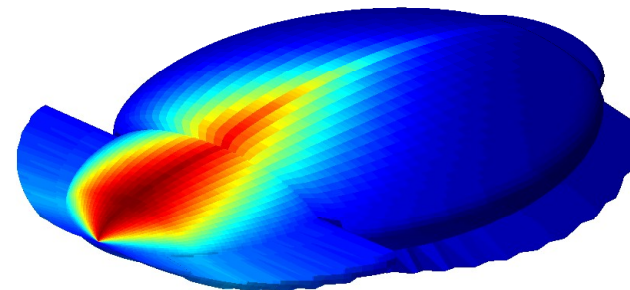
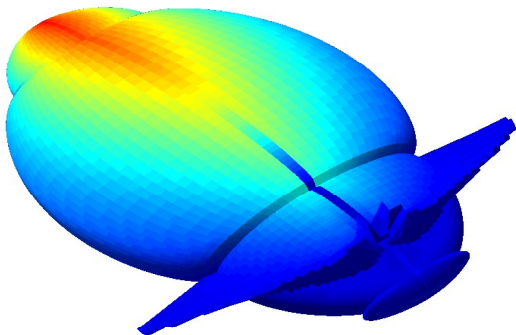


Lecture 12

Associative Memory



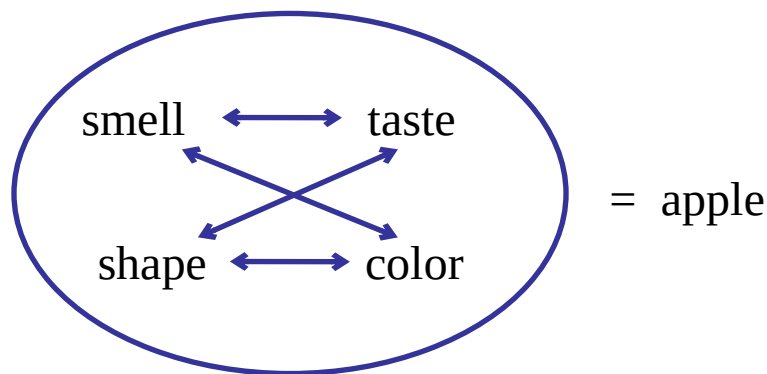
Associative Memory

A memory which stores items by association.

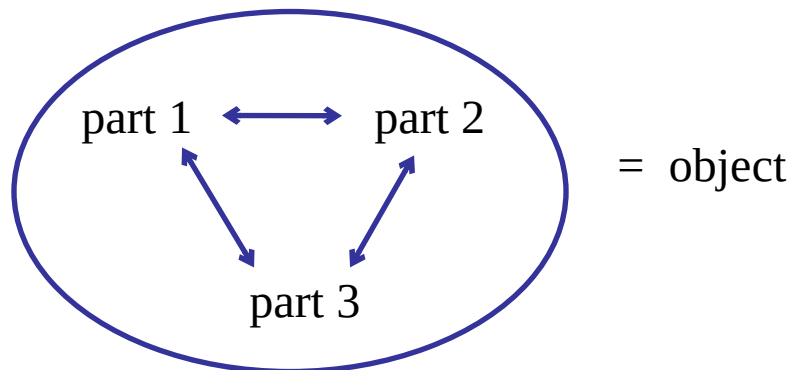
e.g. names \longleftrightarrow faces

The idea is that the entire memory can be recovered given a part of it,

e.g. a name recalls a face



an apple's shape and color recall its taste and smell



We will consider memories to be represented by patterns of activity over a group of neurons.

Associative Memory

- An associative memory neural network stores patterns of activity (e.g. binary vectors) by association.
 - When part of the stored memory is given as input, the network
recovers (recalls) the rest.

For this reason, associative memories are also called
content-addressable memories

i.e. memories which retrieve stored data based on part of their content rather than on the address of the storage location.

Indeed, in neural associative memories, there is no specific storage location.



“Holographic” memories

Two Types of Associative Memories

Hetero-associative:

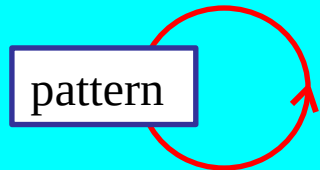


or



used for recall of
information associated
with the input pattern, or
for recoding.

