Chapter 10

Attractor Neural Networks

Neural Networks: A Classroom Approach
Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

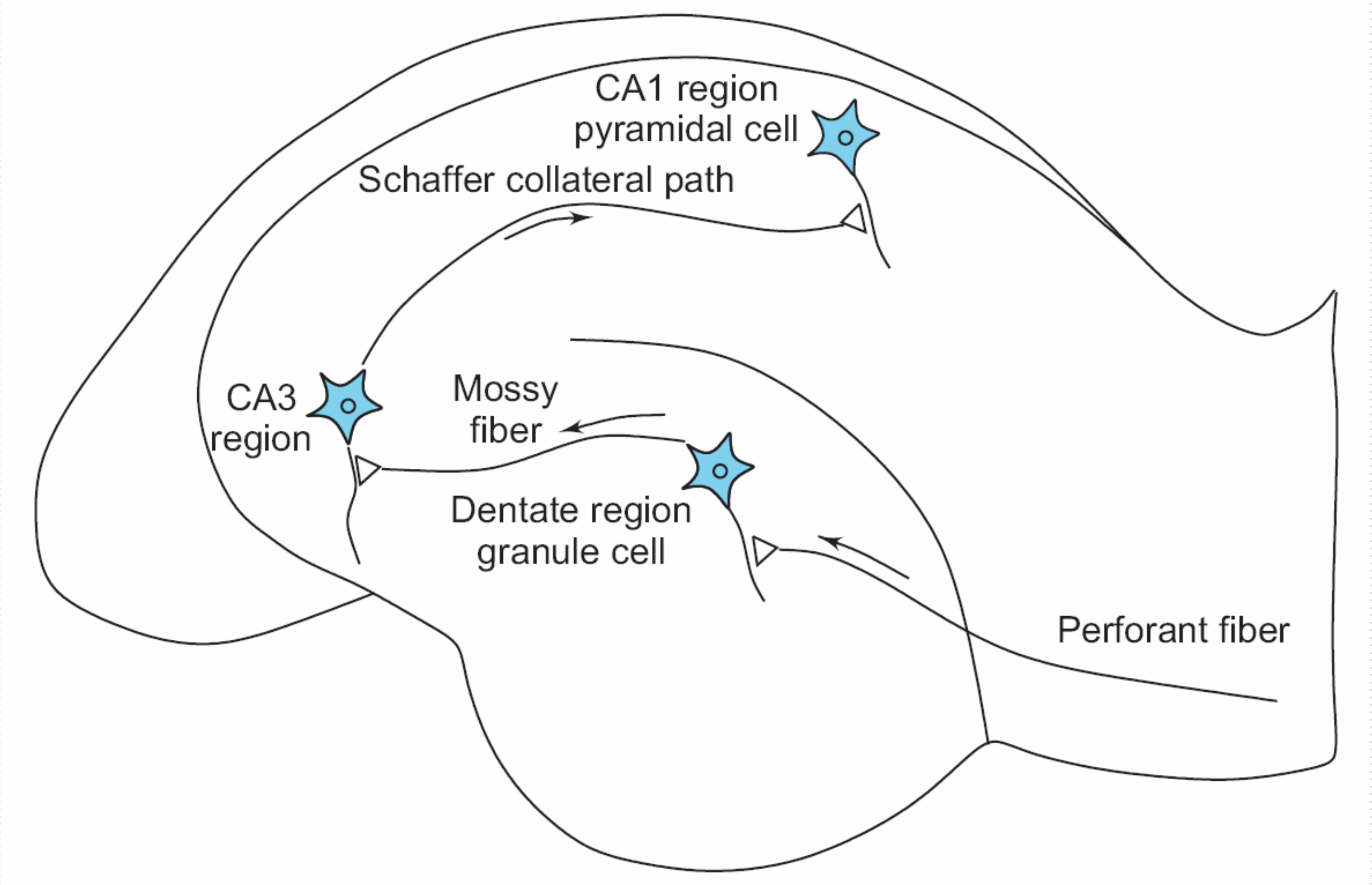
Human Memory is Associative

- We recall facts and information by invoking a chain of associations within a complex web of embedded knowledge.
- Chains of thought can be generated through conscious or unconscious reasoning
  - Fragrance of a flower evokes a pleasant memory of the past,
  - Name of a friend suddenly create a surge of emotions
- Entire experiences are recalled in complete richness of fact
  - on a simple cue
  - within fractions of a second

An Unanswered Question

- How do we accomplish such complex computations so effortlessly?
- Largely motivates the research currently being conducted on associative memory

Hippocampal Pathways Involved in Long-term Potentiation (LTP)



Three Major Pathways

- *Perforant fiber pathway* runs to granule cells in the *dentate region.*
- Granule cells in turn send axons that form the *mossy fiber pathway* to cells in the CA3 region.
- CA3 cells project axons onto dendrites of pyramidal cells in the CA1 region (*Schaffer collateral path*)

LTP

- Extensive research reveals that a brief high frequency stimulus train to any one of these three pathways leads to an enduring increase in the excitatory postsynaptic potential (EPSP) of CA1 pyramidal neurons.
- The strength of the EPSP increases in response to the stimulus train, and this change can last for anything from hours to weeks.

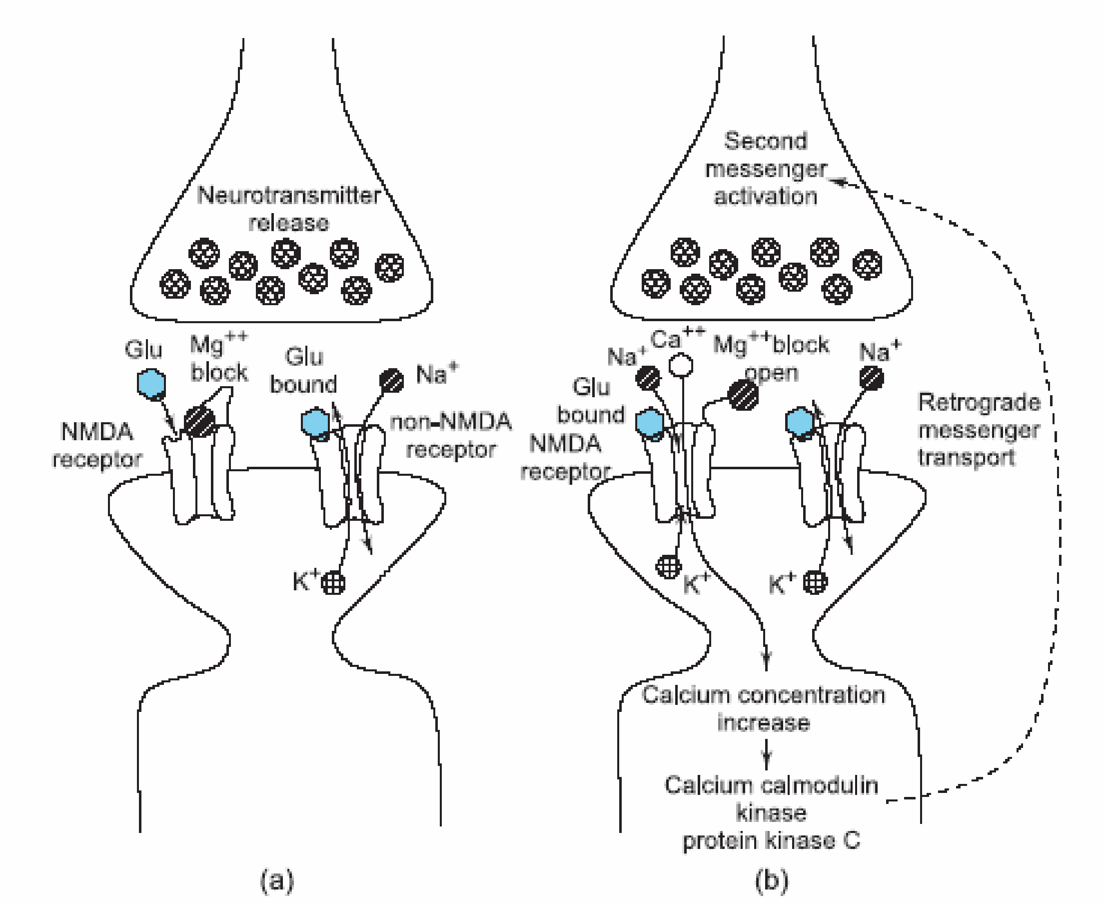
Three Important Properties of CA1 LTP

- Cooperativity:
  - CA1 LTP cannot be produced by activating only one fiber.
  - A minimum number of fibers must be activated together to achieve LTP.
- Associativity:
  - When both strong and weak excitatory inputs arrive in the same dendritic region of a pyramidal cell, the weak input gets potentiated if it is activated in association with the strong one.
- Specificity:
  - LTP is specific to the dendrites where it is produced.

Hebb's Postulate

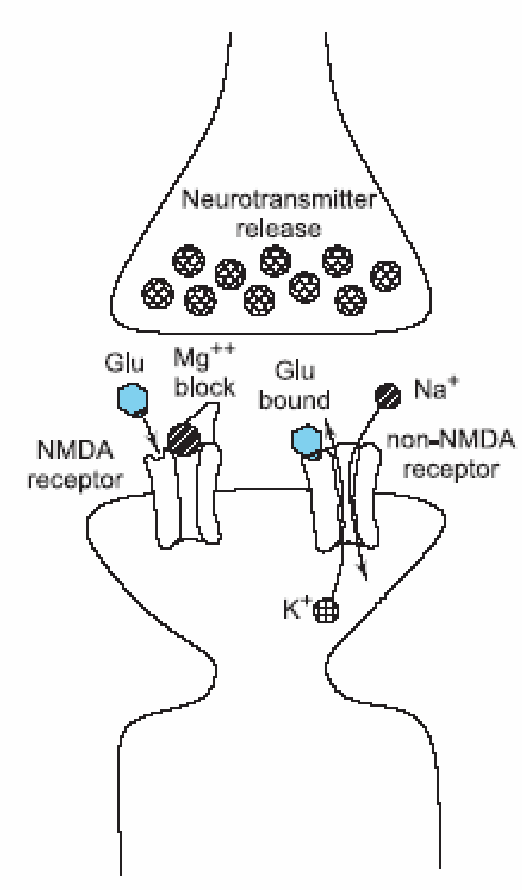
- "When an axon of cell A excite[s] cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells so that A's efficacy as one of the cells firing B is increased."

NMDA Synapse as a Model for LTP



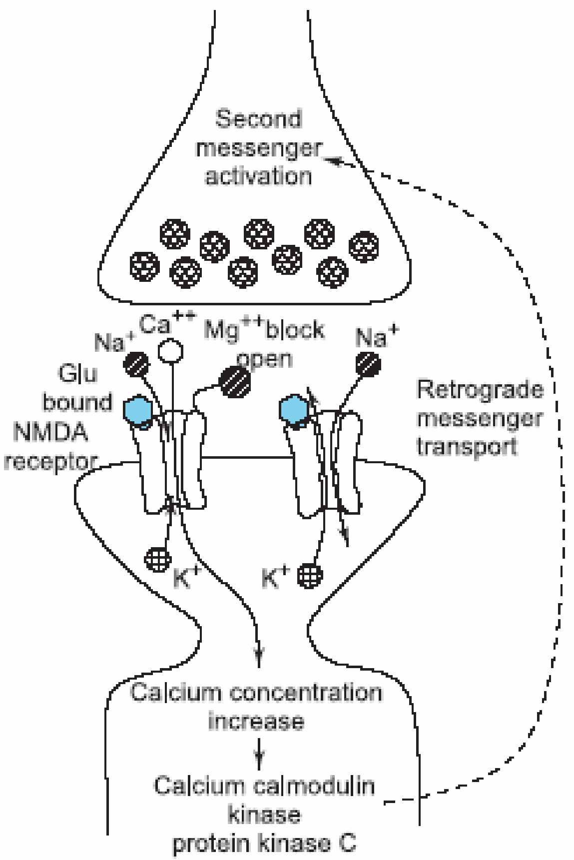
NMDA and Non-NMDA Receptors

- Axons from CA3 neurons that terminate on CA1 neurons use glutamate as their neurotransmitter.
- CA1 neurons have both NMDA and non-NMDA receptors to which glutamate binds.
- The NMDA receptor channel is a uniquely double gated channel: it is usually blocked by Mg++ ions
- Initially the channel remains blocked even when glutamate binds to it.



Unblocking an NMDA Channel

- Magnesium ions decouple from the channel only if the postsynaptic cell is sufficiently depolarized.
- An NMDA channel can become unblocked only when
  - glutamate binds to the receptor, and
  - the postsynaptic cell is sufficiently depolarized by strong cooperative inputs from the presynaptic neurons.
- Depolarization is achieved only when many non-NMDA receptors open in response to the simultaneous firing of many presynaptic inputs since unblocking these receptors allows an influx of Na+.



(b)

Induction of LTP

- Evidence suggests that the influx of calcium comes through NMDA channels and *not* from the regular voltage gated calcium channels.
- Activation of non-NMDA channels depolarizes the dendritic spines and this removes the magnesium ion block which allows calcium to enter.
- Synaptic action is thus restricted to synapses that are active.
- The influx of calcium initiates the enduring enhancement of synaptic transmission by activating two calcium dependent protein kinases called
  - *calmodulin kinase*
  - *protein kinase C*
- These protein kinases become persistently active.
- This first stage is called the *induction* of LTP

Maintenance of LTP

- Requires enhanced presynaptic transmitter release.
- Induction of LTP requires a postsynaptic event
- Maintenance requires a presynaptic event!
- Some message goes back from the post to presynaptic neuron.
- Carried by a calcium activated *retrograde transmitter* (believed to be nitric oxide) that diffuses from the post to presynaptic neuron.
- In the presynaptic neuron, the retrograde transmitter activates one or more *second messengers* in the presynaptic terminal that enhances transmitter release of neurotransmitter. This *maintains* LTP.

Attractor Neural Network Associative Memory

- Hebbian learning principle implies that synaptic dynamics are governed by a conjunction of pre and postsynaptic activities.
- Leads to powerful associative memories that relate or *associate* one concept or idea with another

Memories are Minima of a Lyapunov Function

- Dynamics of the system can be visualized by imagining some high dimensional undulating *energy surface* generated from a Lyapunov function
- Gravitational dynamics: Rubber sheet analogy
- Neural network dynamics: state of the network maps to a point on the Lyapunov energy surface
  - Each point on the energy surface corresponds to point in state space
- An attractor neural network falls to an energy state that represents the closest local minimum under the influence of the governing vector field

Neural Network Dynamics

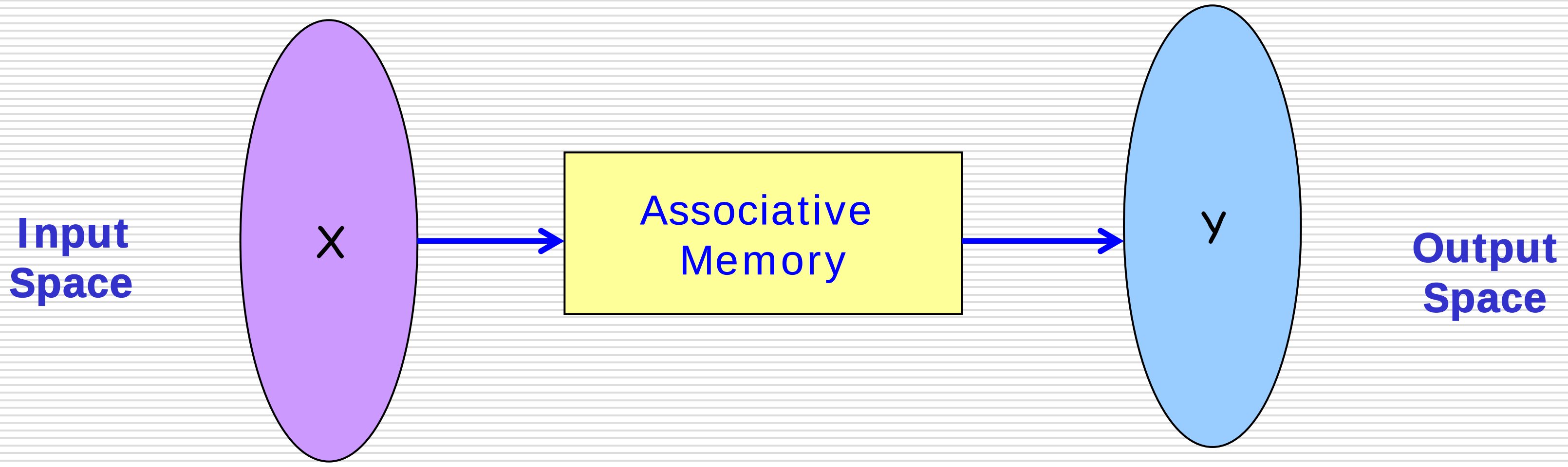
- Vector field governed by
  - structure of connections
  - nature of the signal function
- Network states evolve in time until the local minimum is reached, after which the states stop evolving further
- Guaranteed for systems that follow the Cohen-Grossberg structure:
  - Admit a Lyapunov function that is bounded below and whose time derivative is negative definite

Broad Objective

- Point of local minimum at which the network state stabilizes is to be interpreted as a memory
- Given a cue or an initial input vector, the initial state can be represented by a point in the energy surface
- Network state evolves in time till the system hits the energy minimum
- Final state is the recalled memory vector
- *Therefore, the minimum energy points of the Lyapunov surface have to be mapped to desired memory states*

