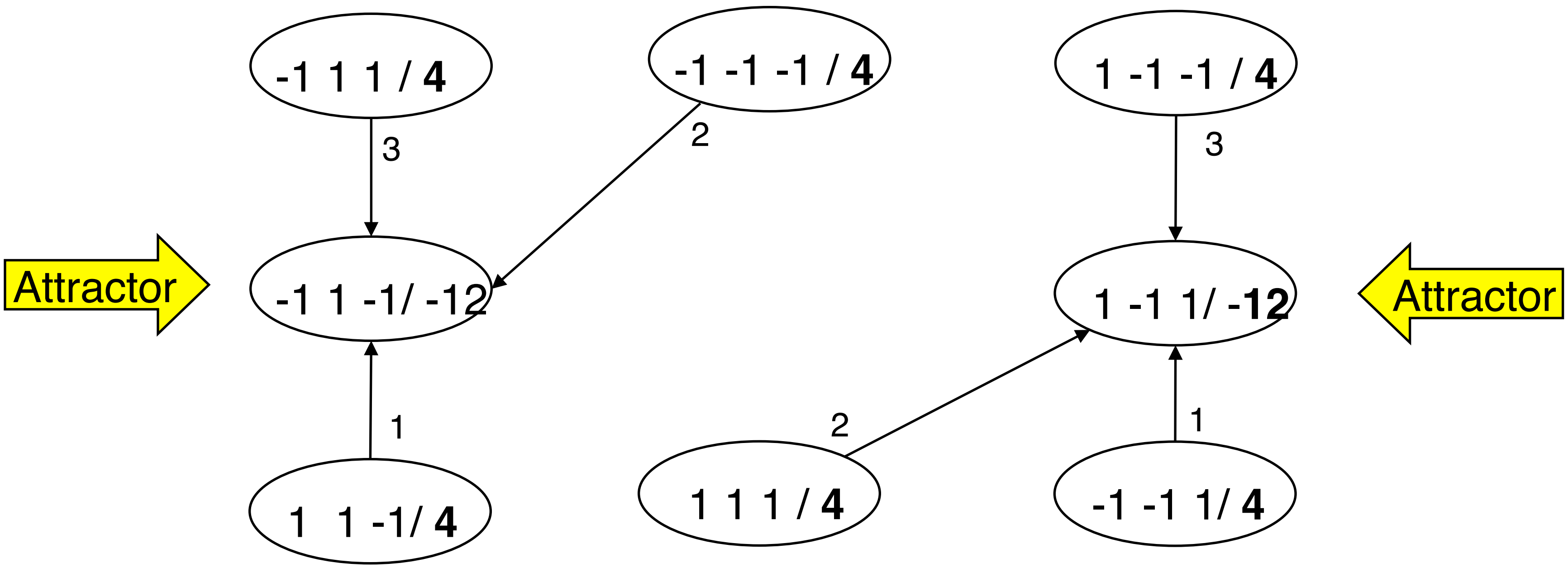
- In contrast to asynchronous firing, synchronous firing *may* increasing the energy.
- The analysis doesn't go through if several neurons fire at the same time.

Attractors

- Minimal energy states are known as “attractors” in the theory of dynamical systems.

- There can also be “repellors” and “saddles” (aka “meta-stable states”).

Attractors



-1 1 1 / **4**
-1 -1 -1 / **4**
1 -1 -1 / **4**
3
2
3
Attractor
-1 1 -1/ -12
1 -1 1/ -**12**
Attractor
1
2
1
1 1 -1/ **4**
1 1 1 / **4**
-1 -1 1/ **4**