Auto-associative:

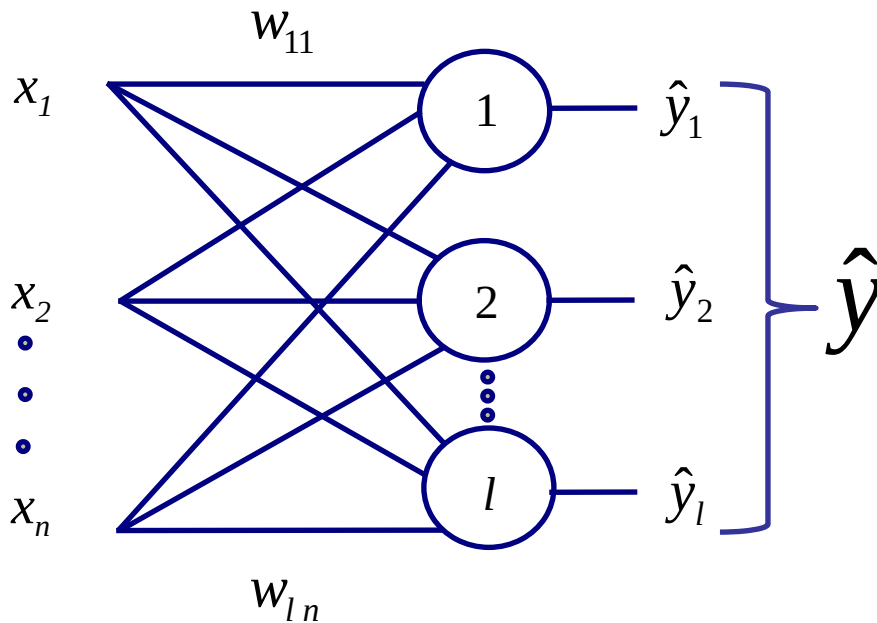


An incomplete or noisy version of the
pattern recalls the correct stored
pattern

used for

- pattern completion
- noise-suppression
- optimization

The Correlation Matrix Memory



neuron equation:

$$\hat{y}_i = \text{sgn} \left(\sum_{j=1}^n w_{ij} x_j + \theta_i \right)$$

weight equation:

$$w_{ij} = \frac{1}{n} \sum_{q=1}^N x_j^q y_i^q$$

Stimulus patterns

$$\begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^N \end{bmatrix} \in \{-1, 1\}^n \longrightarrow \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix} \in \{-1, 1\}^l$$

Response patterns

$$W = [w_{ij}] = \frac{1}{n} \sum_{q=1}^N x^q y^{qT}$$

correlation matrix

sum of outer products

We will take $\theta_i = 0 \quad \forall i$

Suppose pattern x^r is presented to the network

$$\text{net input to neuron } i \equiv s_i = \sum_{j=1}^n w_{ij} x_j^r$$

$$\therefore s_i = \sum_{j=1}^n \frac{1}{n} \sum_{q=1}^N x_j^q y_i^q x_j^r$$

$$= \underbrace{\frac{1}{n} y_i^r \sum_{j=1}^n x_j^r x_j^r}_{\text{signal term}} + \underbrace{\frac{1}{n} \sum_{q \neq r} y_i^q \sum_{j=1}^n x_j^q x_j^r}_{\text{noise term} = C}$$

$$\text{since } x_j^r x_j^r = 1$$

$$s_i = y_i^r + C$$

$$\text{Now } \hat{y}_i = \text{sgn}(s_i) = \text{sgn}(y_i^r + C)$$

$$\therefore \text{if } \text{sgn}(y_i^r + C) = y_i^r \rightarrow \hat{y}_i = y_i^r$$

i.e. C does not change the sign of s_i
i.e. no error in \hat{y}_i

Condition for error

$$C > 1 \quad \text{if} \quad y_i^r = -1$$

$$C < -1 \quad \text{if} \quad y_i^r = 1$$

or

$$-y_i^r C > 1 \quad \text{for} \quad y_i^r = \pm 1$$

i.e., # matched bits = # mismatched bits

If the x^q are orthogonal, $\sum_j x_j^q x_j^r = 0 \quad \forall q \neq r$

i.e., input is actually x^r

and there are no errors:

$$y_i = \text{sgn}(y_i^r) = y_i^r$$

If x^q are not orthogonal, recall is still correct if $-y_i^r C$ is not too large (> 1)

Using only orthogonal x^q would be too restrictive, so it is good that the system allows us some latitude.