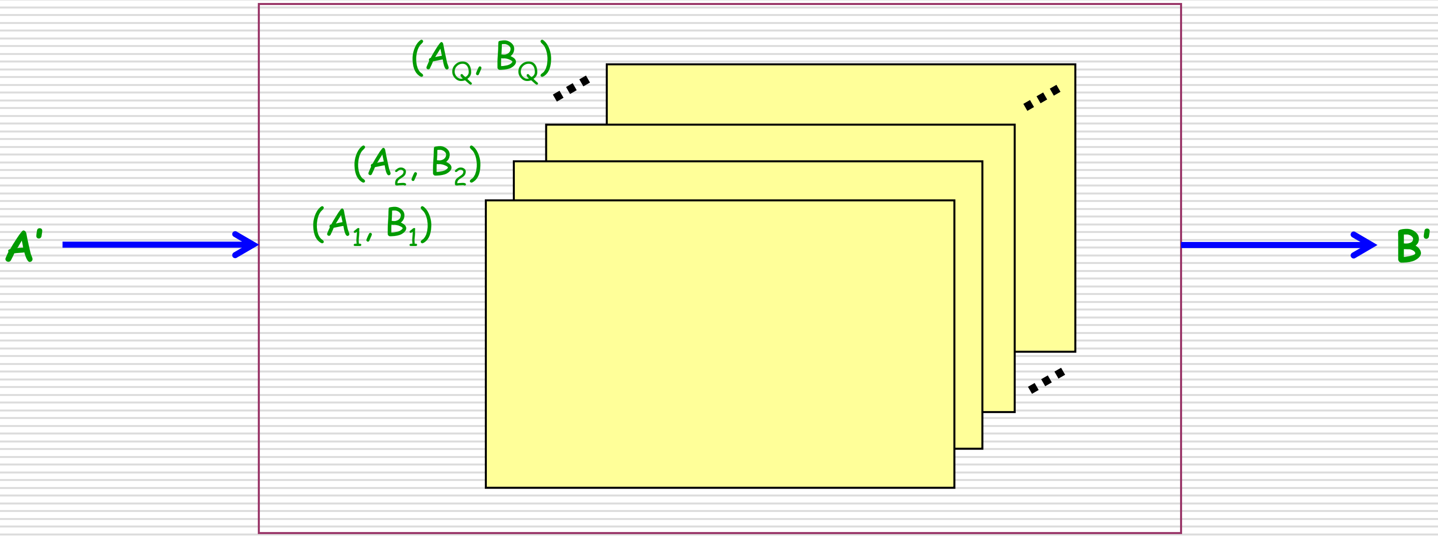
The Associative Memory Model

- Mapping from unknown domain points to known range points



- The memory learns an underlying association from a training data set

Matrix Associative Memory

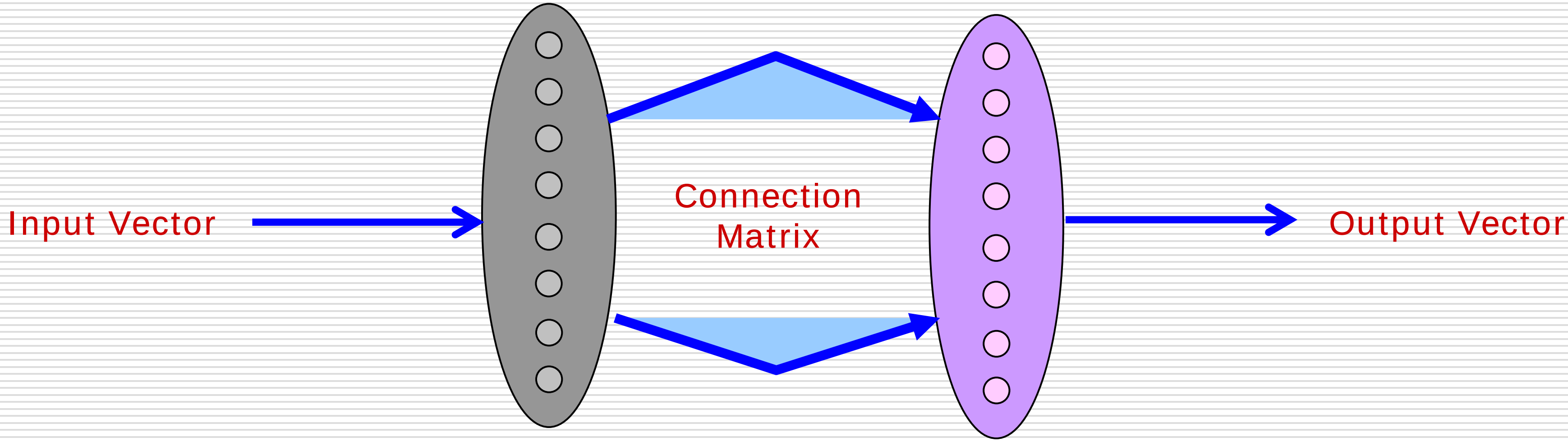


Encodes associations $\{A_i, B_i\}^Q_{i=1}$ $A_i \in B^n$, $B_i \in B^m$ into a connection matrix

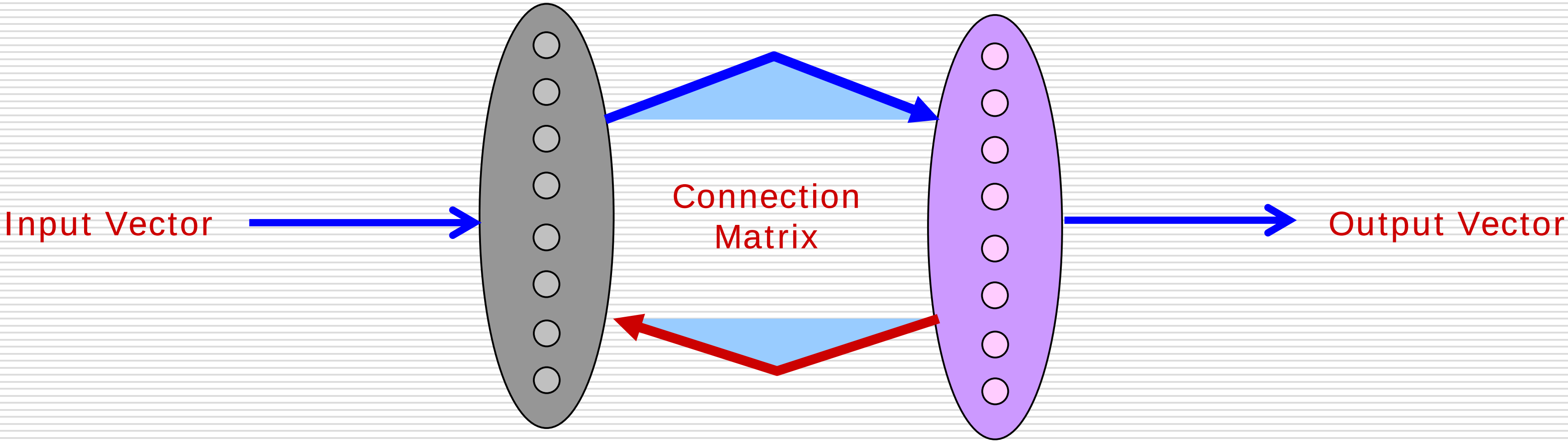
Hetero- and Auto-associative Memory

- When A_i, B_i are in different spaces $A_i \in B^n$, $B_i \in B^m$ the memory is heteroassociative
- When A_i, B_i are in the same space A_i, $B_i \in B^n$ the memory is autoassociative

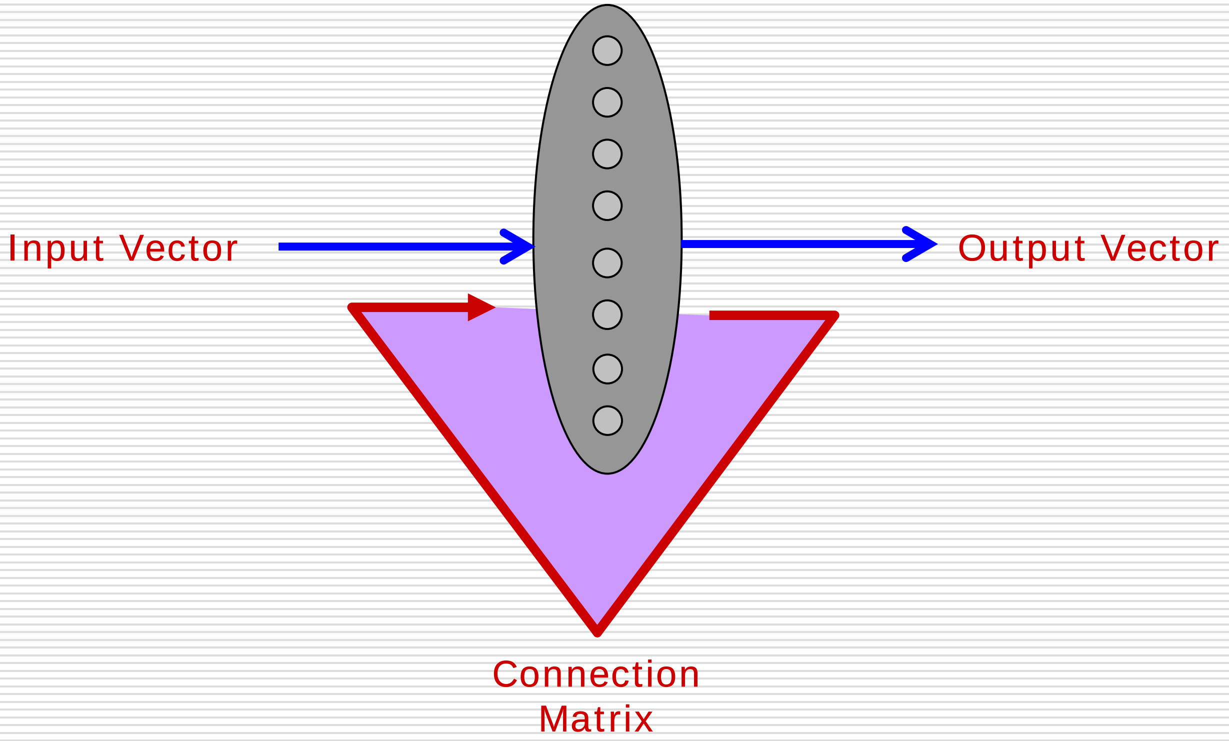
Two Layer Feedforward Neural Network Associative Memory



Two Layer Feedback Neural Network Associative Memory



One Layer Feedback Neural Network Associative Memory



Hebb Encoding Scheme

- Consider two layers F_x, F_y of n,m neurons with connection matrix W
- Weight matrix is constructed using Generalized Hebb Rule
  - modifies synapses in accordance with the product of the presynaptic activity with the activity of the postsynaptic neuron

$$\delta w_{ij} = \eta a_i b_j$$

$$\mathbf{W} = \begin{pmatrix} a_1 b_1 & \cdots & a_1 b_m \\ \vdots & \ddots & \vdots \\ a_n b_1 & \cdots & a_n b_m \end{pmatrix} = AB^T$$

Linear Associative Memory

- Assume that a single association (A,B) has been encoded into the connection matrix **W**, using the outer product

$$\mathbf{W} = AB^T$$

- Presentation of the vector A yields perfect recall

$$(B')^T = A^T\mathbf{W} = A^T A B^T = \|A\|^2 B^T$$

Linear Associative Memory

- Encoding more than one association…
  - superpose the individual association matrices.
- For Q associations
  - $\{Ai, Bi\}$, $A_i \in \mathbf{B}^n$, $B_i \in \mathbf{B}^m$

$$\mathbf{W} = \sum_{k=1}^{Q} A_k B_k^T$$

Orthogonal Linear Associative Memory (OLAM)

- A special (and very restrictive) case arises if the vectors A_k are *orthonormal*
  - orthogonal to one another
  - normalized to unity magnitude

$$
\begin{aligned}
(B')^T &= A_i^T \mathbf{W} \\
&= A_i^T \sum_{k=1}^{Q} A_k B_k^T \\
&= A_i^T A_i B_i^T + \sum_{k \neq i} A_i^T A_k B_k^T \\
&= \| A_i \|^2 B_i^T + 0 \\
&= B_i^T
\end{aligned}
$$

Orthogonal Linear (One-shot) Associative Memories: Example

- Encode the vector associations
  - $A_1 = (1\ 0\ 0\ 0)$ $B_1 = (1\ 2\ 3)$
  - $A_2 = (0\ 1\ 0\ 0)$ $B_2 = (-2\ 3\ 1)$
  - $A_3 = (0\ 0\ 1\ 0)$ $B_3 = (4\ 0\ 4)$

$$\mathbf{W} = A_1 B_1^T + A_2 B_2^T + A_3 B_3^T$$

$$= \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ -2 & 3 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

Orthogonal Linear (One-shot) Associative Memories: Example

- It is easy to verify that A_1, A_2, A_3 are fixed points of the system

$$A_1^T \mathbf{W} = (1\ 0\ 0\ 0) \begin{pmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix} = (1\ 2\ 3) = B_1^T$$

- Verify for A_2, A_3

Orthogonal Linear (One-shot) Associative Memories: Example

- The addition of input vectors yields the addition of corresponding associants

$$(A_1 + A_2)^T \mathbf{W} = (1\ 1\ 0\ 0) \begin{pmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

$$= (-1\ 5\ 4) = B^T = (B_1 + B_2)^T$$

Orthogonal Linear (One-shot) Associative Memories: Example

- Input vectors close to a memory recall a vector close to its associant

$$A^T\mathbf{W} = (.9\ .1\ .1\ .1)\begin{pmatrix} 1 & 2 & 3 \\ -2 & 3 & 1 \\ 4 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

$$= (1.1\ \ 2.1\ \ 3.2) = B^T \simeq B_1^T$$

More Questions…

- What is the nature of the dynamics that underlies these models?
- How do we analyze the stability of these models?
- What are the important pathologies that arise in Hebb encoded matrix memories?
- What are the capacities of these memories?

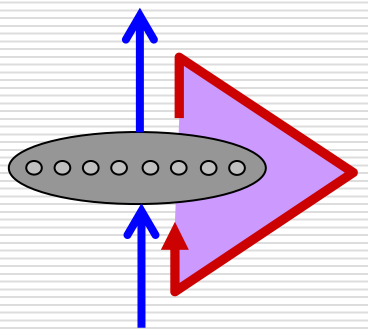
Hopfield Network

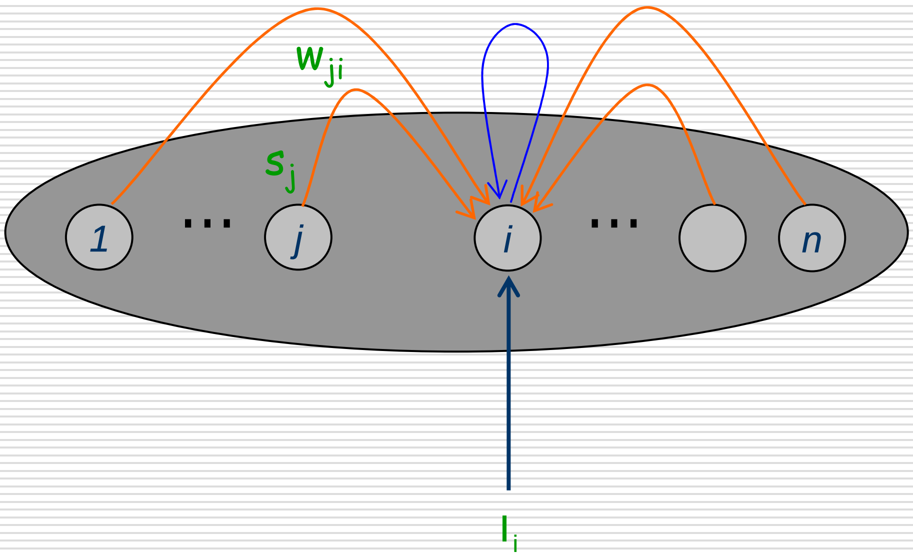
- Popularized by Nobel Laureate John Hopfield
- Basic idea:
  - Map stable states to correspond to certain desired memory vectors or to solutions of an optimization problem
  - Then the time evolution of dynamics leads to a stable state where the outputs of the network correspond to the answer

Fundamental Issues

- How do we “program” the solutions of the problem into stable states of the network?
- How do we ensure that the feedback system designed is stable?