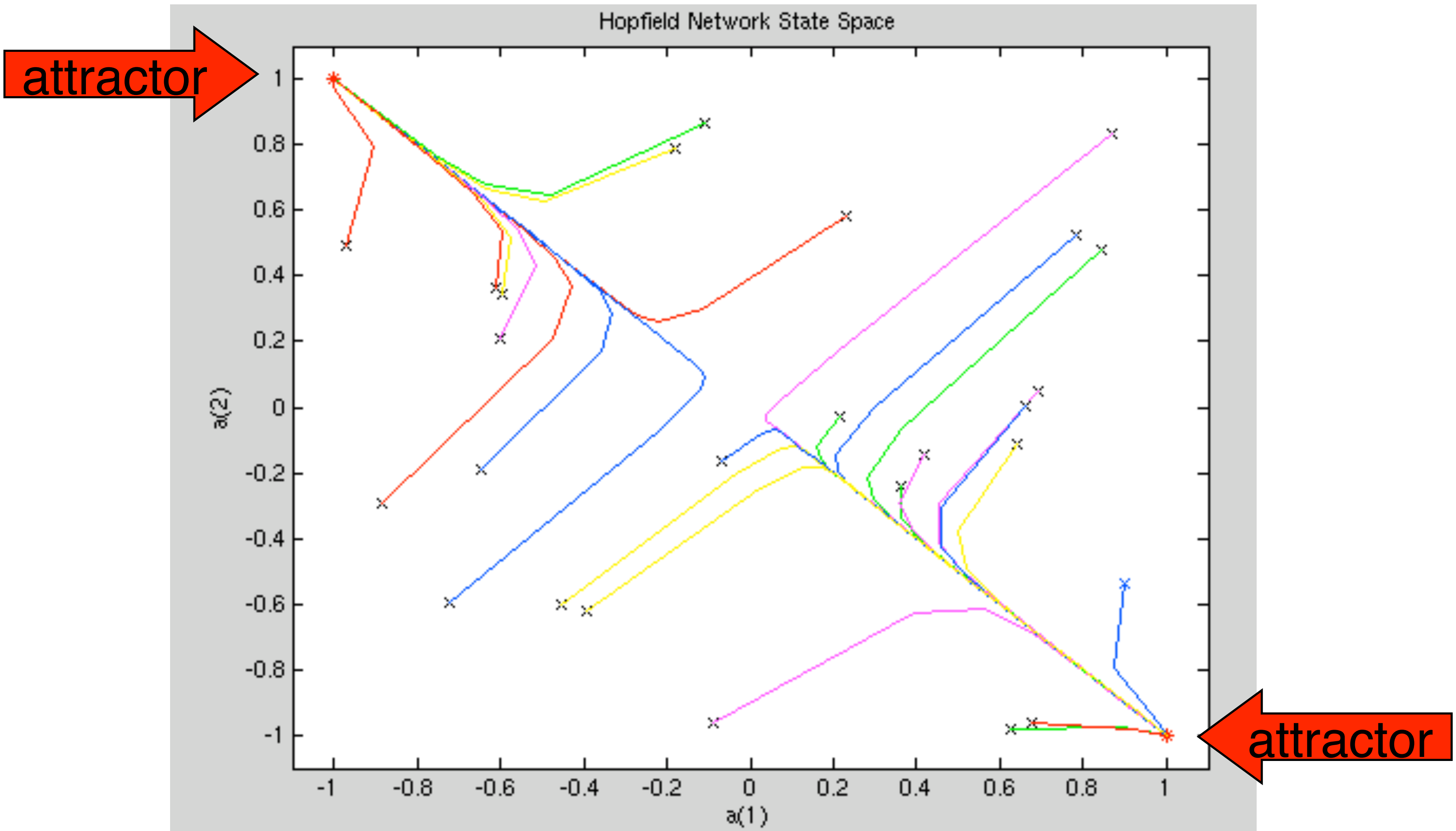
Demonstrating Attractors

- The phenomenon is easier to see with continuous-valued states, as there are more of them.

- matlab: demohop1, 2, 3 (uses *continuous* activation with satlins)

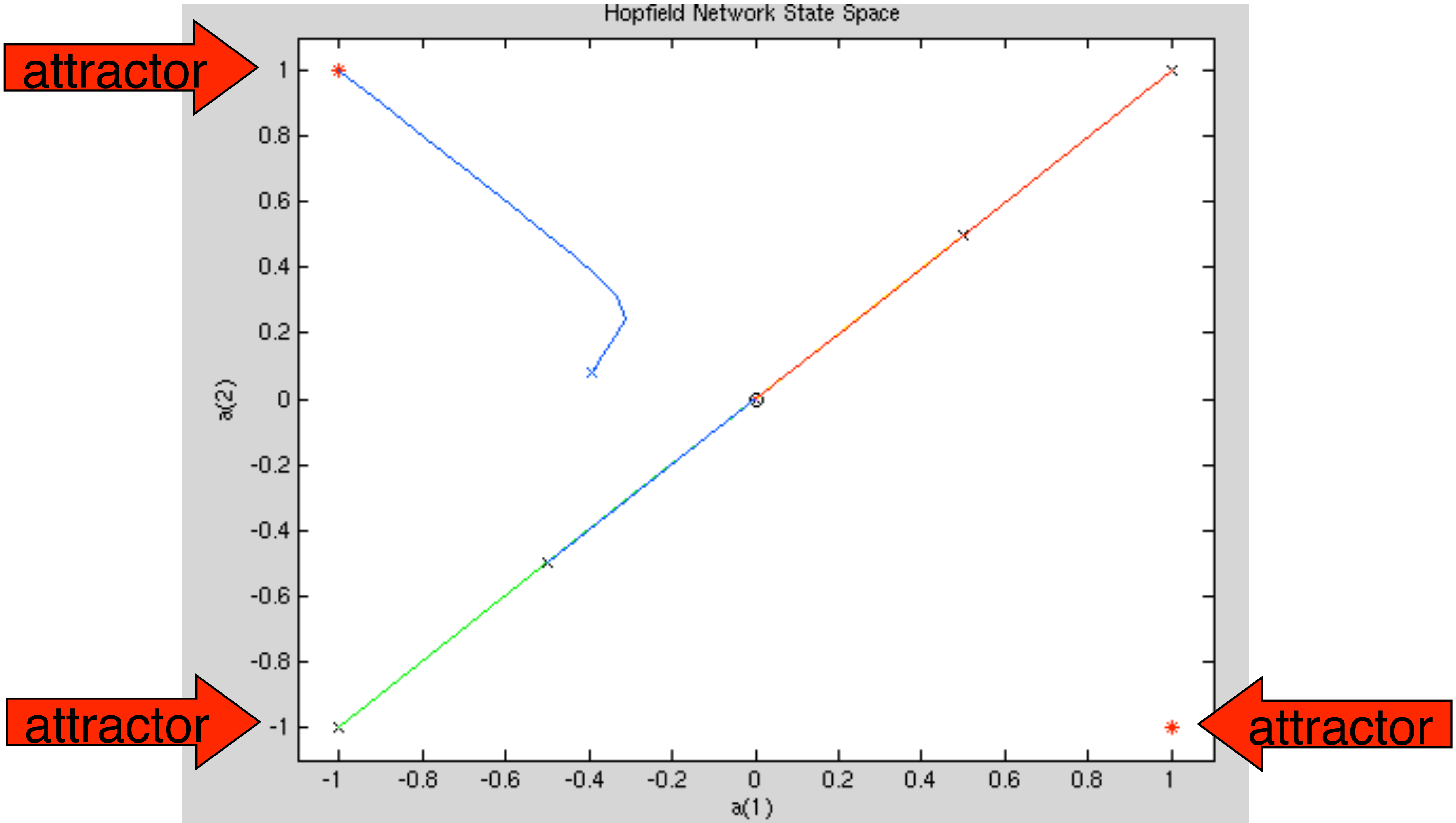
demohop1

continuous state space, 2 neurons, satlins



demohop2

continuous state space, 2 neurons, satlins



Stored Patterns Correspond to Attractors

- When the Hebb rule is used with ***orthogonal*** patterns, stored patterns correspond to attractors (stable, or minimum-energy, states).

- The reasoning is analogous to the case with the linear associative memory.

Stored Patterns Correspond to Attractors

- The supervised Hebb weight matrix is given by
  $W = \Sigma pp^T$ (with diagonals forced to 0)
  where the summation is over all patterns p as column vectors (pp^T is the outer product).

- Let q be a pattern. Assuming **linear** activation functions for the moment, we have *stability* (i.e. minimum energy) if Wq = q (actually satlins(Wq) = q).

- Also, stored patterns are **eigenvectors** of W, since $Wq = \lambda q$ is the equation determining eigenvalues λ and eigenvectors q.

Clarification: Stability of Stored Attractors

- Suppose $W = \Sigma pp^T$ where the patterns p are orthonormal.
- Suppose q is one of the patterns p
- Then $Wq = (\Sigma pp^T)q = \Sigma(pp^Tq) = \Sigma p(p^Tq)$
- Assuming that patterns are orthonormal, $(p^Tq) = 0$ unless $p = q$, in which case $(p^Tq) = 1$.
- Thus $Wq = q$.

What if the patterns are not orthogonal?

- Then $Wq = q$ might not hold for a pattern q, and a pattern input could move to another stable state.