What if the input has some errors?

Suppose input = \hat{x}^r which differs from x^r in m bits

$$\text{i.e. } \sum_{j=1}^n \hat{x}_j^r x_j^r \equiv n' = (n - m) - m = n - 2m$$

$$- n \leq n' \leq n$$

($n-m$ bits match, giving 1s
 m bits don't, giving -1s)

Define $\frac{n'}{n} = \phi$ (called the direction cosine)

Repeating the earlier analysis

$$s_i = \frac{1}{n} y_i^r \sum_{j=1}^n \hat{x}_j^r x_j^r + \frac{1}{n} \sum_{q \neq r} y_i^q \sum_{j=1}^n \hat{x}_j^r x_j^q$$

$$= \frac{1}{n} y_i^r \sum_{j=1}^n \hat{x}_j^r x_j^r + C' = \phi y_i^r + C'$$

$$\text{i.e. } s_i = \left[\frac{\# \text{correct bits} - \# \text{incorrect bits}}{\# \text{total bits}} \right] y_i^r + C'$$

Error condition:

$$\text{sgn}(\phi y_i^r + C') \neq y_i^r$$

$$\text{if } C' > \phi \text{ and } y_i^r = -1$$

$$\text{or } C' < -\phi \text{ and } y_i^r = 1$$

which, as before gives the general error condition

$$- y_i^r C' > \phi \Rightarrow \text{error in } y_i$$

Thus, if $- y_i^r C' < \phi$, y_i is still recalled correctly

The network has the ability to correctly recall the response pattern even if the stimulus pattern is somewhat corrupted.

As # stimulus error bits increase, ϕ decreases, and eventually recall is lost

What if some of the weights are missing?

Suppose the weight vector for neuron i , $w_i = [w_{i1} \ w_{i2} \ \dots \ w_{in}]$ is corrupted so that n' of the weights become 0

Denote the corrupted weight vector as $\hat{w}_i = [\hat{w}_{i1} \ \hat{w}_{i2} \ \dots \ \hat{w}_{in}]$

Repeating the earlier analysis

$$\begin{aligned} s_i &= \sum_{j=1}^n \hat{w}_{ij} x_j^r = \sum_{j=1}^n w_{ij} x_j^r - \sum_{\text{bad } k} w_{ik} x_k^r \\ &= \frac{1}{n} y_i^r \sum_{j=1}^n x_j^r x_j^r - \frac{1}{n} y_i^r \sum_k x_k^r x_k^r + \underbrace{\frac{1}{n} \sum_{q \neq r} y_i^q \sum_{j=1}^n x_j^r x_j^q - \frac{1}{n} \sum_{q \neq r} y_i^q \sum_k x_k^r x_k^q}_{C'} \end{aligned}$$

$$= y_i^r - \frac{n'}{n} y_i^r + C' = \frac{n - n'}{n} y_i^r + C'$$

$$\text{i.e. } s_i = \left[\frac{\# \text{ total bits} - \# \text{ bits with missing weights}}{\# \text{ total bits}} \right] y_i^r + C'$$

Error condition:

$$\text{sgn}\left(\frac{n - n'}{n} y_i^r + C'\right) \neq y_i^r$$

$$\text{if } C' > \frac{n - n'}{n} \text{ and } y_i^r = -1$$

$$\text{or } C' < -\frac{n - n'}{n} \text{ and } y_i^r = 1$$

which, as before gives the general error condition

$$- y_i^r C' > \frac{n - n'}{n} \Rightarrow \text{error in } y_i$$

Thus, if $- y_i^r C' < \frac{n - n'}{n}$, y_i is still recalled correctly

The network has the ability to correctly recall the response pattern even if some of the weights are missing

As # missing weights increase, $n - n' \rightarrow 0$ and eventually recall is lost

A similar argument can be made if weights are corrupted by noise, some inputs are set to 0, etc.

Conclusion:

This associative memory is **very robust**, and can produce correct recall even even when some inputs or weights are incorrect or missing.