Hopfield Network Architecture





$$\dot{x}_i = \underbrace{-A_i x_i}_{\text{Passive decay}} + \underbrace{\sum_{j=1}^{n} w_{ji} \mathcal{S}_j(x_j)}_{\text{Signal feedback}} + \underbrace{I_i}_{\text{External input}}$$

Neuron Characteristic and Inverse

- Assume that individual neurons have distinct sigmoidal characteristics

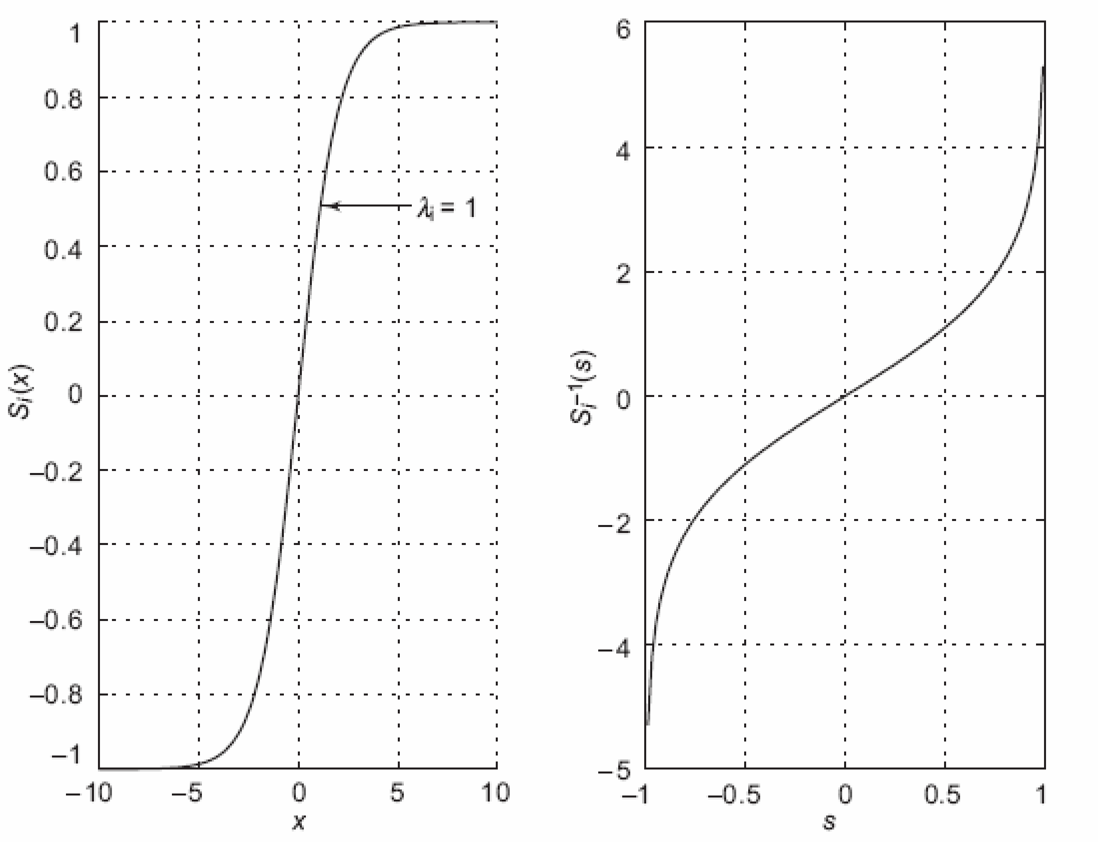
$$\mathcal{S}_i(x) = \frac{1 - e^{-\lambda_i x}}{1 + e^{-\lambda_i x}}$$

- With inverse

$$\mathcal{S}_i^{-1}(s) = \frac{1}{\lambda_i} \ln\left(\frac{1+s}{1-s}\right) = \frac{1}{\lambda_i} \mathcal{S}^{-1}(s)$$

$$s = \mathcal{S}(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Neuron Characteristic and Inverse

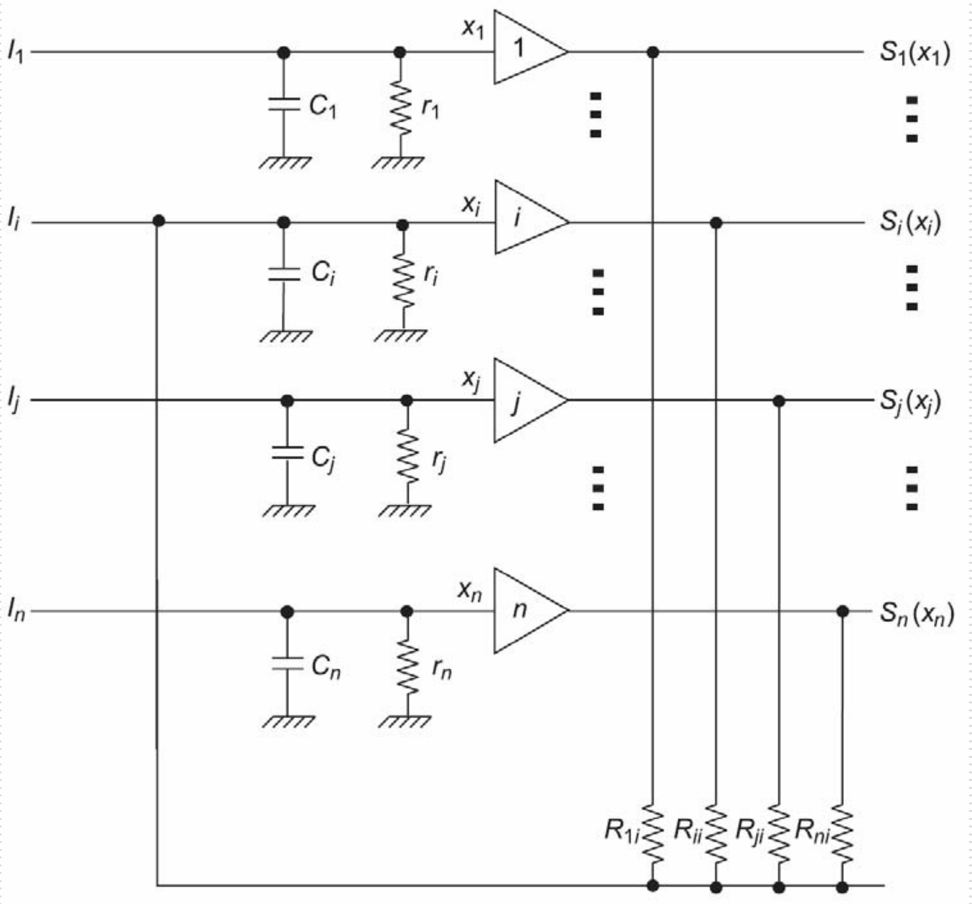


Notes on the Hopfield Network

- The system represents a high dimension cross-coupled non-linear dynamical system
  - Difficult to find find a closed form solution.
- In the original Hopfield network the weight w_{ii} is usually set to zero
  - Neurons do not feed back signals to themselves
- The model under consideration has Cohen-Grossberg form which necessarily dictates that connections be symmetric for stability

Electrical Interpretation of Additive Dynamics

- The circuit has n amplifiers which feedback currents through resistances R_{ji}
- Each amplifier has
  - Input capacitor C_i
  - Input resistance r_i
  - Input current I_i



Electrical Circuit Analysis

- Kirchhoff current law equation at the input node of the i^{th} amplifier

$$C_i \dot{x}_i + \frac{x_i}{r_i} = \sum_{j=1}^{n} \frac{(\mathcal{S}_j(x_j) - x_i)}{R_{ji}} + I_i$$

- Which is easily rearranged as

$$\dot{x}_i = -\frac{x_i}{R_i C_i} + \sum_{j=1}^{n} \frac{\mathcal{S}_j(x_j)}{C_i R_{ji}} + \frac{I_i}{C_i}$$

$$\frac{1}{R_i} = \frac{1}{r_i} + \sum_{j=1}^{n} \frac{1}{R_{ji}}$$

Hopfield Model has Cohen-Grossberg Form

- Cohen-Grossberg dynamics have the general form

$$\dot{x}_i = a_i(x_i)\left(b_i(x_i) - \sum_{j=1}^{n} c_{ji}\, d_j(x_j)\right) \quad i = 1, \ldots, n$$

  - where $a_i(x_i) \geq 0$, $w_{ij} = w_{ji}$, $d_j'(x_j) > 0$
- These models admit the Lyapunov function

$$E = -\sum_{i=1}^{n} \int_0^{x_i} b_i(\alpha_i)\, d_i'(\alpha_i)\, d\alpha_i + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{jk}\, d_j(x_j)\, d_k(x_k)$$

  - $\Rightarrow$ Global asymptotic stability

Hopfield Model has Cohen-Grossberg Form

- The Hopfield model can be re-arranged into Cohen-Grossberg form with straightforward substitutions

$$\dot{x}_i = (-A_i x_i + I_i) + \sum_{j=1}^{n} w_{ji}\, \mathcal{S}_j(x_j)$$

$$\downarrow$$

$$a_i(x_i) = 1$$

$$b_i(x_i) = -A_i x_i + I_i$$

$$c_{ji} = -w_{ji}$$

$$d_j(x_j) = \mathcal{S}_j(x_j) = s_j$$

Hopfield Model Lyapunov Function

- Easily derived from the Cohen-Grossberg Lyapunov function using the aforesaid substitutions

$$E = -\frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} w_{jk} s_j \, s_k$$

$$+\sum_{i=1}^{n}\frac{A_i}{\lambda_i}\int_0^{s_i} \mathcal{S}^{-1}(\beta_i)\, d\beta_i - \sum_{i=1}^{n} I_i s_i$$

Stability Analysis in Continuous Time

- Need to show that the energy derivative is strictly negative
- From the chain rule of calculus,

$$\dot{E} = \sum_{i=1}^{n} \frac{\partial E}{\partial s_i} \dot{s}_i$$

Stability Analysis in Continuous Time

- From the additive dynamics equations

$$\frac{\partial E}{\partial s_i} = -\sum_{j=1}^{n} w_{ji} s_j + A_i x_i - I_i$$
$$= -\dot{x}_i$$

- Which yields $$\dot{E} = -\sum_{i=1}^{n} \dot{x}_i \dot{s}_i$$

- or $$\dot{E} = -\sum_{i=1}^{n} \frac{d}{dt}(\mathcal{S}_i^{-1}(s_i))\,\dot{s}_i$$

$$= -\sum_{i=1}^{n} (\mathcal{S}_i^{-1}(s_i))'\,(\dot{s}_i)^2$$
$$< 0$$

Lyapunov Energy for Hopfield Network with High Gain Neurons

- Binary threshold signal function is actually a limiting case of the sigmoidal function as $\lambda \to \infty$
- Lyapunov energy function reduces to

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i s_j - \sum_{i=1}^{n} I_i s_i$$

- In the absence of external inputs

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i s_j = -\frac{1}{2} S^T \mathbf{W} S$$

Neuron Switching in High Gain

- A high gain neuron switches its state from -1 to 1 or from 1 to -1 at discrete intervals of time by sampling its current activation.

$$x_i^{k+1} = \sum_{j=1}^{n} w_{ji} \mathcal{S}(x_j^k) + I_i$$

$$s_i^{k+1} = \mathcal{S}(x_i^{k+1}) = \begin{cases} 1 & \text{if} \quad x_i^{k+1} > 0 \\ -1 & \text{if} \quad x_i^{k+1} < 0 \\ \mathcal{S}(x_i^k) & \text{if} \quad x_i^{k+1} = 0 \end{cases}$$

Stability Analysis in Discrete Time

- Energy function at time instant k is

$$E_k = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i^k s_j^k - \sum_{i=1}^{n} I_i s_i^k$$

- Define the change in energy from time instant k to k+1 as $\Delta E_k = E_{k+1} - E_k$
- Assume only the I^{th} neuron changes state (without any loss of generality)

Stability Analysis in Discrete Time

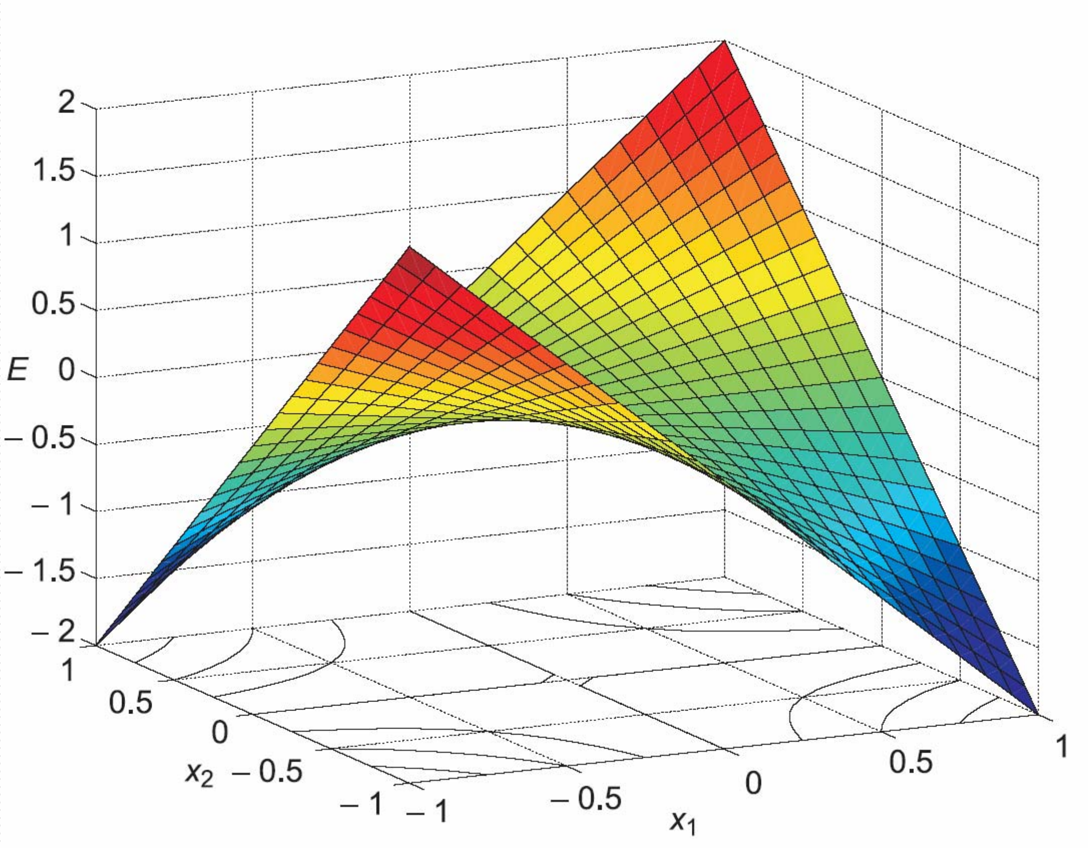
$$
\begin{aligned}
\Delta E_k &= E_{k+1}^{(I)} - E_k^{(I)} \\
&= -s_I^{k+1} \sum_{j=1}^{n} w_{jI} s_j^{k+1} - I_I s_I^{k+1} + s_I^k \sum_{j=1}^{n} w_{jI} s_j^k + I_I s_I^k \\
&= -\left(\sum_{j=1}^{n} w_{jI} s_j^k + I_I \right) \left(s_I^{k+1} - s_I^k \right) \\
&= -x_I^{k+1} \Delta s_I^k \\
&< 0
\end{aligned}
$$

Global Asymptotic Stability

- The energy E is a function of states and decreases monotonically in time.
- In accordance with Lyapunov's Theorem (Chapter 9) the system is therefore globally asymptotically stable.

Example

- Energy function for a Hopfield network that encodes two vectors (-1,1) and (1,-1)
- *The network relaxes to an attractor in whose basin of attraction the initial state lies.*



Content Addressable Memory

- Example: recall a binary number 11001010 based on input 11001000 with one bit distortion
- Key to the correct memory vector is the partially distorted input data itself
- Recall is essentially a clean up operation on a partially distorted input
- CAMs solve the completion problem

Hamming Distance Based Recall

- If
  - the Hamming distance of the input is such that the point lies within the basin of attraction of the memory which is represented by a distorted input
- Then
  - we expect that the correct memory should get recalled

Encoding Memories via Outer Products

- Bipolar outer product encoding

$$\mathbf{W} = \sum_{k=1}^{Q} X_k X_k^T$$

- Where $X_k = 2A_k - 1$, weight matrix $\mathbf{W}$ is symmetric, $\mathbf{W} = \mathbf{W}^T$
- Original model

$$\mathbf{W} = \sum_{k=1}^{Q} X_k X_k^T - Q\mathbf{I}$$

Zero off the diagonal elements

Avoids identity operator type behaviour

Asynchronous Neuron Update

- Standard Hopfield network update is always asynchronous *and* *deterministic*
  - A neuron is randomly selected for update at an instant of time.
  - The neuron activation transforms to the new signal using the deterministic signal function
- The number of times any neuron gets a chance to update is the same *on average*
- Serial asynchronous update guarantees that the network converges to a fixed point equilibrium

Synchronous Neuron Update

- Activations of all neurons are calculated at an instant of time
- All neurons update their signal values together to generate the signal vector at the next instant of time
- Under a synchronous update the Hopfield network approaches either a fixed point equilibrium or a two-step limit cycle

Operational Summary of the Hopfield CAM algorithm

Given	A set of binary vectors $\{A_i\}_{i=1}^{Q}$ to be encoded into a Hopfield CAM using bipolar encoding and decoding.
Encode	↬ $\mathbf{W} = \sum_{k=1}^{Q} X_k X_k^T - Q\mathbf{I}$
Initialize	↬ Set-up neuron signals vector to the probe vector X_0.

Operational Summary of the Hopfield CAM algorithm

Iterate ⭮Repeat

{

⇝ Compute the neuron activations $y_i^{k+1} = \sum_{j=1}^{n} w_{ji} s_j^k \quad i = 1, \ldots, n$

⇝ Select a neuron I at random from the n neurons in the field

⇝ Update the neuron in accordance with the signal function

$$s_I^{k+1} = \mathcal{S}(y_I^{k+1}) = \begin{cases} 1 & \text{if} \quad y_I^{k+1} > 0 \\ -1 & \text{if} \quad y_I^{k+1} < 0 \\ \mathcal{S}(y_I^k) & \text{if} \quad y_I^{k+1} = 0 \end{cases}$$

} until (signals of all neurons no longer change.)