Spurious Attractors

- Not every attractor is necessarily a pattern.
- For example, if **p** is an attractor, then so is -**p** (i.e. the *negative* of an image).
- Also, certain *linear combinations* of attractors may be attractors themselves.
- These aspects limit the applicability of Hopfield nets as pattern retrieval devices.

Example

- Patents:

$$[1 \quad 1 \; -1 \; -1]$$
$$[1 \quad 1 \quad 1 \quad 1]$$
$$[-1 \; -1 \quad 1 \quad 1]$$

(all Euclidean lengths = 2 = sqrt(4))

- Normalized Patterns :

$$[.5 \quad .5 \; -.5 \; -.5]$$
$$[.5 \quad .5 \quad .5 \quad .5]$$
$$[-.5 \; -.5 \quad .5 \quad .5]$$

Matlab

```
p =           % normalized patterns are columns

    0.5000    0.5000   -0.5000
    0.5000    0.5000   -0.5000
   -0.5000    0.5000    0.5000
   -0.5000    0.5000    0.5000

>> W = p*p'          % outer product

W =

    0.7500    0.7500   -0.2500   -0.2500
    0.7500    0.7500   -0.2500   -0.2500
   -0.2500   -0.2500    0.7500    0.7500
   -0.2500   -0.2500    0.7500    0.7500
```

Matlab

```
>> for i = 1:4
      W(i, i) = 0
      end

W =          % weight matrix

        0     0.7500   -0.2500   -0.2500
   0.7500     0        -0.2500   -0.2500
  -0.2500    -0.2500    0         0.7500
  -0.2500    -0.2500    0.7500    0
```

Matlab

```
>> W*p
ans =

    0.6250    0.1250   -0.6250
    0.6250    0.1250   -0.6250
   -0.6250    0.1250    0.6250
   -0.6250    0.1250    0.6250

>> hardlims(W*p)

ans =          % each column is the original pattern
     1     1    -1
     1     1    -1
    -1     1     1
    -1     1     1
```

Example

- A Non-Pattern:
  [-1   -1 -1 -1]

Matlab

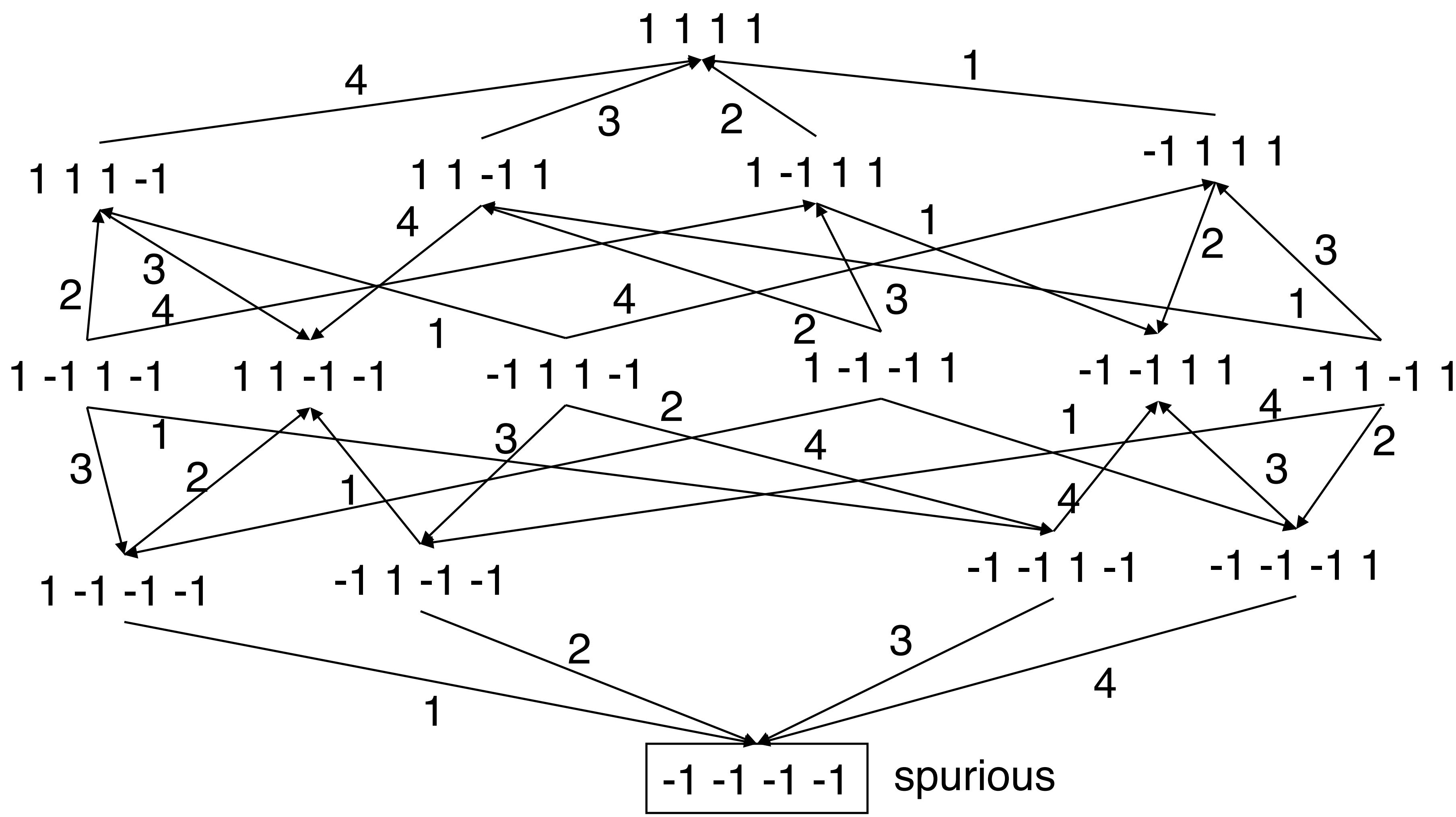
```
q = [-1 -1 -1 -1]'

q =
    -1
    -1
    -1
    -1

>> hardlims(W*q)

ans =               % non-pattern is also stable
    -1
    -1
    -1
    -1
```

Transition Diagram



Spurious attractors in the 8 digits example (stable, but not equal to stored patterns)