Some comments on the weight equation

it is not strictly necessary to use $\frac{1}{n}$ in the learning equation

The necessary condition is: $w_{ij} \propto \sum_{q=1}^N y_i^q x_j^q$

Thus, the weights can be determined by learning as follows:

- 1 set $w_{ij} = 0 \quad \forall i, j$
- 2 For $q = 1$ to N
 $w_{ij} = w_{ij} + \eta y_i^q x_j^q$

The rule

$$\Delta w_{ij} \propto y_i^q x_j^q$$

is the Hebb Rule

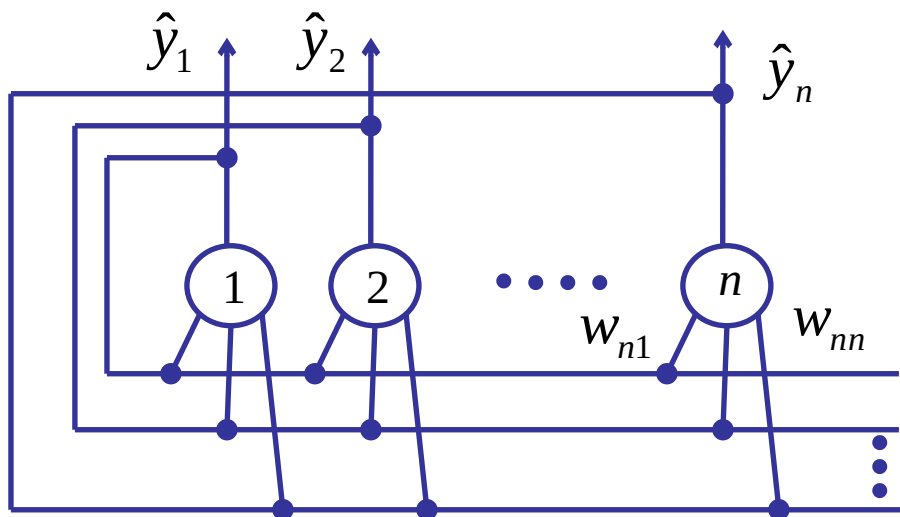
if $\eta = \frac{1}{n}$, we get the weight equation given earlier

In general, the Hebb Rule for associative learning states:

- if the activity of i and j is correlated
→ increase w_{ij}
- if the activity of i and j is anti-correlated
→ decrease w_{ij}
- if i and j are uncorrelated → $w_{ij} = 0$

Many variants of Hebbian Learning exist in the literature, especially for the case of 0/1 neurons, which are biologically more plausible.

Recurrent Autoassociative Memory



stored patterns : $\left\{ y^q = [y_1^q \ y_2^q \ \cdots y_n^q]^T \right\}_{q=1}^N$

$$\hat{y}(t+1) = \text{sgn} \left(\sum_{j=1}^n w_{ij} \hat{y}_j(t) + \theta_i \right)$$

$$s_i(t+1)$$

same as the
correlation matrix memory

$$w_{ij} = \frac{1}{n} \sum_{q=1}^N y_i^q y_j^q \quad (\text{Hebb Rule})$$

The weights associate each y^q with itself.

If the system is started with $\hat{y}(0) \approx y^q$
it recovers y^q over a few iterations

→ Attractor Network

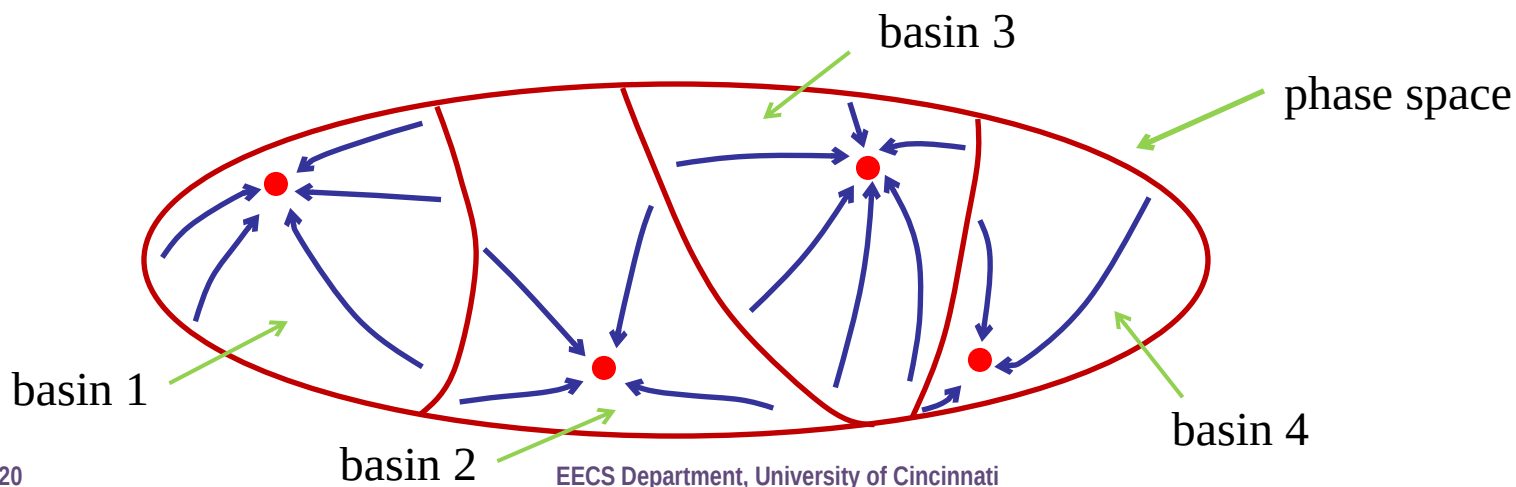
Invariant set: A set \mathcal{V} in phase space such that if the system is in \mathcal{V} at time t_0 , it remains in \mathcal{V} for all time $t > t_0$.