MATLAB Code to Implement Hopfield CAM

```
mem_vectors = [
1 1 0 0 0 0
0 0 0 0 1 1
0 0 1 1 0 0];
q = size(mem_vectors,1);% number of vectors
n = size(mem_vectors,2);% dim of the vectors
bip_mem_vecs = 2*(mem_vectors) - 1; %
        Convert

% Initialize and compute the weight matrix
zd_wt_mat = zeros(n,n);
for i=1:q
  zd_wt_mat = zd_wt_mat +

        bip_mem_vecs(i,:)'*bip_mem_vecs(i,:);
end
zd_wt_mat = zd_wt_mat - q*eye(n);% Zero diag

probe = input('Enter the probe vector: ');
signal_vector = 2*probe-1;
```

```
flag = 0; % Initialize flag

while flag ~= n
   permindex = randperm(n);% Randomize order
   old_signal_vector = signal_vector;

for j = 1:n % update all neurons once per epoch
  act_vec = signal_vector * zd_wt_mat;
  if act_vec(permindex(j)) > 0
    signal_vector(permindex(j)) = 1;
  elseif act_vec(permindex(j)) < 0
    signal_vector(permindex(j)) = -1;
  end
end

flag = signal_vector*old_signal_vector;
end

disp('The recalled vector is ')
0.5*(signal_vector + 1)
```

Example 1

- Encode, recall, analyze stability
  - 110, 001

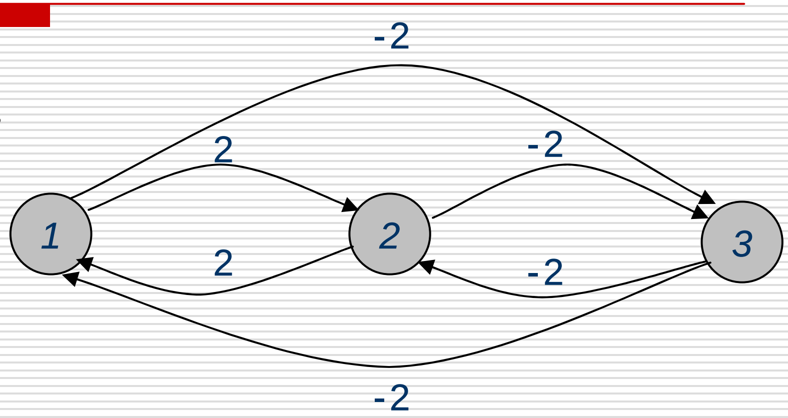
$$\mathbf{W} = \sum_{k=1}^{2} X_k X_k^T - 2\mathbf{I}$$

$$= \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} (1\ 1\ -1) + \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} (-1\ -1\ +1) - 2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 2 & -2 \\ 2 & 0 & -2 \\ -2 & -2 & 0 \end{pmatrix}$$

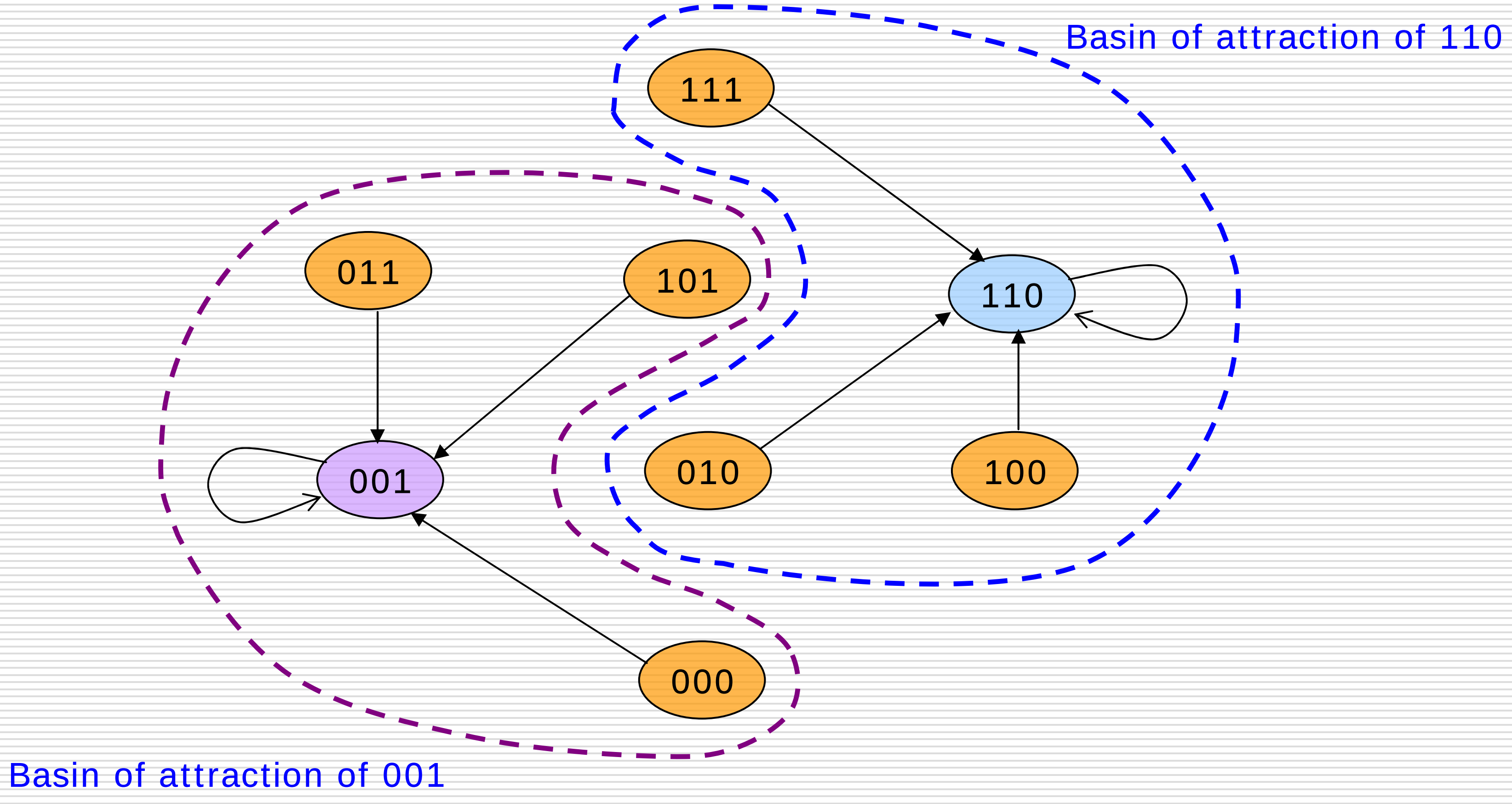
Example 1

- Network architecture



Probe			Initial Energy	Final state			Final Energy
−1	−1	−1	4	−1	−1	1	−12
−1	−1	1	−12	−1	−1	1	−12
−1	1	−1	4	1	1	−1	−12
−1	1	1	4	−1	−1	1	−12
1	−1	−1	4	1	1	−1	−12
1	−1	1	4	−1	−1	1	−12
1	1	−1	−12	1	1	−1	−12
1	1	1	4	1	1	−1	−12

Basins of Attraction



Multiple Encoding

- When taking outer products note that $X_k X_k^T = (-X_k)(-X_k^T) = X_k^c (X_k^c)^T$
- In our example 100 and 011 are complements!

$$\begin{aligned} \mathbf{W} &= X_1 X_1^T + (X_1^c)(X_1^c)^T - 2\mathbf{I} \\ &= X_1 X_1^T + (-X_1)(-X_1^T) - 2\mathbf{I} \\ &= 2X_1 X_1^T - 2\mathbf{I} \end{aligned}$$

- Leads to multiple encoding

$$\mathbf{W} = \sum_{k=1}^{Q} p_k X_k X_k^T - Q\mathbf{I}$$

Example 2: Limit Cycles

- Now encode 110, 000

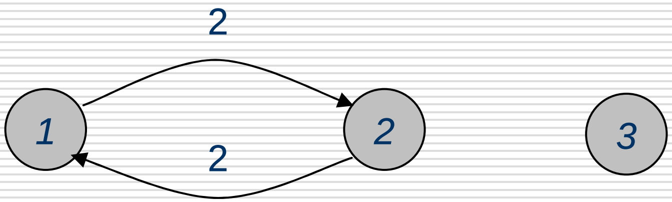
$$\mathbf{W} = \sum_{k=1}^{2} X_k X_k^T - 2\mathbf{I}$$

$$= \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & -1 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 \end{pmatrix} - 2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

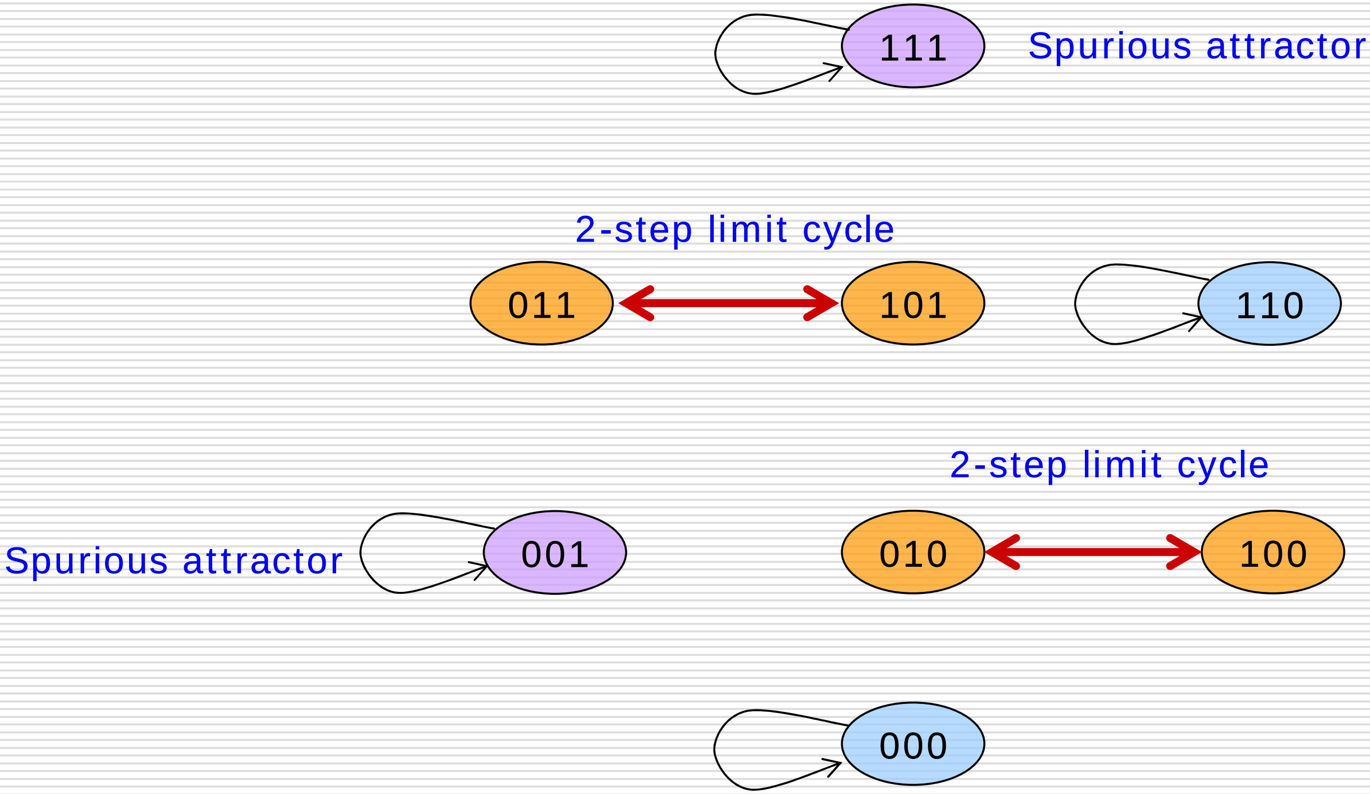
Example 2

- Network architecture



Probe			Initial Energy	Final State							Final Energy		
−1	−1	−1	−4	−1	−1	−1					−4		
−1	−1	1	−4	−1	−1	1					−4		
−1	1	−1	4	1	−1	−1	↔	−1	1	−1	4	↔	4
−1	1	1	4	1	−1	1	↔	−1	1	1	4	↔	4
1	−1	−1	4	−1	1	−1	↔	1	−1	−1	4	↔	4
1	−1	1	4	−1	1	1	↔	1	−1	1	4	↔	4
1	1	−1	−4	1	1	−1					−4		
1	1	1	−4	1	1	1					−4		

Basins of Attraction



Recall of Memories in Continuous Time

- Encode (1,1) (-1,-1)

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

- Write down the additive dynamics equations (no inputs)

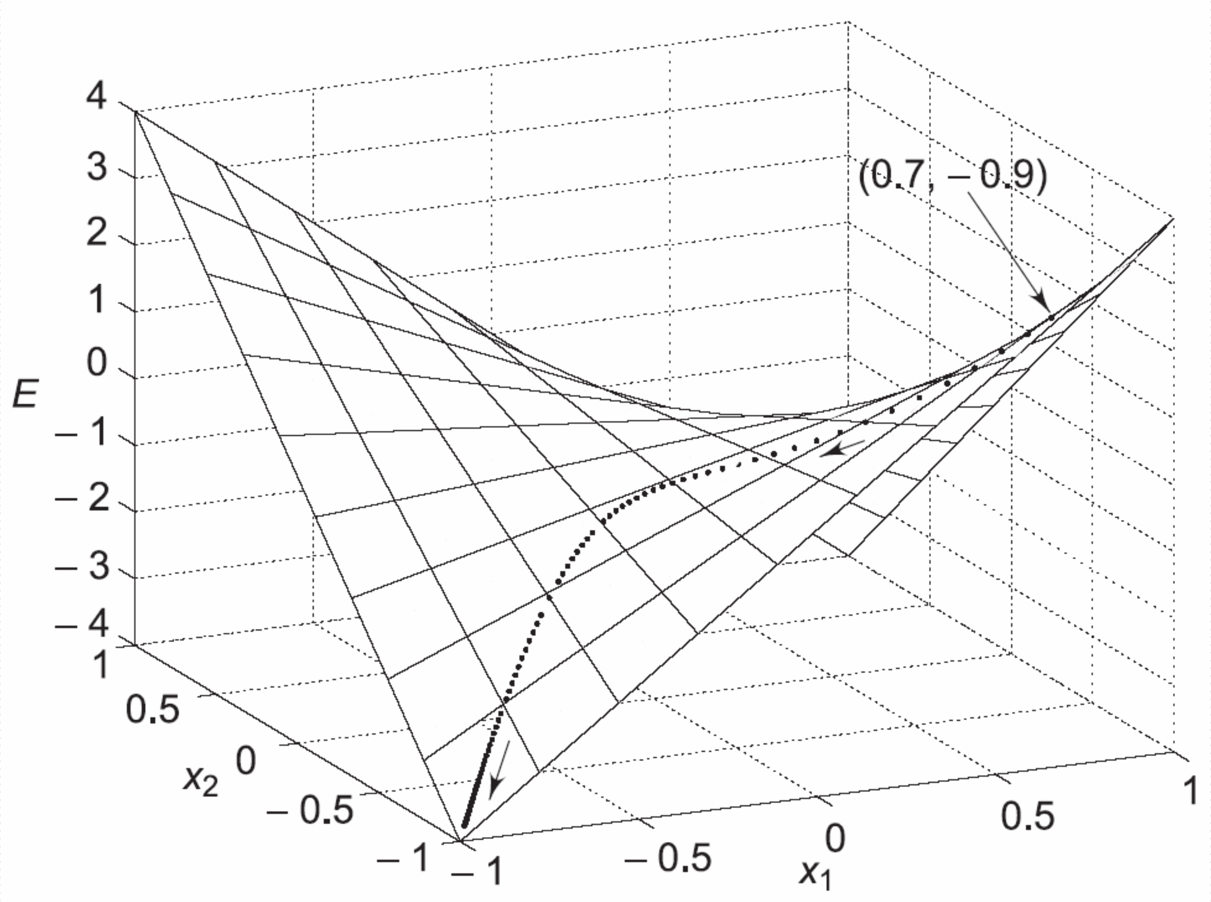
- Integrate 4th order Runge Kutta Technique

$$\dot{x}_1 = f(x_1) = -ax_1 + w_{21}\mathcal{S}(x_2)$$
$$\dot{x}_2 = f(x_2) = -ax_2 + w_{12}\mathcal{S}(x_1)$$

$$x_i^{k+1} = x_i^k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = hf(x_i^k)$$
$$k_2 = hf(x_i^k + \frac{1}{2}k_1)$$
$$k_3 = hf(x_i^k + \frac{1}{2}k_2)$$
$$k_4 = hf(x_i^k + k_3)$$

Recall of Memories in Continuous Time



Spurious Attractors

- Undesired attractors are called spurious attractors
  - *Complement states*
  - *Mixture states*
  - *Spin glass states*
  - *Alien attractors*

Error Correction with Bipolar Encoding

- Bipolar outer product encoding builds an error correction capability into the decoding process
- Assume a weight matrix **W** encoding Q vectors, and a probe equal to one of the encoded vectors
- Perform a signal noise decomposition…

Signal-Noise Decomposition

$$
\begin{aligned}
Y^T = X_i^T \mathbf{W} &= X_i^T \left(\sum_{k=1}^{Q} X_k X_k^T \right) \\
&= X_i^T X_i X_i^T + \sum_{k \neq i} X_i^T X_k X_k^T \\
&= \underbrace{n X_i^T}_{\text{signal}} + \underbrace{\sum_{k \neq i} c_{ik} X_k^T}_{\text{noise}}
\end{aligned}
$$

Correction Coefficient

- Note the correction coefficient
  - $c_{ik} = X_i^T X_k = n - 2\, h_{ik}$

$$
\begin{aligned}
h_{ik} &< \frac{n}{2} \quad \text{then} \quad c_{ik} > 0 \\
h_{ik} &> \frac{n}{2} \quad \text{then} \quad c_{ik} < 0 \\
h_{ik} &= \frac{n}{2} \quad \text{then} \quad c_{ik} = 0
\end{aligned}
$$