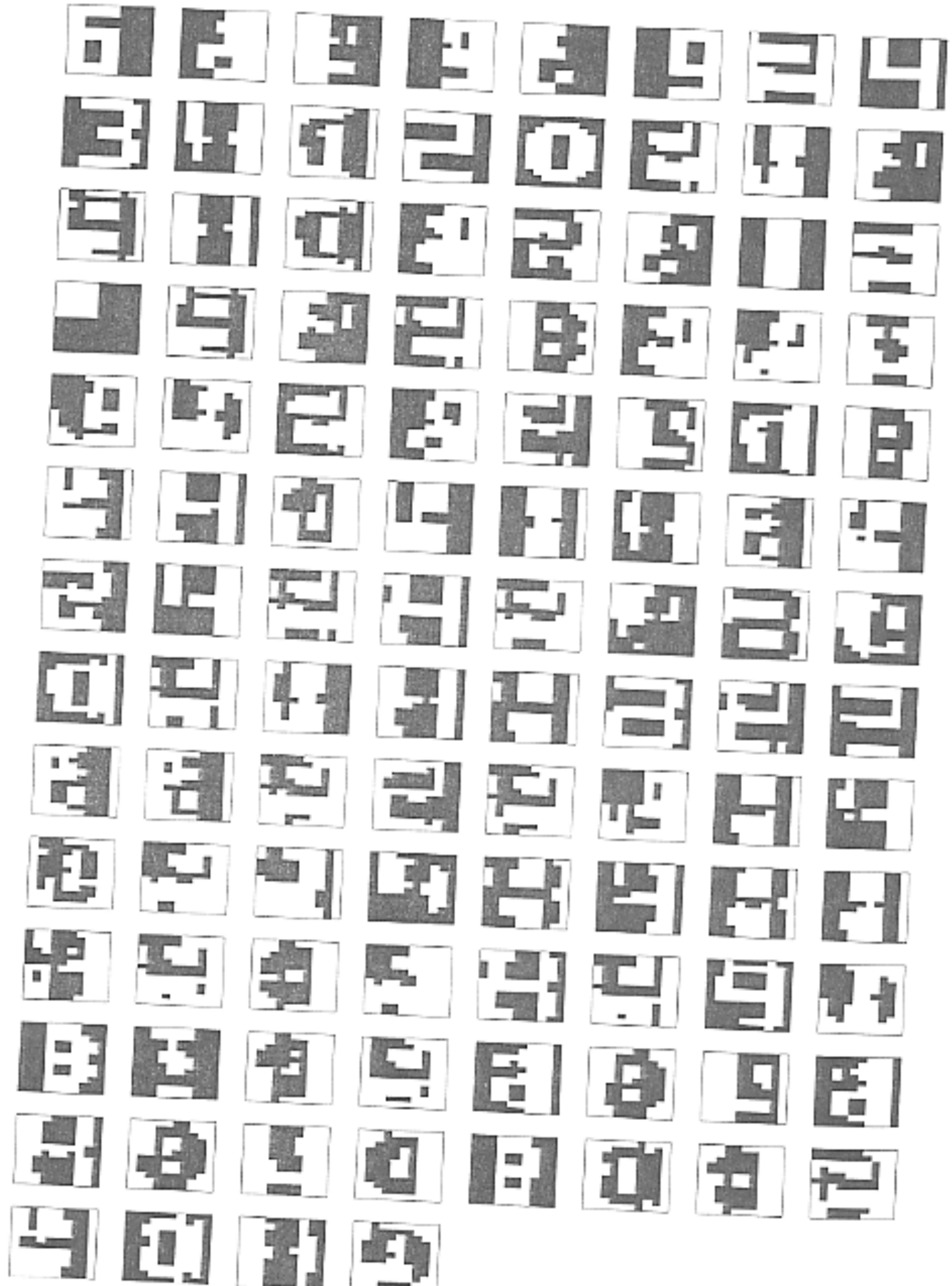


FIGURE 14.20 Compilation of the spurious states produced in the computer experiment on the Hopfield network.

Minimizing Spurious Attractors

- Hopfield, et al. proposed "unlearning" as a way to get rid of spurious attractors.

- The Hebb rule is not the only way to set weights. The following paper presents a weight setting technique for minimizing the number of spurious attractors:

  Li, Michel, and Porod, Analysis and synthesis of a class of neural networks: linear systems operating on a closed hypercube, IEEE Trans. on Circuits and Systems, **36**, 11, 1405-1422, Nov. 1989.

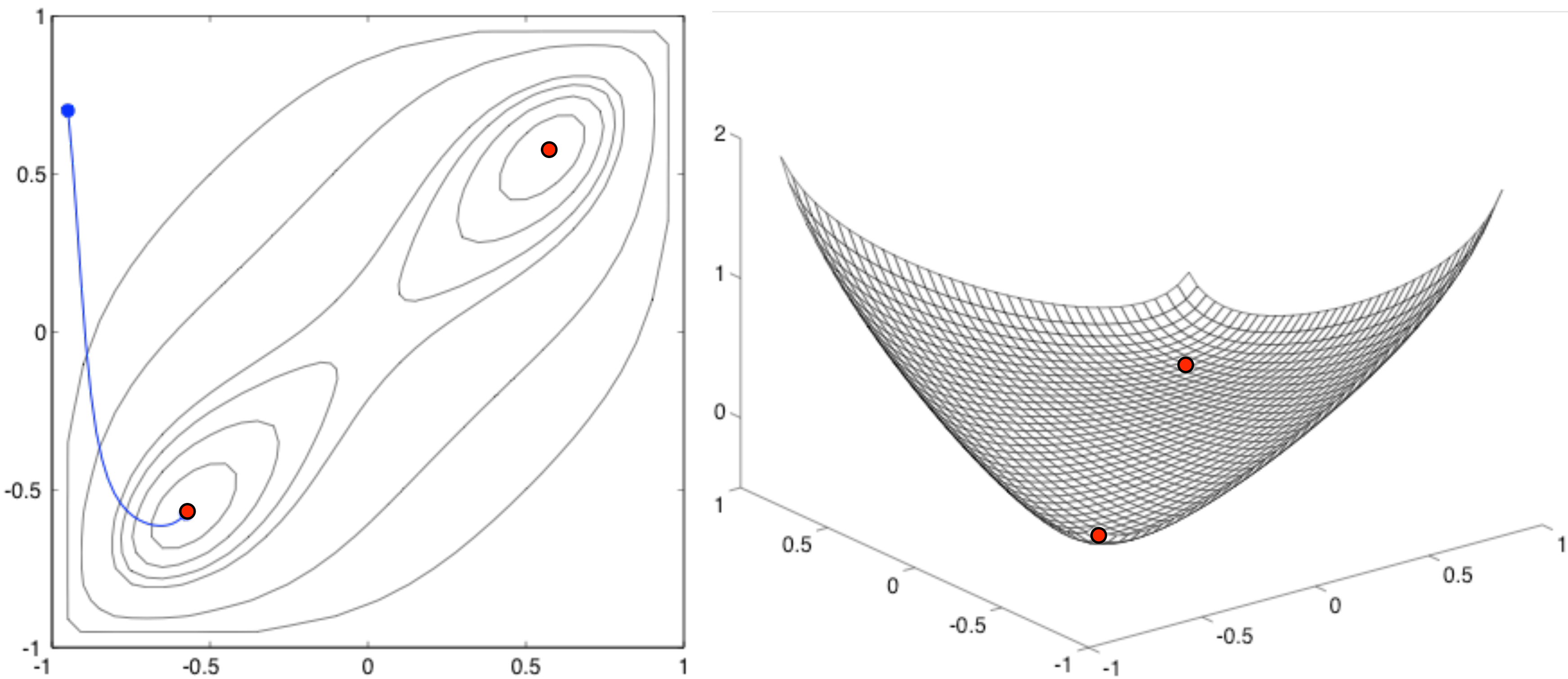
BSB Model again

- Iterative setting of weights from patterns (effectively gradient descent):

$$\Delta W_{ji} = \eta \left(f_{\mu j} - \sum_{k=1}^{N} w_{jk} f_{\mu k} \right) f_{\mu i}$$

- Here f_μ is the μ^{th} pattern, with the second indices indexing the bits of the pattern.

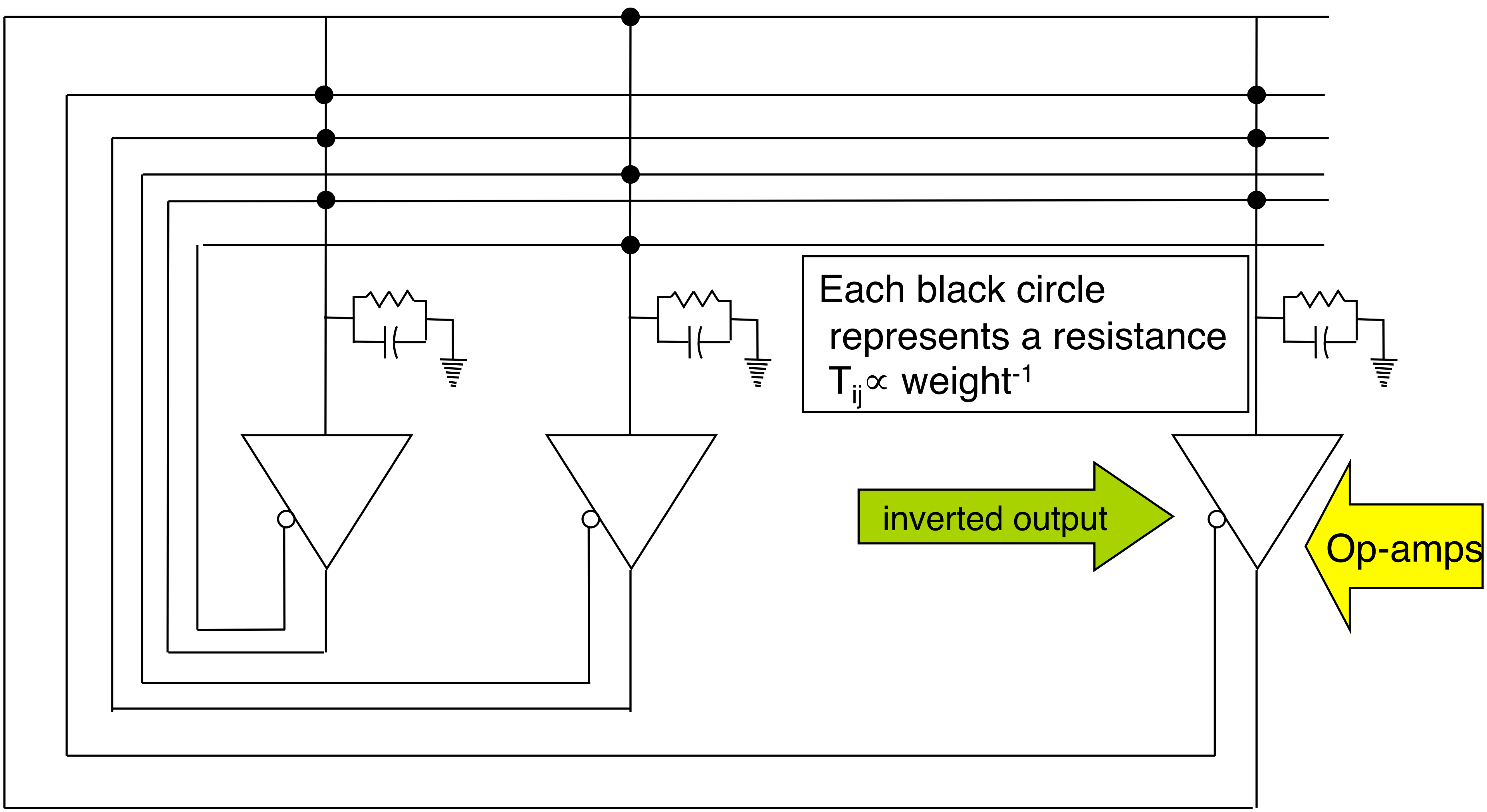
Attractors in a Continuous Analog of the Example



Lyapunov Functions

- For the continuous case, the energy function is called a Lyapunov function.
- The Hopfield network minimizes the value of the Lyapunov function.

Physical Realization of a Continuous Hopfield Net



Equations of Operation

$$C\frac{dn_i(t)}{dt} = \sum_{j=1}^{S} T_{i,j}a_j(t) - \frac{n_i(t)}{R_i} + I_i$$

n_i - input voltage to the ith amplifier
a_i - output voltage of the ith amplifier
C - amplifier input capacitance
I_i - fixed input current to the ith amplifier

$$|T_{i,j}| = \frac{1}{R_{i,j}} \qquad \frac{1}{R_i} = \frac{1}{\rho} + \sum_{j=1}^{S}\frac{1}{R_{i,j}} \qquad n_i = f^{-1}(a_i) \qquad a_i = f(n_i)$$

Commercial Success?

- At least one company, Attrasoft
  http://attrasoft.com/
  claimed to have products based on
  Hopfield nets and Boltzmann machines
  (to be discussed next).