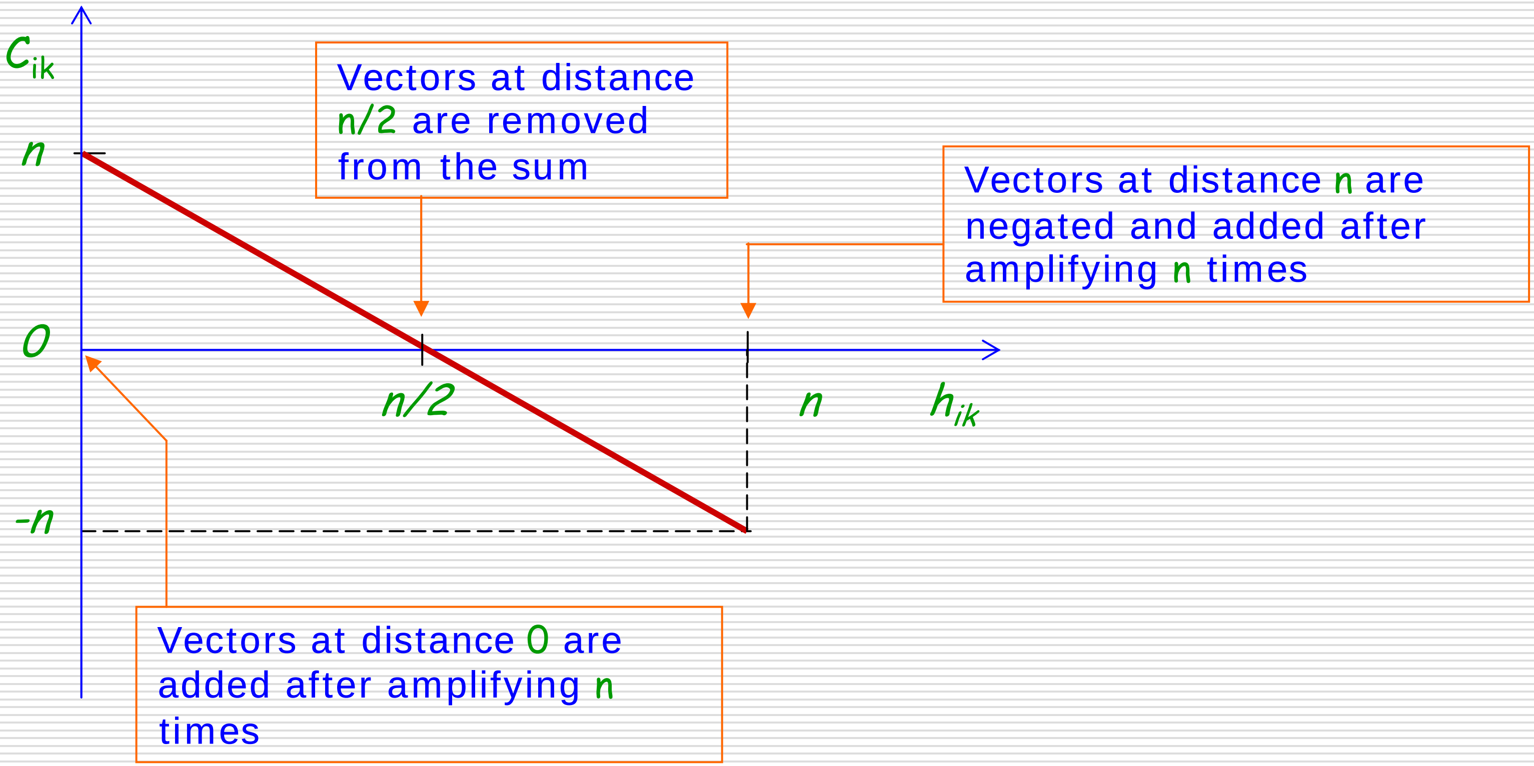
Important Observations

- If X_k has greater than half of its bits different from X_i, X_k^c will have less than half of its bits different from X_i
- In the limiting case if X_k has all n bits different from X_i, $X_k = X_i^c$, $-X_k = -X_i^c = X_i$
- If X_k has half of its bits different from X_i, $-X_k$ also has half its bits different from X_i

Operational Implications

- For vectors in the noise term whose Hamming distance is greater than n/2 from the memory it is more beneficial to first negate and then add them to the signal term since negation brings them closer to the signal vector
- Not if Hamming distance of the noise vector is less than n/2

Correction Coefficient vs Hamming Distance



Error Performance of Hopfield Networks

- In using outer product encoding the ji component of the weight matrix can be expressed as

$$w_{ji} = \sum_{k=1}^{Q} x_j^k x_i^k$$

- The activation of the i^{th} neuron in response to presentation of probe X_p is as follows:

$$\begin{aligned}
y_i &= \sum_{j=1}^{n} w_{ji} x_j^p \\
&= \sum_{j=1}^{n} \left(\sum_{k=1}^{Q} x_j^k x_i^k \right) x_j^p \\
&= \sum_{j=1}^{n} \left(x_j^p x_j^p x_i^p + \sum_{\substack{k=1 \\ k \neq p}}^{Q} x_j^p x_j^k x_i^k \right) \\
&= \underbrace{n x_i^p}_{\text{signal}} + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq p}}^{Q} \underbrace{x_j^p x_j^k x_i^k}_{\text{noise}}
\end{aligned}$$

Signal-Noise Ratio

- Assumption: Encoded vectors were generated by a sequence of Bernoulli trials
  - The noise term is asymptotically Gaussian distributed. Gaussian distribution has a mean of zero and a variance n(Q-1)
  - The signal variance is n^2 and its mean is 0

$$\rho = \frac{\text{Variance of signal}}{\text{Variance of noise}} = \frac{n^2}{(Q-1)n} = \frac{n}{Q-1} \simeq \frac{n}{Q}$$

Probability of Correct Recall

- The probability for correct recall for one bit can be shown to take the form

$$P(y_i > 0 | x_i^p = 1) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\sqrt{\frac{\rho}{2}}\right)\right]$$

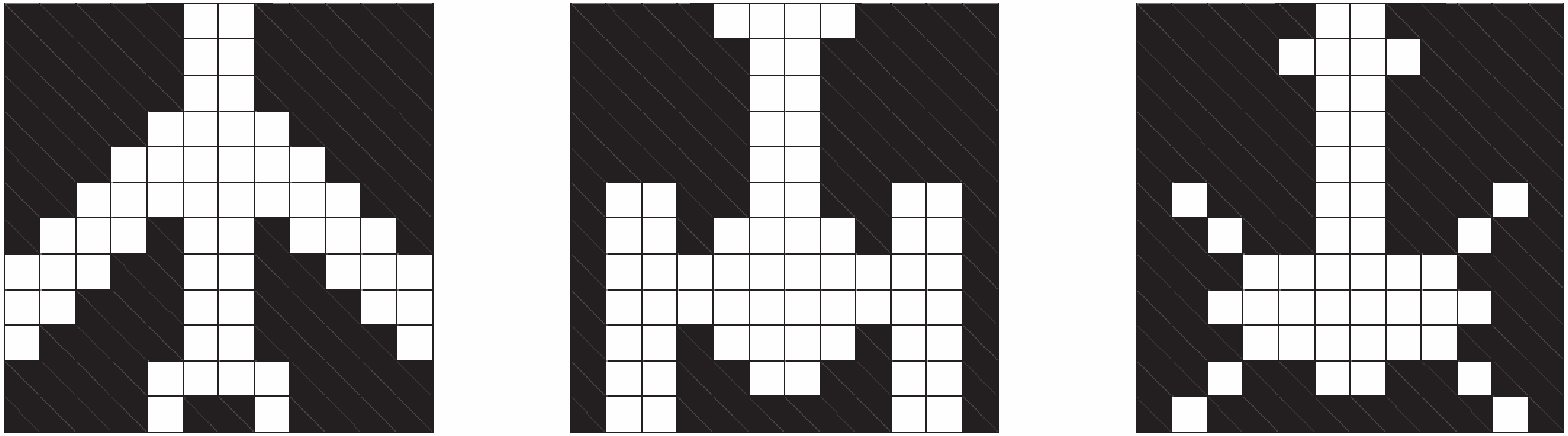
- Stability of most of the fundamental memories, requires the minimum signal-noise ratio satisfies the condition

$$\rho_{\min} = 2 \ln n$$

- See text for algebra.

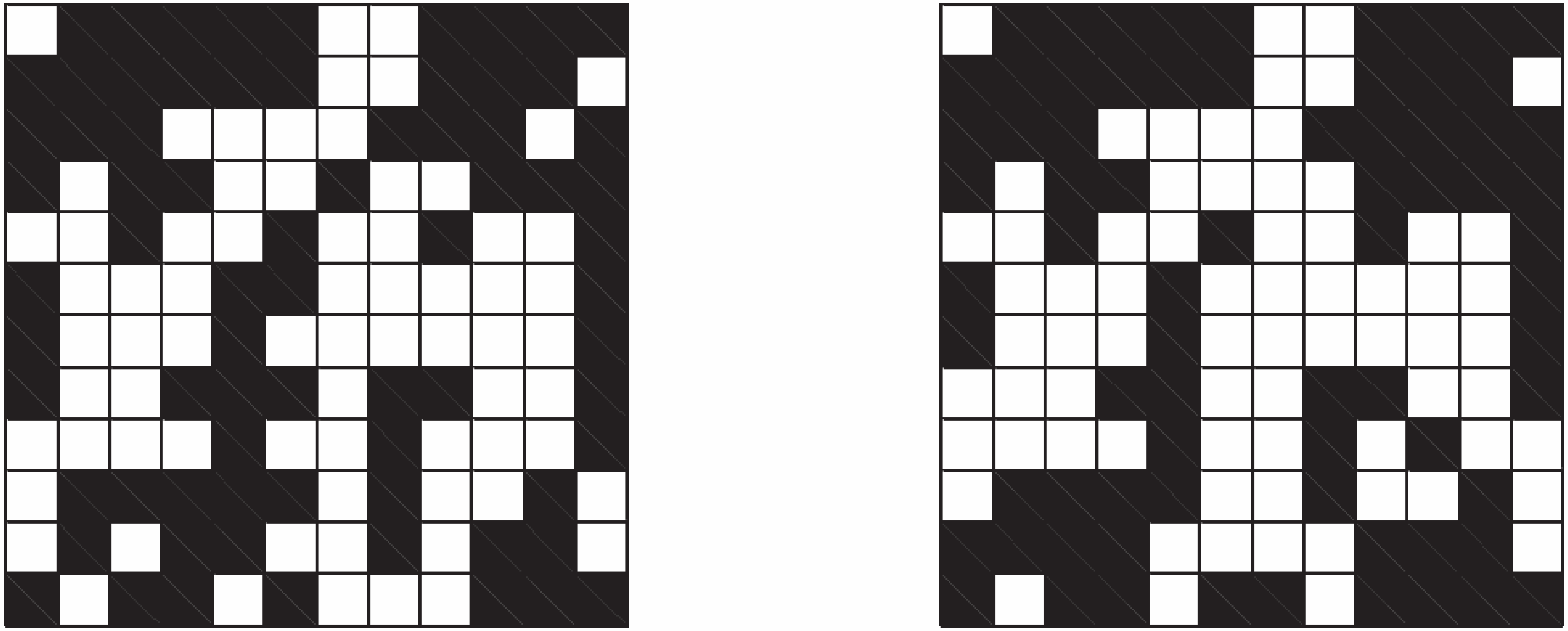
Application: Pattern Completion

- Consider three 12×12 pixel based images



(a) Plane (b) Tank (c) Helicopter

Application: Pattern Completion

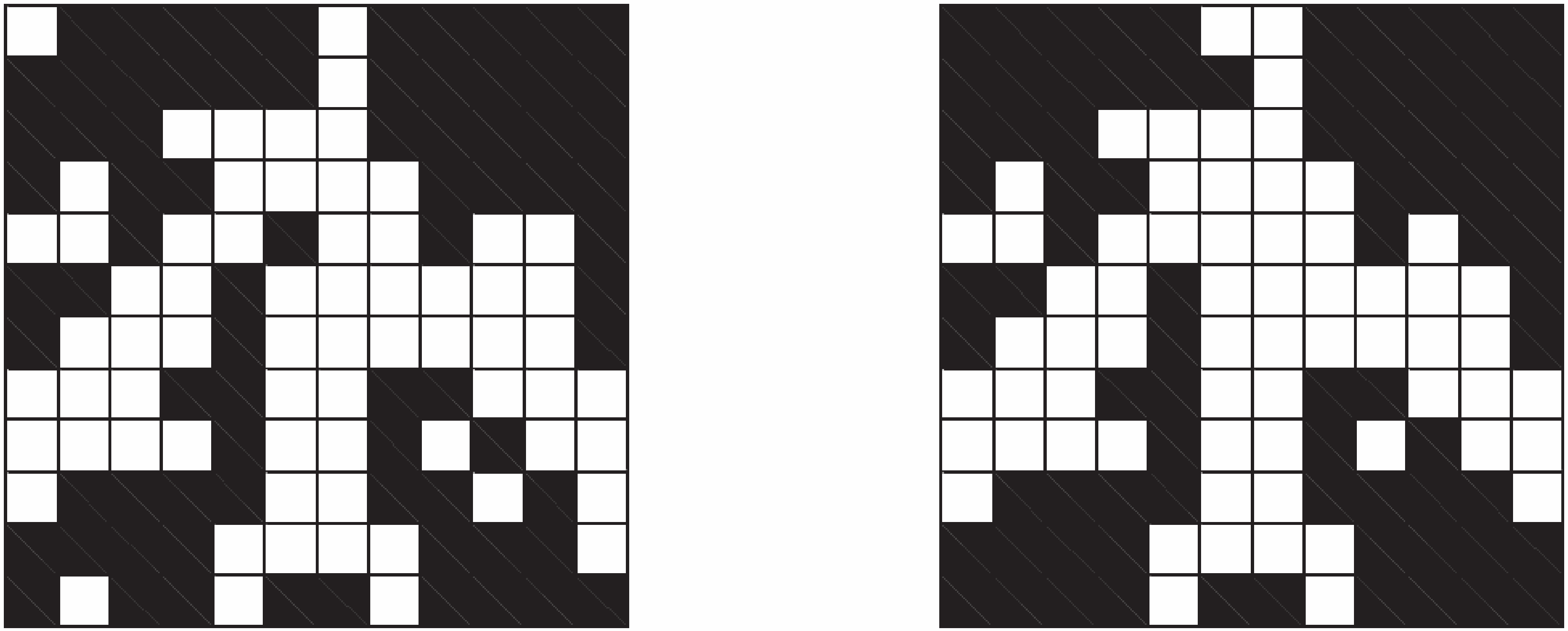


(a) 40 % Distorted Plane

(b) Iteration: 48

Contd.

Application: Pattern Completion

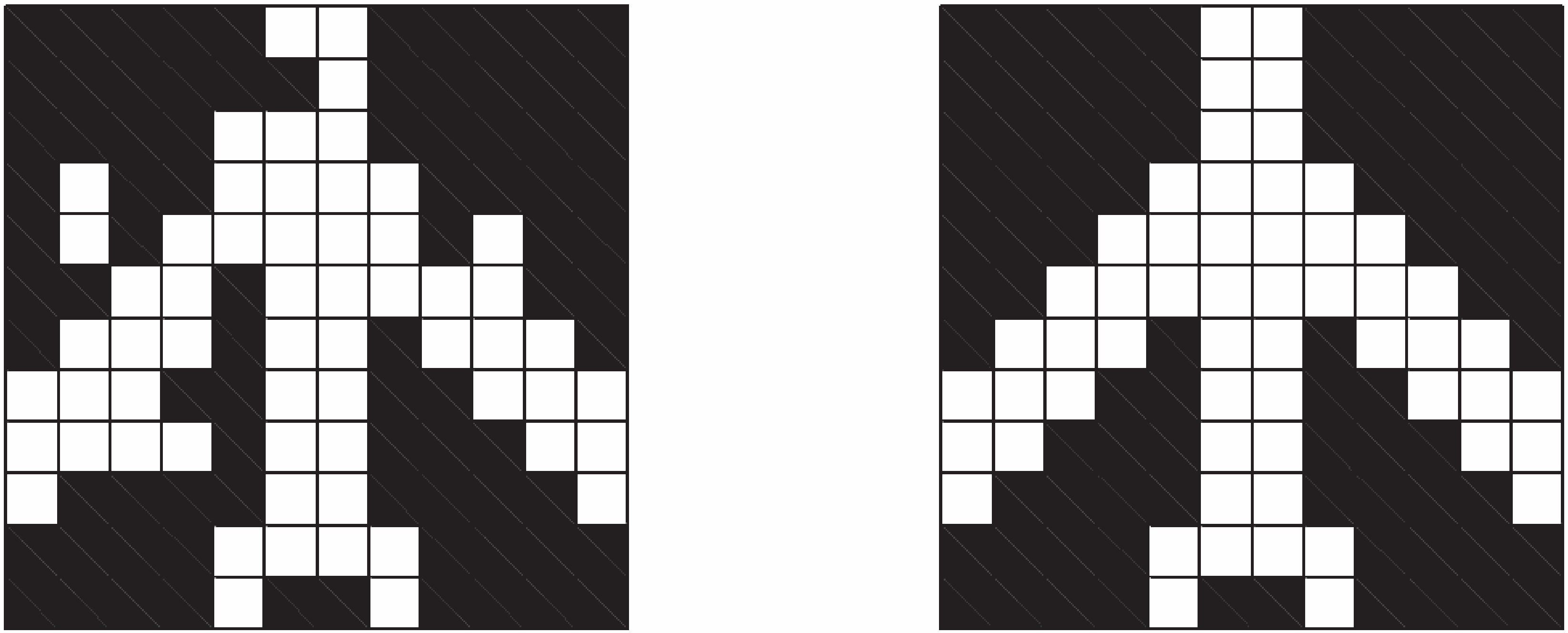


(c) Iteration: 72

(d) Iteration: 96

Contd.

Application: Pattern Completion



(e) Iteration: 120 (f) Iteration: 144

Brain-State-in-a-Box (BSB) Neural Network

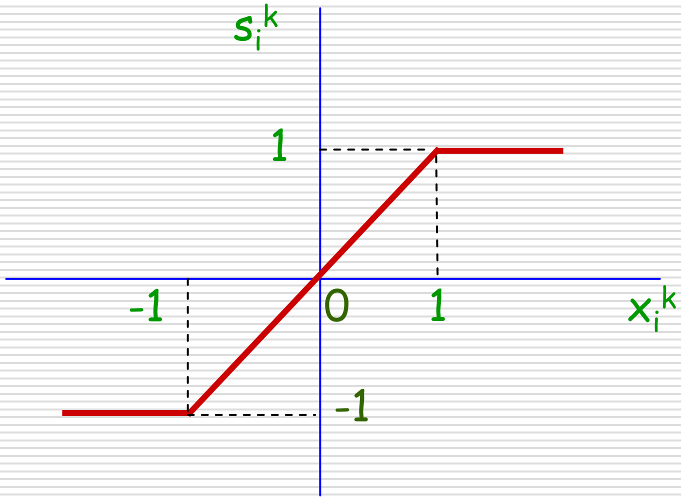
- Predecessor of the Hopfield network
- Extends the linear associator
- Similar to the Hopfield network
  - autoassociative model
  - connection matrix computed using outer products
- Operation of both models is also very similar
- Differences arising primarily
  - In the way activations are computed in each iteration
  - Signal function
- BSB stands apart from other models in its use of the linear-threshold signal function.