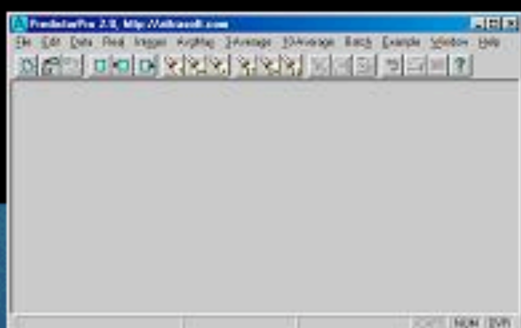
DataMining Products

Attrasoft PredictorPro 2.8

Attrasoft PredictorPro

**PredictorPro has 100,000 neurons.

- Predict indicators / indexes for strategic thinking & Policy issues
- Predict short & long terms interest rates
- Predict earning & revenue of a company
- Predict price flucuations for merchandise
- Predict commodity prices (ex. oil, corn, etc.)
- Predict the Stock Market (Attrasoft Dow 5 beats Dow Jones)



Representing Constraints

- Constraints on a solution tell you what you **cannot** do.
- Somehow represent these as **inhibitory** connections.

Finding Solutions to the TSP using a Hopfield Net

- Global minimum is not necessarily found (although this might be doable with a Boltzmann style algorithm / simulated annealing instead).

- The trick is to encode the instance of the TSP as a net with a specific energy function:

  minimal cost ↔ minimal energy

Traveling Salesperson Problem

- The problem is: given a set of n nodes ("cities") with a specified minimum cost between each pair of nodes, find a permutation ("tour") of the nodes that minimizes the summed costs between the nodes in the permutation sequence.

- The costs are symmetric, and the general problem does not require that there be any Euclidean relationship among the nodes.

TSP Formulation

- Represent a given problem as a matrix:
  - Cities correspond to rows.
  - Positions on the tour correspond to columns.
  - Example:

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

means B occurs first on the tour, D occurs second, A third, E fourth, C fifth.

TSP Formulation

- Assume {0, 1} values rather than {-1, 1}.
- The neurons correspond to entries in the matrix (n^2 neurons for n cities).
- Neurons in a row have **inhibitory** connections from other neurons in same row:
  - If one neuron is on, then others tend to be off, especially in minimum energy state.
- Similarly for neurons in the same column

TSP Formulation

- Need to favor tours that include *all* n cities, as opposed to just a subset of them.
- Need to represent costs between cities as neural weights:
  - Want to inhibit selection of adjacent cities in proportion to the cost between those cities.
  - Let X and Y be rows (cities) and i and j be columns (positions).

TSP Formulation

- Using the expression for energy in a Hopfield net $\Sigma\ \Sigma\ w_{ij}\ y_i y_j$, the corresponding energy is computed to have the form

$$A\ \Sigma_X \Sigma_{\ i} \Sigma_{j \neq i}\ y_{Xi} y_{Xj}$$

$$+\ B\ \Sigma_i\ \Sigma_X\ \Sigma_{Y \neq X}\ y_{Xi} y_{Yi}$$

$$+\ C\ (\Sigma_X \Sigma_{\ i}\ y_{Xi}\ -\ n)^2$$

$$+\ D\ \Sigma_i\ \Sigma_X \Sigma_{Y \neq X}\ c_{XY} y_{Xi}\ (y_{Y,i+1} + y_{Y,i-1})$$

- At energy minimum, only the last term, which represents the tour cost, is non-zero.
- We'll explain these terms one at a time.

$A \Sigma_X \Sigma_i \Sigma_{j \neq i} y_{Xi} y_{Xj}$

- The outer summation is over all cities X. The inner summations are over all pairs of distinct positions.
- There is a contribution of +1 if the same city occurs in more than one position in the tour.
- Therefore this term should ideally be 0.

$B \sum_i \sum_X \sum_{Y \neq X} y_{Xi} y_{Yi}$

- The outer summation is over all positions in the tour. The inner summations are over all pairs of distinct cities in position i.
- There is a contribution of +1 if the same position in the tour occurs more than once.
- Therefore this term should ideally be 0 also.

$C\,(\Sigma_X \Sigma_i\, y_{Xi} - n)^2$

- This term tries to guarantee that all cities get used. If the summation is n, the term is 0. If it is less than n, the term will be positive.
- This term should ideally be 0.

$D \Sigma_i \Sigma_X \Sigma_{Y \neq X} c_{XY} y_{Xi} (y_{Y,i+1} + y_{Y,i-1})$

- This term represents the cost of the tour. The outer sum is over all positions in the tour, the inner sums over all distinct pairs.

- c_{XY} represents the cost of going from X to Y. This term gets added provided X is at the i^{th} position in the tour, represented by $y_{Xi} = 1$, **and** Y is either at the $(i+1)^{th}$ or $(i-1)^{th}$ position (it can't be at both, by the other constraints). (i+1) and (i-1) are computed mod n.

Weight Derivation

- In order to get the energy function to come out as specified, choose the weight from Xi to Yj as

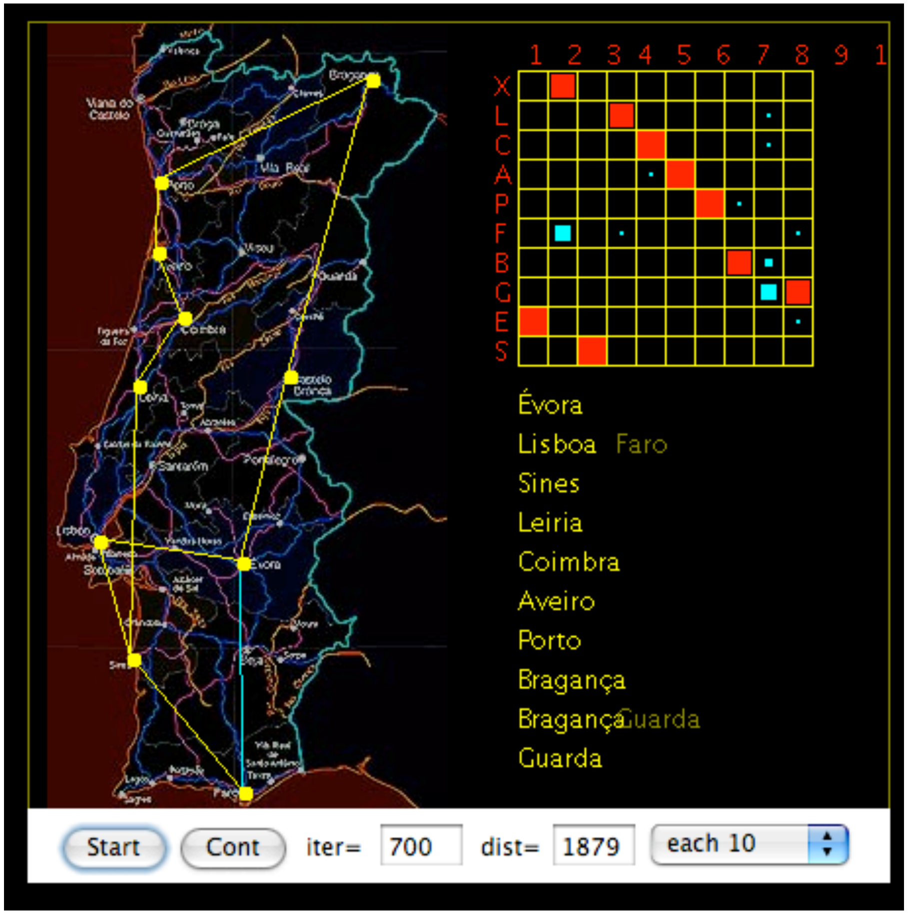
$$w_{XiYj} = -A\delta_{XY}(1-\delta_{ij}) - B\,\delta_{ij}(1-\delta_{XY}) - C - Dc_{XY}(\delta_{j,i+1} + \delta_{j,i-1})$$

for appropriate constants A, B, C, D.

- $\delta_{j,i}$ is the Kronecker delta (1 if i = j, 0 otherwise)

"Optimization" / Constraint Satisfaction Using Hopfield Nets (Hopfield and Tank, 1985)

http://to-campos.planetaclix.pt/neural/hope.html

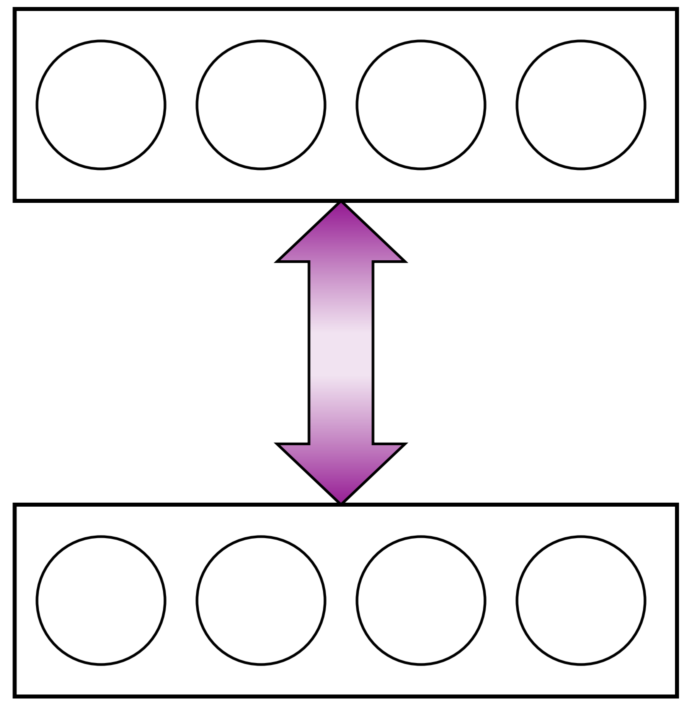


Related Topics

- Boltzmann machine
- Cauchy machine
- Helmholtz machine
- Willshaw nets

Bidirectional Associative Memories (BAM, Kosko 1988)

- Uses binary nodes (0 or 1)
- Symmetric weights
- Input and output layer
- Layers are updated in order, using threshold activation rule
- Nodes within a layer are updated *synchronously*



BAM

- BAM is a Hopfield network with two layers of nodes.
- Intra-layer weights are 0.
- These neurons are not dependent on each other (no mutual inputs).
- If updated synchronously, there is therefore no danger of increasing the network energy.

BAM Example

- Store the following associations:

  $(1,\ 1, -1, -1) \leftrightarrow (1,\ 1)$

  $(1,\ 1,\ 1,\ 1) \leftrightarrow (1, -1)$

  $(-1, -1, 1, 1) \leftrightarrow (-1,\ 1)$

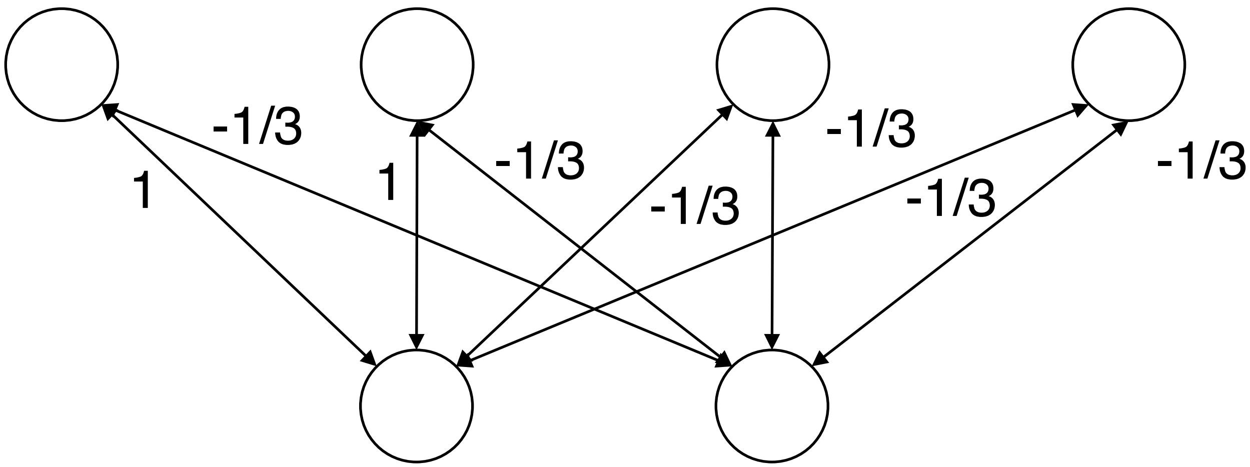
- Using the Hebb (outer-product) rule, weights are computed as:

  $(1,\quad 1,\quad -1/3, -1/3)$

  $(-1/3, -1/3, -1/3, -1/3)$

BAM Example

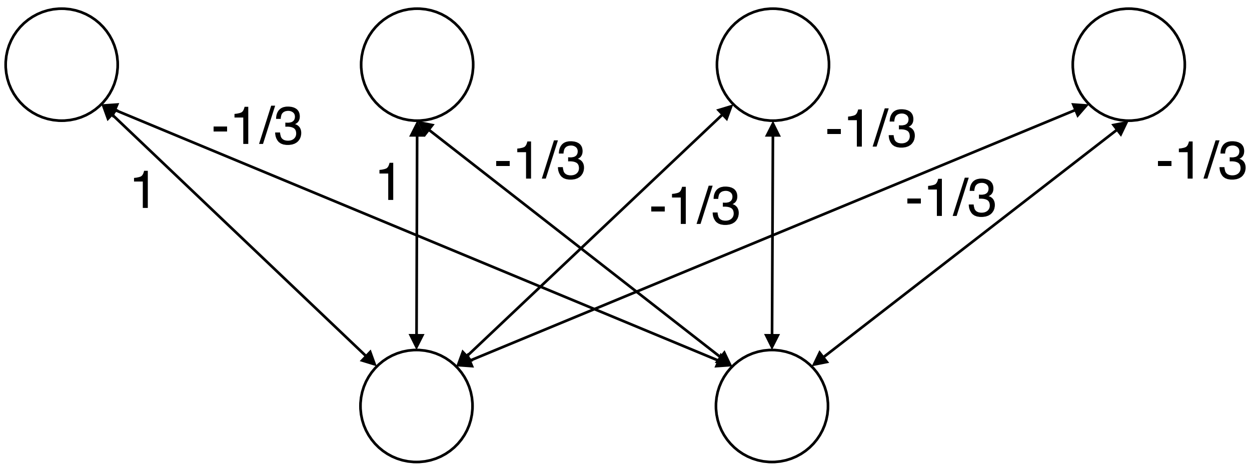
- The network is:



- Weight matrix:
  (1, 1, -1/3, -1/3)
  (-1/3, -1/3, -1/3, -1/3)

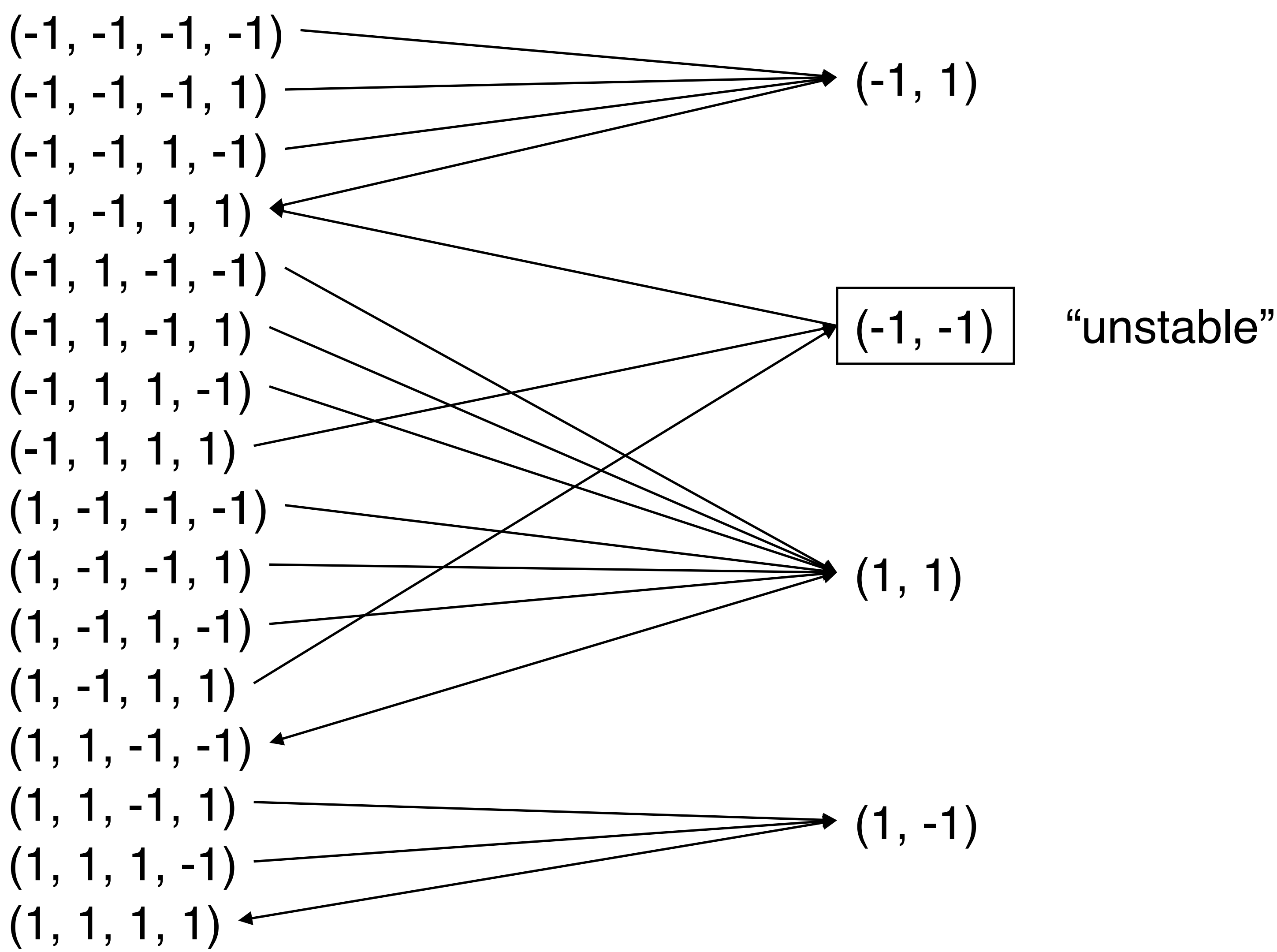
BAM Example

- The network is:



- Sample sequences:
  - (1, 1, 1, 1) → (1, -1)
  - (-1, -1, -1, -1) → (-1, 1)
  - (1, 1) → (1, 1, -1, -1)

BAM Behavior
(some arrows are bi-directional)



Neural "Movies"

- See Müller, Reinhardt, and Strickland, Neural Networks -- An Introduction, Springer, 1995.

- "… we have to study the properties of networks with **asymmetric** synaptic connections, because periodic activity cannot occur in the presence of thermal equilibrium, toward which all symmetric networks develop."

Neural "Movies"

- "One only needs to modify the Hebb rule in the following manner:

  $$w_{i\,j} = \Sigma\, p_{k\,i} * p_{(k+1)(\mathrm{mod}\ n)\ j}$$

  to get temporal periodicity."

  …

  "As a consequence, the network immediately makes a transition into the next pattern."