Operational Details

- Activation Function $$X_{k+1}^T = \gamma S_k^T + \alpha S_k^T \mathbf{W} + \delta S_0^T$$
- Signal Function $$S_{k+1} = \mathcal{S}(X_{k+1})$$
- Weight Matrix $$\mathbf{W} = \sum_{k=1}^{Q} \lambda_k A_k A_k^T$$
  - Arranged so that A_i is an eigenvector of W with eigenvalue λ_i

Operational Details: Signal Function

- Piecewise linear signal function.
- States are boxed into the hypercube: ***brain-state-in-a-box!***



$$s_i^k = \mathcal{S}(x_i^k) = \begin{cases} -1 & x_i^k < -1 \\ x_i^k & -1 \le x_i^k \le 1 \\ 1 & x_i^k > 1 \end{cases}$$

Operational Summary of the BSB Model

Given	A set of binary vectors $\{A_i\}_{i=1}^{Q}$ to be encoded into a BSB network using binary encoding.
Encode	↬ $\mathbf{W} = \sum_{k=1}^{Q} \lambda_k A_k A_k^T$
Initialize	↬ Set up neuron signals vector to the probe vector S_0. ↬ Set γ, α, δ.
Iterate	↻Repeat { ⇝ Compute the neuron activations $X_{k+1}^T = \gamma S_k^T + \alpha S_k^T \mathbf{W} + \delta S_0^T$ ⇝ Update the neurons synchronously $S_{k+1} = \mathcal{S}(X_{k+1})$, where $\mathcal{S}(x_i^k) = \begin{cases} -1 & x_i^k < -1 \\ x_i^k & -1 \le x_i^k \le 1 \\ 1 & x_i^k > 1 \end{cases}$ } until $(S_{k+1} = S_k)$

Rework Signal Update Equation

- Neuron-specific scalar form of the BSB vector update equation with $\gamma = 1$ and $\delta = 0$

$$s_i^{k+1} = \mathcal{S}\left(s_i^k + \alpha \sum_{j=1}^{n} w_{ji} s_j^k\right)$$

$$= \mathcal{S}\left(\sum_{j=1}^{n} (\delta_{ji} + \alpha w_{ji}) s_j^k\right)$$

$$= \mathcal{S}\left(\sum_{j=1}^{n} B_{ji} s_j^k\right)$$

- BSB is a special case of Cohen-Grossberg dynamics (see text)

$$B_{ji} = \underbrace{\delta_{ji}}_{\text{Kronecker delta}} + \alpha w_{ji}$$

Signal-Sum Exchange (Grossberg)

- Rewrite the discrete equation in continuous form → $$\dot{s}_i = -s_i + \mathcal{S}\left(\sum_{j=1}^{n} B_{ji} s_j\right)$$
- Transform variables → $$y_i = \sum_{j=1}^{n} B_{ji} s_j$$
- Re-write the equation in transformed space: Additive Dynamics → $$\dot{y}_i = -y_i + \sum_{j=1}^{n} B_{ji} \mathcal{S}(y_j)$$
- Lyapunov Function

$$E = \sum_{i=1}^{n} \int_0^{y_i} \alpha_i \mathcal{S}'(\alpha_i) d\alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} B_{ij} \mathcal{S}(y_i) \mathcal{S}(y_j)$$

BSB Network Trajectories

Encode

$A_1 = (1,-1)$, $A_2 = (1,1)$

Eigenvectors

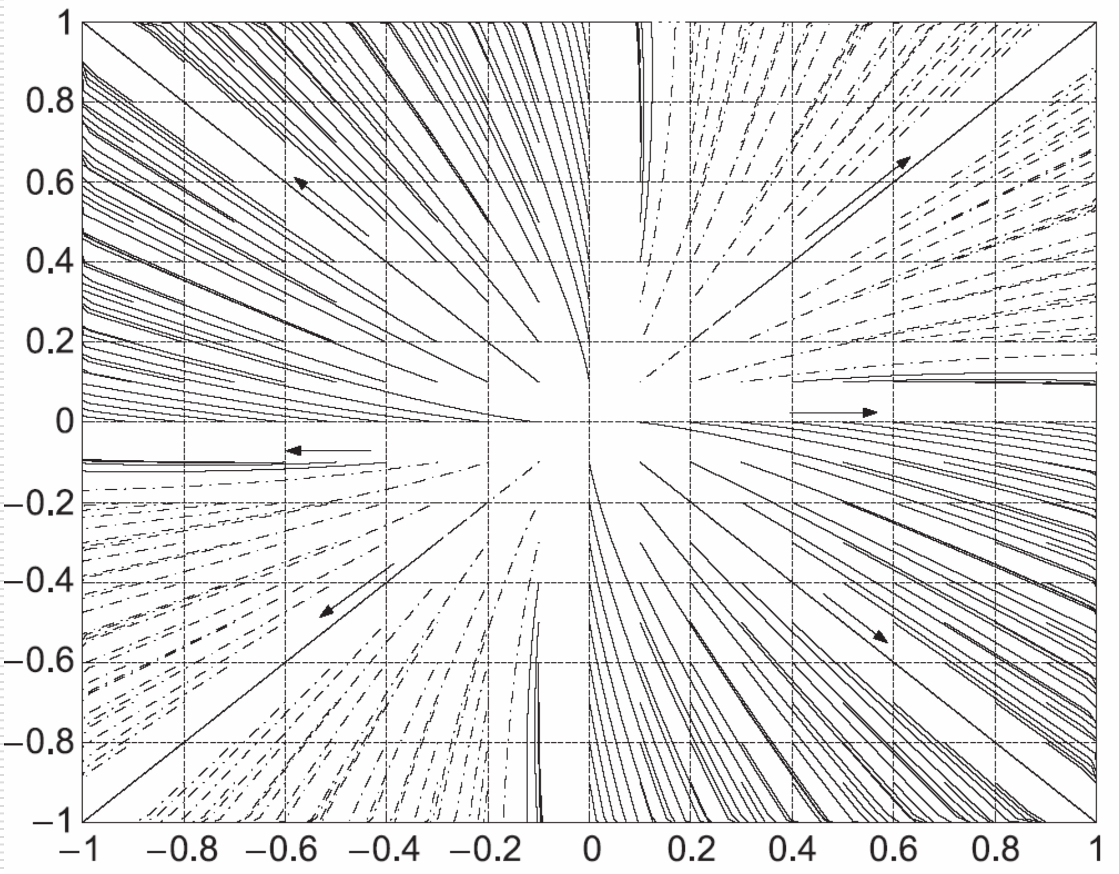
$\lambda_1 = 0.04 \quad \lambda_2 = 0.03$

Choose

$\gamma = \alpha = 1$, $\delta = 0$

$$\mathbf{W} = \frac{1}{\sqrt{2}}\left(\lambda_1 A_1 A_1^T + \lambda_2 A_2 A_2^T\right)$$

$$= \begin{pmatrix} 0.035 & -0.005 \\ -0.005 & 0.035 \end{pmatrix}$$



MATLAB Code for BSB

```
delta = 0;
gamma = 1;
alpha = 1;
lim_u = 1;
lim_l = -1;
lambda1 = 0.04;
lambda2 = 0.03;
%Specify vectors to encode
x1 = [1 -1]';
x2 = [1 1]';
%Normalize them
x1_n = x1/sqrt(2);
x2_n = x2/sqrt(2);
%Encode them
W = lambda1*x1_n * x1_n' + lambda2*x2_n *
        x2_n';
figure(1);
hold on;
grid on;
%Start at an initial point of (-1,-1) and go to (1,1)
%in steps pf 0.1
x = -1;
y = -1;
inc = 0.1;
while(x <=1)
while(y <= 1);
s(:,1) = [x y]';
for i = 2:150
act = gamma*s(:,i-1)'+alpha*s(:,i-1)'*W + delta*s(:,1)';
for j=1:2
if ( act(j) <= lim_l) s(j,i) = lim_l;
elseif (act(j) >= lim_u) s(j,i) = lim_u;
else s(j,i) = act(j);
end
end
end
...
...%Line plotting code goes here
...
y = y + inc;
end
y = -1;
x = x + inc;
end
```

BSB Applications

- Speech perception and probability learning
- Multistable perception
- Cognitive computation
- Radar signal categorization

Design of High Dimensional and Complex Pattern Classifiers

- Analytical methods or techniques that use local derivatives for gradient descent turn out to be inadequate
- Error surfaces of non-linear systems in high dimensional spaces have multiple minima, and finding the true global minimum is indeed a difficult task
- Exhaustive search in solution space to find a good set of parameters is almost impossible
- Increasing complexity of the problem is accompanied by less training data and less prior knowledge

Simulated Annealing

- Provides a general framework for the optimization of the behaviour of complex systems
- Operates by introducing noise in a controllable fashion into the operational dynamics of the system
- Robust iterative search

Configuration, Temperature, Ground State

- A specific combination of neuron states is a *configuration* of the network
  - In an n node network there are 2^n configurations
- Basic Idea:
  - Generate different configurations of the system at various values of a control parameter called the *temperature*
  - Gradually reduce the value of this parameter to search for an optimal or *ground state* solution to the problem

An Important Result from Statistical Mechanics

- Low energy configurations are very few:
  - Corresponding to the vectors that are encoded into the network and other spurious memories
- Many more possible configurations that correspond to higher energies
- From statistical mechanics:
  - A system is in thermal equilibrium when a configuration γ with energy E_γ occurs with probability

$$\boxed{P(\gamma)} = \frac{e^{-E_\gamma/T}}{\sum_{\gamma'} e^{-E_{\gamma'}/T}} = \frac{e^{-E_\gamma/T}}{\boxed{\mathcal{Z}(T)}}$$

Boltzmann Probability Distribution

Partition Function

Simulated Annealing Procedure

- Randomize neuron states once in the beginning, and initialize the temperature to a high value.
- Choose a neuron I randomly from the network
- Compute the energy E_A of the present configuration A
- Flip the state of neuron I to generate a new configuration B
- Compute energy E_B of configuration B

Contd.

Simulated Annealing Procedure

- If $E_B < E_A$ accept configuration B
- Else accept configuration B with probability $\exp(-\Delta E/T)$, $\Delta E = E_B - E_A$
- Continue selecting and testing neurons randomly, and set their states several times in this way until a *thermal equilibrium* is reached.
- Finally, lower the temperature and repeat the procedure.

Notes

- At high temperatures
  - all configurations are somewhat equally likely.
  - transitions to energetically unfavourable states are frequent
- At lower temperatures
  - transitions to energetically unfavourable states become less frequent
  - search becomes more like the usual descent procedures
- If the cooling is sufficiently slow, the network has a very high probability of finding itself in an optimal configuration that represents a minimum energy configuration

Notes

- Critical aspect:
  - Choice of initial temperature
  - Annealing schedule $T_{k+1} = cT_k$
    - Typical working range : 0.8-0.9

Stochastic Simulated Annealing Algorithm: Hopfield Network

Given	A set of binary vectors $\{A_i\}_{i=1}^{Q}$ to be encoded into a Hopfield CAM using bipolar encoding.
Encode	$\hookrightarrow \mathbf{W} = \sum_{k=1}^{Q} X_k X_k^T - Q\mathbf{I}$
Initialize	$\hookrightarrow T_0,\ k = 0,\ k_{\max}$ (temperature, iteration, iteration limit) $S_0,\ c$ (signal vector, temperature contraction)

Contd.

Stochastic Simulated Annealing Algorithm

Iterate ⟳Repeat

{

⟳Repeat

{

$\rightsquigarrow$ Select neuron I randomly.

$\rightsquigarrow$ Compute: $\Delta E^{(I)} = 2s_I \sum_{j=1}^{n} w_{jI} s_j$

$\rightsquigarrow$ if $\Delta E^{(I)} < 0$, $s_I = -s_I$

else if $e^{-\Delta E^{(I)}/T_k} >$ rand $[0,1)$, $s_I = -s_I$

} until(all nodes are polled several times)

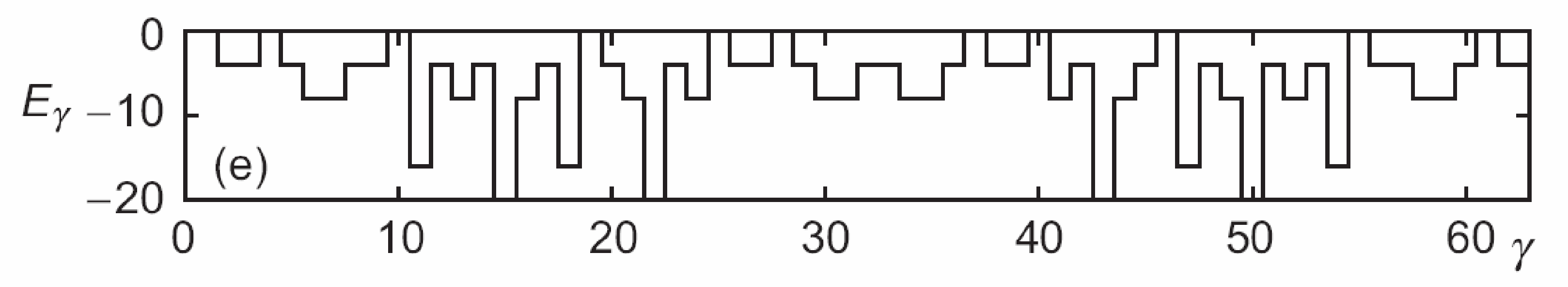
$\rightsquigarrow$ Reduce temperature: $T_{k+1} = cT_k$

} until ($k = k_{\max}$ or stopping criterion met)

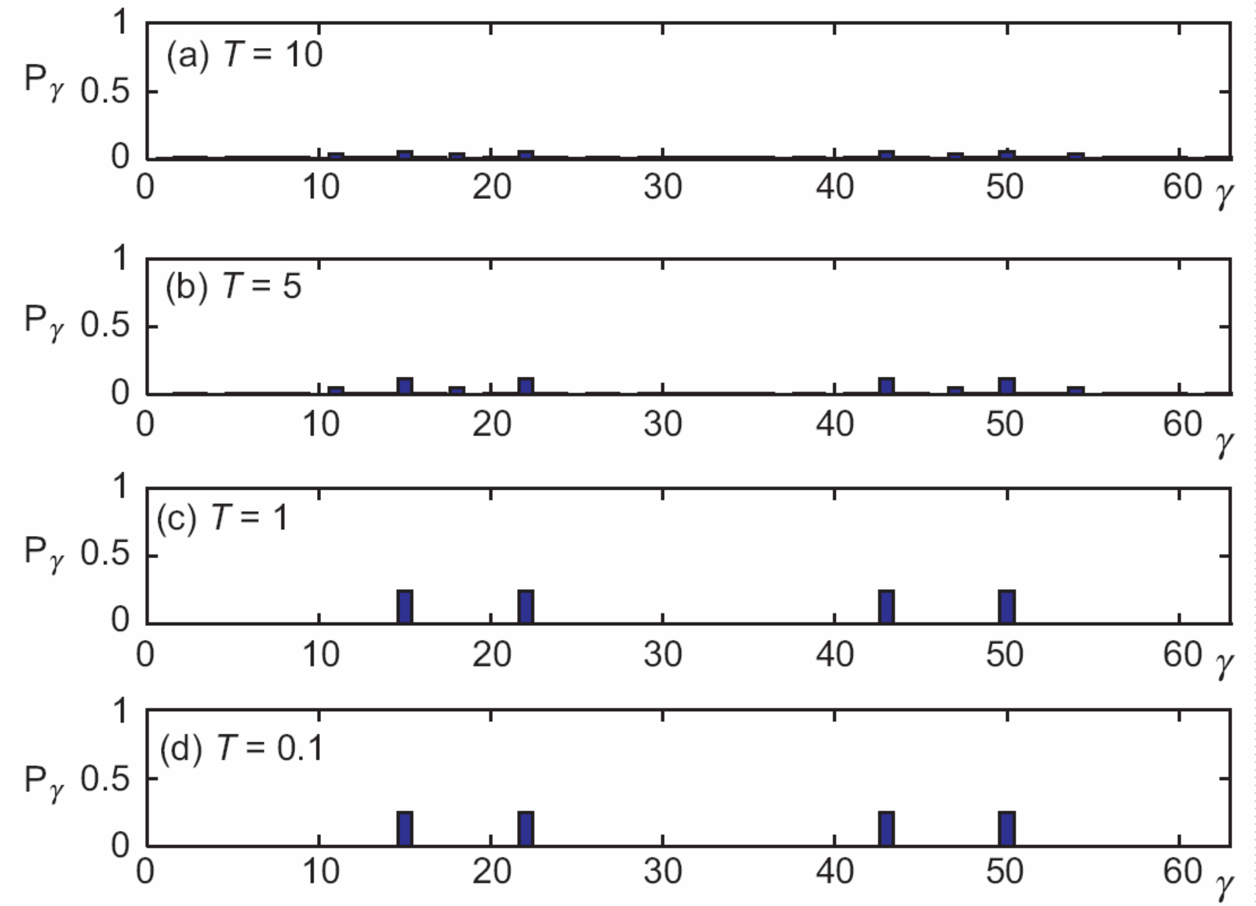
Example

- Consider a Hopfield network encoding vectors $A_1 = (110001)$ $A_2 = (101010)$

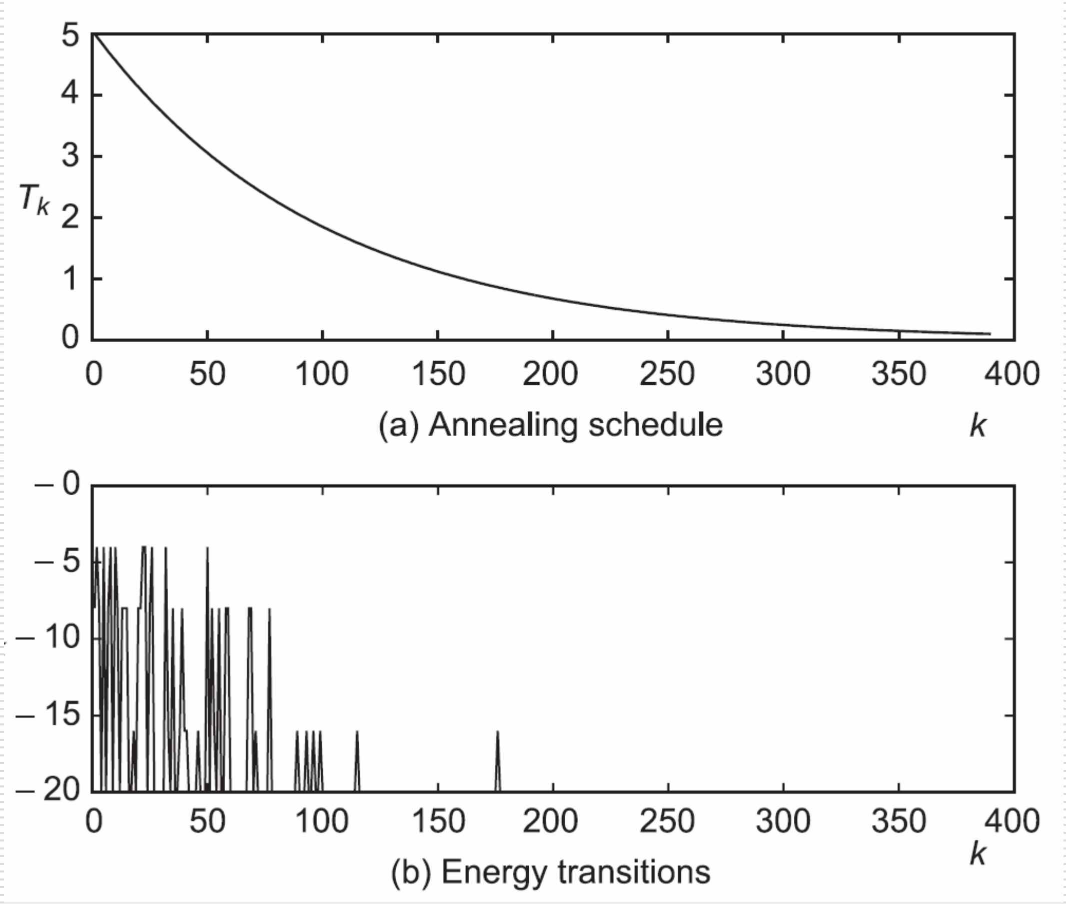
$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -2 & 0 & -2 & 2 \\ 0 & -2 & 0 & 0 & 2 & -2 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 2 & 0 & 0 & -2 \\ 0 & 2 & -2 & 0 & -2 & 0 \end{pmatrix}$$



Hopfield Net SA: Simulation Result



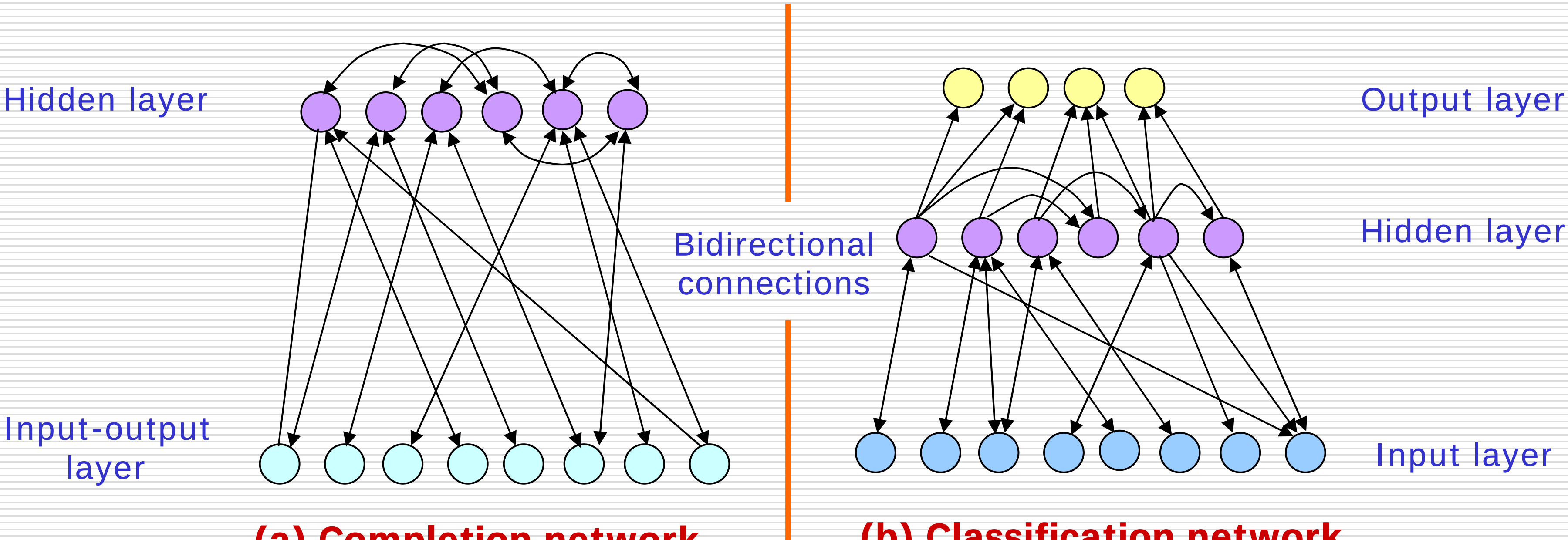
Hopfield Net SA: Simulation Result



Boltzmann Machine

- Extension of the discrete Hopfield network
- Replaces the deterministic local search dynamics by randomized local search dynamics.
- Introduces a powerful stochastic learning algorithm in place of the simple Hebbian rule
- Relaxation is done using the simulated annealing procedure

Boltzmann Machine: Architecture



(a) Completion network

(b) Classification network

Hopfield Network vs Boltzmann Machines

- Similarities
  - Neuron states are bipolar
  - Weights are symmetric
  - Neurons are selected at random for asynchronous update
  - There is no self-feedback
- Differences
  - Architecture
    - Boltzmann machine uses a hidden layer
  - Neuron update
    - Hopfield network is deterministic
    - Boltzmann networks is stochastic
- Differences (Contd.)
  - Learning
    - Hopfield network vectors are encoded into the system using outer product correlation encoding and there is no separate learning phase
    - Boltzmann machines learning is based on simulated annealing and gradient descent

Operational Details

- Consider a Boltzmann completion network with configurations described by an energy function

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} s_i s_j$$

- Glauber Dynamics: for the I^{th} neuron,

$$P(s_I = \pm 1) = \frac{1}{1 + e^{\mp 2x_I/T}}$$

Boltzmann Machine Relaxation Procedure: Operational Summary

Given	A Boltzmann machine with weights specified in advance.
Initialize	↬ $T_0, k = 0$ (temperature, iteration index) ↬ $T_{\min}, c$ (temperature limit and contraction) ↬ Clamp the known signal values ↬ Randomize remaining signals to values in $\{-1, 1\}$

Contd.

Boltzmann Machine Relaxation Procedure: Operational Summary

Iterate ⟳Repeat

{

⟳Repeat

{

⇝ Select neuron I randomly.

⇝ If not a clamped neuron, given its signal, s_I

⇝ Compute: $x_I = \sum_j w_{jI} s_j$

⇝ Compute: $P = \frac{1}{1+e^{-x_I/T}}$

⇝ if rand[0, 1) $< P$, set $s_I = 1$

⇝ else set $s_I = -1$

} until(all nodes are updated many times)

⇝ Reduce temperature: $T_{k+1} = cT_k$

} until $(T_{k+1} < T_{\min})$

Learning Objective in the Boltzmann Machine

- *Find a set of weights such that at any given temperature, the actual distribution* $P(\alpha)$ *matches the desired distribution* $Q(\alpha)$ *as closely as possible.*
- Use standard gradient descent with a relative entropy error function: Kullback-Leibler Distance

$$\mathcal{D}(Q(\alpha),\, P(\alpha)) = \sum_{\alpha} Q(\alpha) \log \frac{Q(\alpha)}{P(\alpha)}$$

Final Weight Update Expression

- Define

$$E[s_i s_j]_{\text{clamped}} = \sum_{\alpha} \sum_{\beta} Q(\alpha) P(\beta|\alpha) s_i(\alpha\beta) s_j(\alpha\beta)$$

- Weight change dictated by

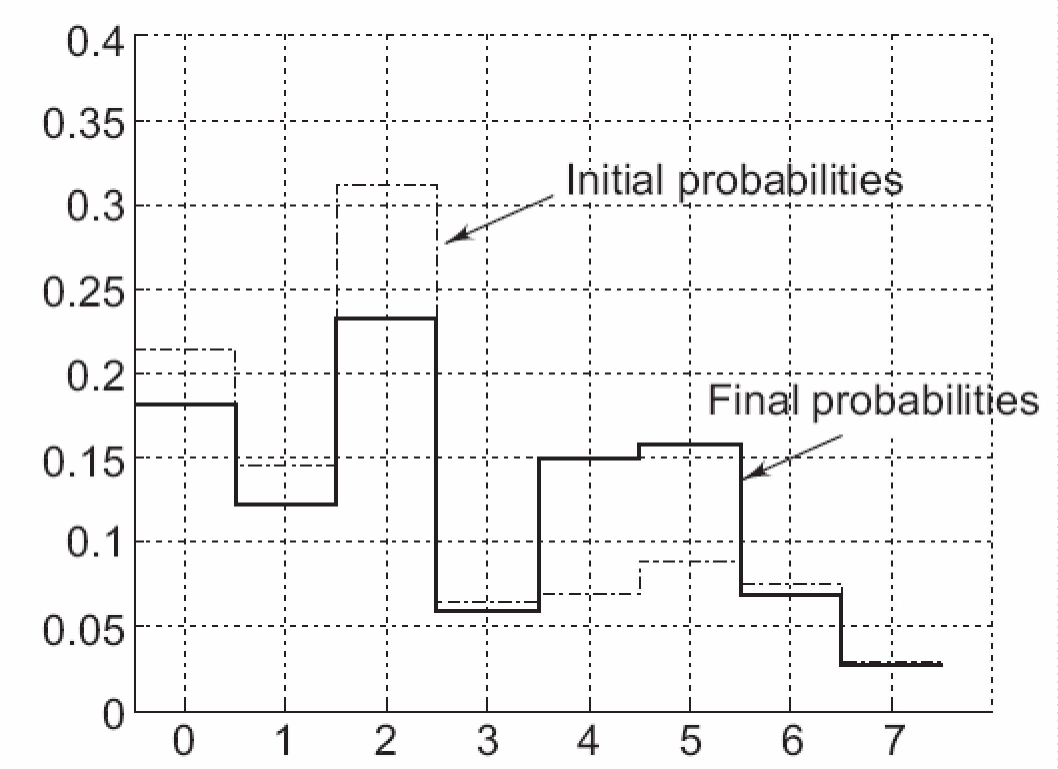
$$\Delta w_{ij} = \frac{\eta}{T} \left(\underbrace{E[s_i s_j]_{\text{clamped}}}_{\text{Learning component}} - \underbrace{E[s_i s_j]_{\text{free}}}_{\text{Unlearning component}} \right)$$

See text for algebra.

MATLAB Code to Implement Boltzmann Learning Algorithm

```
for k = 1:10
rand('state',sum(100*clock));
free_signals = [(2*round(rand(rep_num, 3)) - 1)...
-ones(rep_num,1) ];
clamped_h_signals = [(2*round(rand(rep_num,1)) -
        1)...
-ones(rep_num,1) ];
T = 10;
while (T > 1)
for relax = 1:num_cycles
rand('state',sum(100*clock));
Xf = free_signals * W; % compute activations
pf = 1./(1 + exp(-Xf./T));% compute the flip
        probabilities
randf = rand(rep_num, 3); % generate random #s in
        (0,1)
for i = 1:rep_num % work down network instances
j = round(1+2*rand);
if (randf(i,j) < pf(i,j))
free_signals(i,j) = 1;
else free_signals(i,j) = -1;
end
end
rand('state',sum(100*clock));
clamped_signals = [repmat(Ab, [rep_num/Q 1])
        clamped_h_signals];
Xc = clamped_signals * W;
pc = 1./(1 + exp(-Xc./T)); % compute the flip probabilities
randc = rand(rep_num, 3); % generate random #s in (0,1)
for i=1:rep_num
j = round(1+2*rand);
if (randc(i,j) < pc(i,j))
clamped_signals(i,j) = 1;
else clamped_signals(i,j) = -1;
end
end
clamped_h_signals = clamped_signals(:,3:4);
end
T = T*c
end
clamped_signals = [repmat(Ab, [rep_num 1])
        clamped_h_signals];
%Collect stats and change weights
Expc = (1/rep_num)*clamped_signals' * clamped_signals
Expf = (1/rep_num)*free_signals'* free_signals
deltaW = Expc(:,1:3) - Expf(:,1:3);
W = W + (eta/T) * deltaW.* [(1-eye(3)); ones(1,3)]
end
```

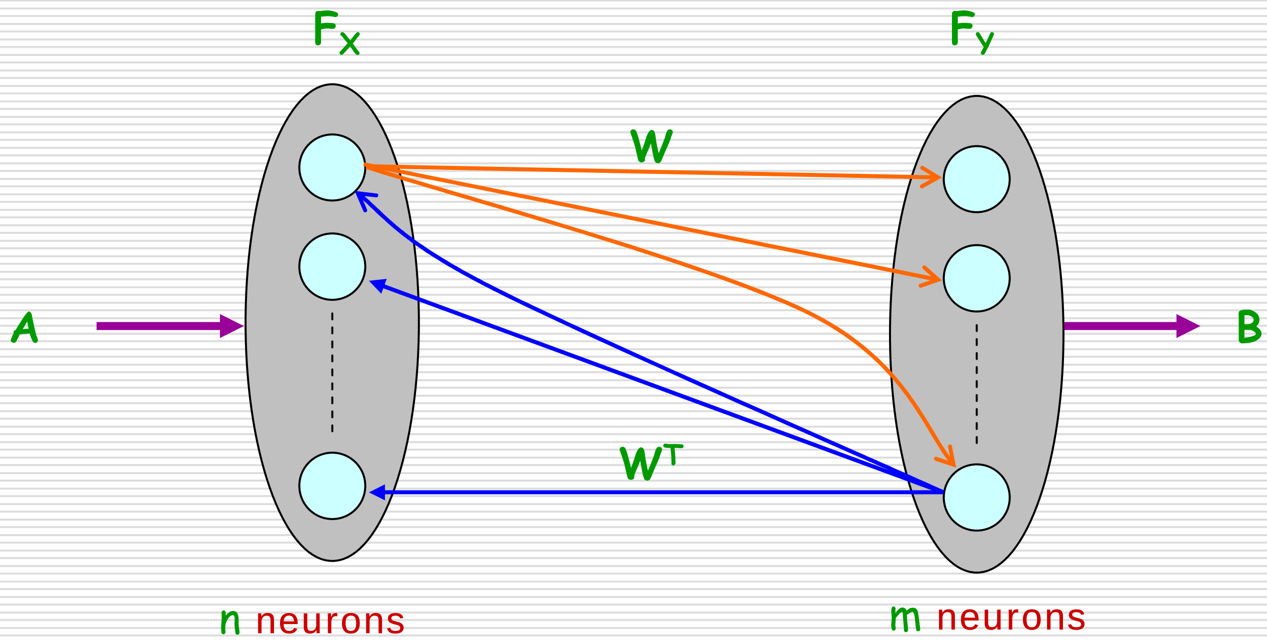
Simulation Result



Bidirectional Associative Memory

- Two field attractor neural network introduced by Kosko
- Continuous BAMs employ additive dynamics
- Heteroassociative in nature

BAM Architecture



Relaxation Procedure

- Through a sequence of forward and reverse transmissions
- A bidirectional relaxation process
- Eventually leading to a *bidirectional equilibrium*

$$(A')^T \mathbf{W} \rightarrow B'$$

$$A'' \leftarrow (B')^T \mathbf{W}^T$$

$$(A'')^T \mathbf{W} \rightarrow B''$$

$$\vdots$$

$$\left.\begin{aligned} A_f^T \mathbf{W} &\rightarrow B_f \\ A_f &\leftarrow B_f^T \mathbf{W}^T \end{aligned}\right.$$

Signal Update: Forward Pass

- Compute activations of F_y neurons

$$y_j^k = \sum_{i=1}^{n} w_{ij} a_i^k$$

- Generate bipolar signals on F_y neurons

$$b_j^k = \mathcal{S}(y_j^k) = \begin{cases} 1 & \text{if} \quad y_j^k > 0 \\ \left.\begin{matrix} 0 \text{ (binary)} \\ -1 \text{ (bipolar)} \end{matrix}\right\} & \text{if} \quad y_j^k < 0 \\ b_j^{k-1} & \text{if} \quad y_j^k = 0 \end{cases}$$

Signal Update: Reverse Pass

- Compute activations of F_X neurons

$$x_i^{k+1} = \sum_{j=1}^{m} w_{ji} b_j^k$$

- Generate bipolar signals on F_X neurons

$$a_i^{k+1} = \mathcal{S}(x_i^{k+1}) = \begin{cases} 1 & \text{if} \quad x_i^{k+1} > 0 \\ \left.\begin{matrix} 0\,(\text{binary}) \\ -1\,(\text{bipolar}) \end{matrix}\right\} & \text{if} \quad x_i^{k+1} < 0 \\ a_i^k & \text{if} \quad x_i^{k+1} = 0 \end{cases}$$

Encoding Associations into a BAM System

- Use bipolar outer product encoding
- Programming each hetero-association
- Hebbian Law embedded
- Given $T = \{X_k, Y_k\}$, $X_k \in \Re^n$ $Y_k \in \Re^m$

$$\mathbf{W} = \sum_{k=1}^{Q} X_k \, Y_k^T$$

These are bipolar

Operational Summary of BAM Algorithm

Given	A set of binary vector pairs $\{A_i, B_i\}_{i=1}^{Q}$ to be encoded into a BAM using bipolar encoding and decoding.
Encode	↬ $\mathbf{W} = \sum_{k=1}^{Q} X_k Y_k^T$
Initialize	↬ Set up F_X signal vector to a probe vector $\mathcal{S}(X_0)$
	↬ Randomize signals vector on field F_Y
	↬ Set time index $k = 0$

Contd.

Operational Summary of BAM Algorithm

Iterate ⟳Repeat

{

⇝ Compute activations of F_Y : $Y_k = \mathcal{S}(X_k)^T \mathbf{W}$

⇝ Update the neurons in F_Y in accordance with the signal function:

$$\mathcal{S}(y_j^k) = \begin{cases} 1 & \text{if} \quad y_j^k > 0 \\ -1 & \text{if} \quad y_j^k < 0 \\ \mathcal{S}(y_j^{k-1}) & \text{if} \quad y_j^k = 0 \end{cases}$$

⇝ Increment time: $k = k + 1$

⇝ Compute activations of F_X : $X_{k+1} = \mathcal{S}(Y_k)^T \mathbf{W}^T$

⇝ Update the neurons in F_X in accordance with the signal function:

$$\mathcal{S}(x_i^{k+1}) = \begin{cases} 1 & \text{if} \quad x_i^{k+1} > 0 \\ -1 & \text{if} \quad x_i^{k+1} < 0 \\ \mathcal{S}(x_i^k) & \text{if} \quad x_i^{k+1} = 0 \end{cases}$$

} until ($\mathcal{S}(X_{k+1}) = \mathcal{S}(X_k)$ and $\mathcal{S}(Y_k) = \mathcal{S}(Y_{k-1})$)

MATLAB Code to Implement BAM Algorithm

```
n = 5; % Dimension of Fx
p = 4; % Dimension of Fy
q = 2; % Number of associations
mem_vectorsx = [0 1 0 1 0; 1 1 0 0 0]; %Specify Fx
        vectors
mem_vectorsy = [1 0 0 1; 0 1 0 1]; %Specify Fy
        vectors
bip_mem_vecsx = 2*mem_vectorsx-1; %Convert to
        bipolar
bip_mem_vecsy = 2*mem_vectorsy-1;
wt_matrix = zeros(n,p); %Initialize weight matrix
for i=1:q %and recursively compute
wt_matrix = wt_matrix +
        bip_mem_vecsx(i,:)'*bip_mem_vecsy(i,:);
end
k = 1; % Set up time index
probe = [0 1 0 1 1]; % Set up probe
signal_x = 2*probe - 1; % Set Fx signals to probe
signal_y = randomize(1,p); % Randomize Fy signals
pattern_x(k,:) = signal_x; % store the pattern on Fx
pattern_y(k,:) = signal_y; % store the pattern on Fy
flag = 0; % Indicates bidirectional eqlm.
while flag ~=1
act_y = signal_x * wt_matrix; % Computer Fx
        activations
for i = 1 : p % Set up signals
if act_y(i) > 0 signal_y(i) = 1;
elseif act_y(i) < 0 signal_y(i) = -1;
end
end
if (k > 1) % Compare for stability if iteration > 1
compare_y = isequal(signal_y, pattern_y(k-1,:));
else compare_y = 0;
end
pattern_y(k,:) = signal_y; % Store the signal on Fy
act_x = signal_y * wt_matrix';% Compute activations of Fx
for i = 1 : n % Set up signals
if act_x(i) > 0 signal_x(i) = 1;
elseif act_x(i) < 0 signal_x(i) = -1;
end
end
k = k + 1; % Increment time
compare_x = isequal(signal_x, pattern_x(k-1,:)); % Compare
pattern_x(k,:) = signal_x; % and store the signal on Fx
flag = compare_x*compare_y; % Check for bidirectional
        eqlm.
end
pattern_x % Display update traces
pattern_y
```

Handworked Example

- Encode the associations
  - $A_1 = (0\ 1\ 0\ 1\ 0)^T$ $B_1 = (1\ 0\ 0\ 1)^T$
  - $A_2 = (1\ 1\ 0\ 0\ 0)^T$ $B_2 = (0\ 1\ 0\ 1)^T$

$$\mathbf{W}_1 = \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \begin{pmatrix} -1 & 1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

Handworked Example

- Final weight matrix

$$\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 = \begin{pmatrix} -2 & 2 & 0 & 0 \\ 0 & 0 & -2 & 2 \\ 0 & 0 & 2 & -2 \\ 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 \end{pmatrix}$$

- Weight matrix transpose

$$\mathbf{W}^T = \begin{pmatrix} -2 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & -2 & 0 \\ 0 & -2 & 2 & 0 & 2 \\ 0 & 2 & -2 & 0 & -2 \end{pmatrix}$$

Handworked Example

- Check bidirectional stability for first association
  - $X_1^T W = (4\ {-4}\ {-6}\ 6) \rightarrow (1\ {-1}\ {-1}\ 1) = Y_1^T$
  - $Y_1^T W^T = (-4\ 4\ {-4}\ 4\ {-4}) \rightarrow (-1\ 1\ {-1}\ 1\ {-1}) = X_1^T$
  - $E(X_1, Y_1) = -\ X_1^T W Y_1 = -20 = -\ Y_1^T W^T X_1$

- Verify for the second…

Accuracy of Recall Based on Hamming Distance

- Probe vector $X' = (-1\ 1\ 1\ 1\ -1)^T$ at a Hamming distance of 1 from X_1
  - Then: $(X')^T W = (4\ -4\ -2\ 2) \rightarrow (1\ -1\ -1\ 1) = Y_1^T$
  - X_1, Y_1 association recalled
- Probe vector $X'' = (-1\ -1\ 1\ 1\ -1)^T$ at a Hamming distance of 2 from X_1 and 1 from X_1^c
  - Then: $(X'')^T W = (4\ -4\ 2\ -2) \rightarrow (1\ -1\ 1\ -1) = (Y_1^c)^T$
  - X_1^c, Y_1^c association recalled

BAM Stability Analysis

- Is there a guarantee that the system will always fall to a stable attractor?

- Energy function:
  - Kosko suggests using an energy function which is the average of the of the forward and reverse signal energies:
    - $E(A,B) = -A^{T}WB$

BAM Stability Theorem

- Any real connection matrix **W** defines a BAM system that is bidirectionally stable.
- See text for proof.

Error Correction in BAMs

- As powerful as that of Hopfield networks
- Comes with a fundamental assumption:
  - Function continuity: $H(A_i, A_j)/n \approx H(B_i, B_j)/m$

$$X_i^T \mathbf{W} = nY_i^T + \sum_{\substack{k=1 \\ k \neq i}}^{Q} c_{ik} Y_k^T$$

Coefficient c_{ik} is computed from domain space inner product but applied to range space inner product. Therefore continuity must hold.

Correction Coefficients

- With the continuity assumption in place, the correction coefficients c_{ik} correct the noise terms in a way that is identical to the correction explained earlier in the context of Hopfield networks:

$$c_{ik} \gtreqless 0, \qquad \text{iff} \quad H(X_i, X_k) \lesseqgtr \frac{n}{2}$$

Memory Annihilation of Structured Maps in BAMs

- An important pathological case
- Under certain mild conditions, the memory of all associations is lost
  - Memory of encoded associations between sets of equidistant vectors (called structured maps) annihilates when the number of such associations exceeds a lower bound.
  - See text for details.

Continuous BAMs

- Discrete time treatment carries over to the continuous time case
- Assume that neuron signal functions are sigmoidal
- Each layer of neurons is governed by an additive dynamics system of equations:

$$\dot{x}_i = -a_i x_i + \sum_{j=1}^{m} w_{ji} \mathcal{S}_j(y_j) + I_i \qquad i = 1, \ldots, n$$

$$\dot{y}_j = -b_j y_j + \sum_{i=1}^{n} w_{ij} \mathcal{S}_i(x_i) + J_j \qquad j = 1, \ldots, m$$

Energy Function

- For continuous BAMs the energy function takes the form:

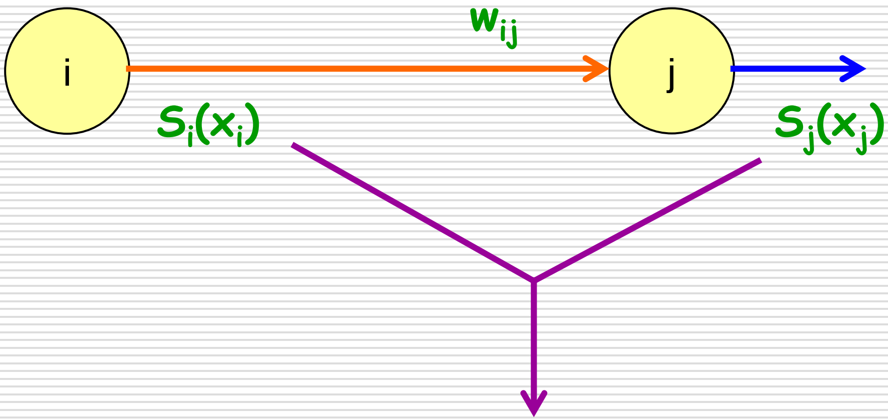
$$E(X,Y) = \sum_i \int_0^{x_i} \alpha_i \mathcal{S}_i'(\alpha_i)\, d\alpha_i - \sum_i \sum_j w_{ij} \mathcal{S}_i(x_i) \mathcal{S}_j(y_j)$$
$$- \sum_i \mathcal{S}_i(x_i) I_i + \sum_j \int_0^{y_j} \beta_j \mathcal{S}_j'(\beta_j) d\beta_j - \sum_j \mathcal{S}_j(y_j) J_j$$

- Time derivative of this function is negative: proves stability

Adaptive BAMs

- Employ Signal Hebbian Learning
- Hebb's observations translate to the following rules:
  - The synaptic efficacy between two simultaneously active neurons is selectively increased
  - The synaptic efficacy between two asynchronously active neurons is selectively decreased

Signal Hebb Law



Signal product dictates weight change

$$\dot{w}_{ij} = -w_{ij} + \mathcal{S}_i(x_i)\mathcal{S}_j(x_j)$$

Solution →

$$w_{ij}(t) = w_{ij}(0)e^{-t} + \int_0^t \mathcal{S}_i(\alpha)\mathcal{S}_j(\alpha)e^{-(t-\alpha)}\, d\alpha$$

Hebb Learning Generates Outer Products in the Limit

- Consider $s_i = s_j = 1$, or $s_i = s_j = -1$
- Then

$$w_{ij}(t) = w_{ij}(0)e^{-t} + \int_0^t e^{-(t-\alpha)}\, d\alpha$$
$$= w_{ij}(0)e^{-t} + (1 - e^{-t})$$

- Or $w_{ij}(t) \to 1$ as $t \to \infty$
- Similarly if $s_i = 1$, $s_j = -1$, or $s_i = -1$ $s_j = 1$ then $w_{ij}(t) \to -1$ as $t \to \infty$
- This clearly shows that the asymptotic encoding of a signal Hebb Law generates a component weight w_{ij} which is none other than the signal product $s_i s_j$, or the outer product generates

Adaptive BAM Equations

- Additive activation dynamics with Signal Hebb learning embedded

$$\dot{x}_i = -a_i x_i + \sum_{j=1}^{m} w_{ji} \mathcal{S}_j(y_j) + I_i \qquad i = 1, \ldots, n$$

$$\dot{y}_j = -b_j y_j + \sum_{i=1}^{n} w_{ij} \mathcal{S}_i(x_i) + J_j \qquad j = 1, \ldots, m$$

$$\dot{w}_{ij} = -w_{ij} + \mathcal{S}_i(x_i)\mathcal{S}_j(y_j) \qquad i = 1, \ldots, n; \quad j = 1, \ldots, m$$

Energy Function

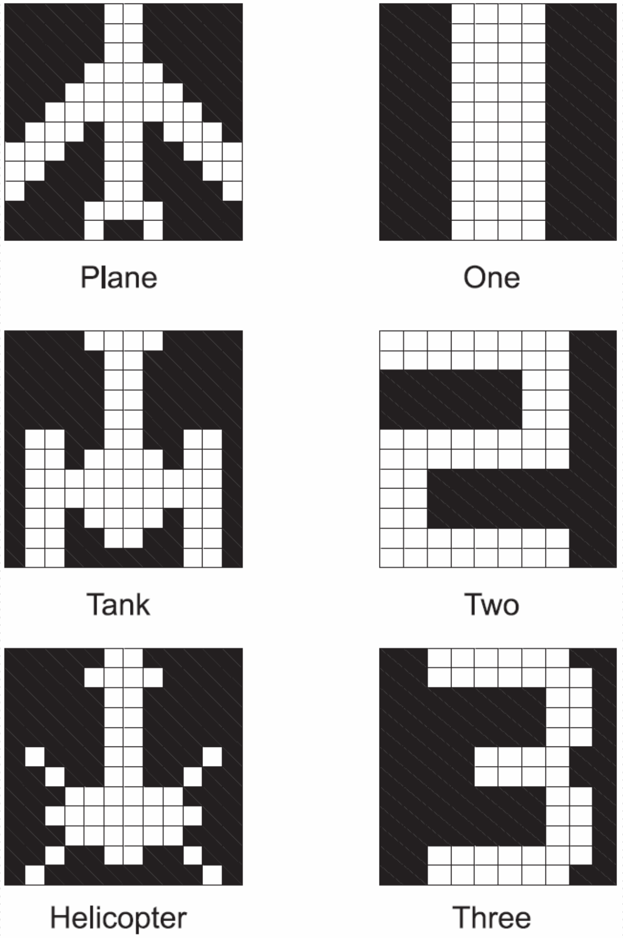
- Adaptive BAMs governed by the energy function

$$E(X, Y, W) = E(X, Y) + \frac{1}{2}\sum_i \sum_j w_{ij}^2$$

- Easy to show that the time derivative is negative, and adaptive BAMs settle down into a stable configuration
  - Adaptive resonance

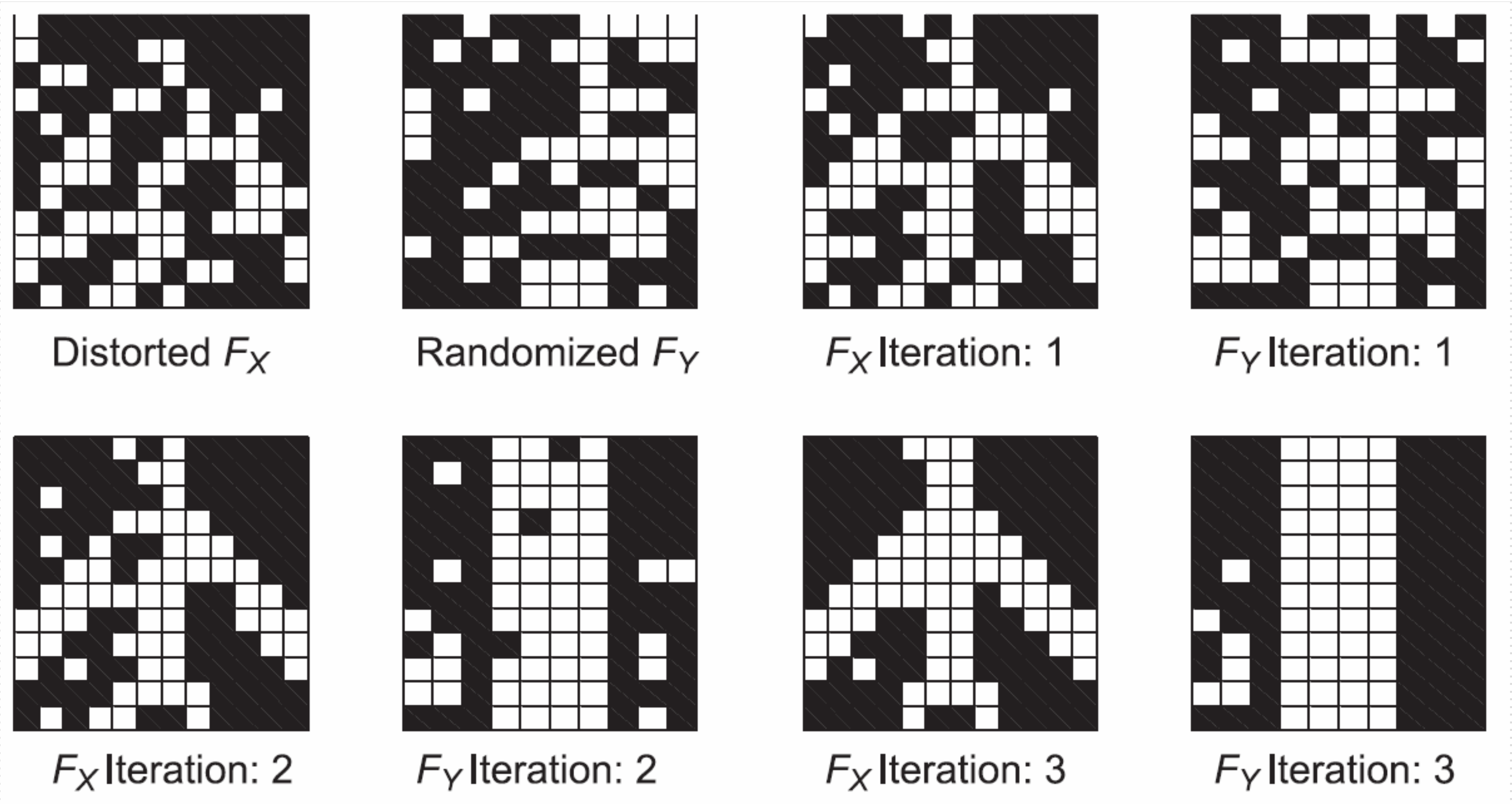
BAM Simulation Example

- Encode the pixel image associations as shown alongside
- Input image defined on 12×12 grid
- Output image defined on 12×10 grid



Simulation

- Presentation of a 40% distorted plane



Simulation

- Final association recalled