e.g. an equilibrium point.

Attractor: An invariant set in phase space towards which a dynamical system is attracted from certain initial conditions.

e.g. a stable fixed point or stable limit cycle.

Basin of attraction: The basin of attraction for an attractor, a , is the set of all initial conditions from which a system is attracted to a .



Recurrent associative memories learn by embedding the desired memory patterns as attractors of the network dynamics.

Given an initial pattern, $\hat{y}^{(0)}$, the network relaxes to the attractor whose basin of attraction includes $\hat{y}^{(0)}$.

→ pattern completion
pattern recall

Hazards

- $\hat{y}^{(0)}$ may not be in the desired BOA.
- The system may have unwanted attractors (spurious memories)

Network Dynamics

Synchronous Updating

$$\hat{y}(t+1) = \text{sgn}(w\hat{y}(t))$$

$$\hat{y} = [\hat{y}_1 \ \hat{y}_2 \ \cdots \hat{y}_n]^T$$

i.e. all \hat{y}_i are updated in parallel at the same time

$$\hat{y}(t+1) = \text{sgn}(w_i^T \hat{y}(t))$$

Asynchronous updating

Bits are updated one at a time, in random order or sequentially.

→ we no longer have strict synchronous time indexed by t .

In a recurrent network with $w_{ij} = w_{ji} \ \forall i, j$ (symmetric network)

- Synchronous update → fixed point or 2-cycle
- Asynchronous update → fixed point

Discrete-Time Hopfield Network

$$w_{ij} = \begin{cases} \frac{1}{n} \sum_{q=1}^N y_i^q y_j^q & i \neq j \\ 0 & i = j \end{cases} \longrightarrow w_{ij} = w_{ji} \quad \forall i, j$$

or, defining

$$w = \begin{bmatrix} w_{11} & w_{12} & \cdot & \cdot & \cdot & w_{1n} \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ w_{n1} & \cdot & \cdot & \cdot & \cdot & w_{nn} \end{bmatrix} = \begin{bmatrix} w_1^T \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ w_n^T \end{bmatrix}$$

$$w = \frac{1}{n} \left[\sum_{q=1}^N y^q y^{q^T} \right] - \frac{N}{n} I \longleftarrow \begin{matrix} n \times n \\ \text{Identity matrix} \end{matrix}$$

$$\hat{y}(t+1) = \text{sgn}(w_i^T \hat{y}(t))$$

Are the stored memories attractors?

Case I: Only one stored pattern y^1

$$w_{ij} = \frac{1}{n} y_i^1 y_j^1$$

$$s_i(t+1) = \sum_j w_{ij} \hat{y}_j(t)$$

$$= \frac{1}{n} \sum_j y_i^1 y_j^1 \hat{y}_j(t) = \frac{1}{n} y_i^1 \sum_j y_j^1 \hat{y}_j(t)$$

$$= y_i^1 \frac{1}{n} \left[\begin{array}{cc} \# \hat{y}(t) & \# \hat{y}(t) \\ \text{bits matching } y^1 & - \text{ bits not matching } y^1 \end{array} \right]$$

→ if the majority of $\hat{y}(t)$ bits match y^1 bits, $\text{sgn}(s_i) = y_i^1$ → recall of y^1

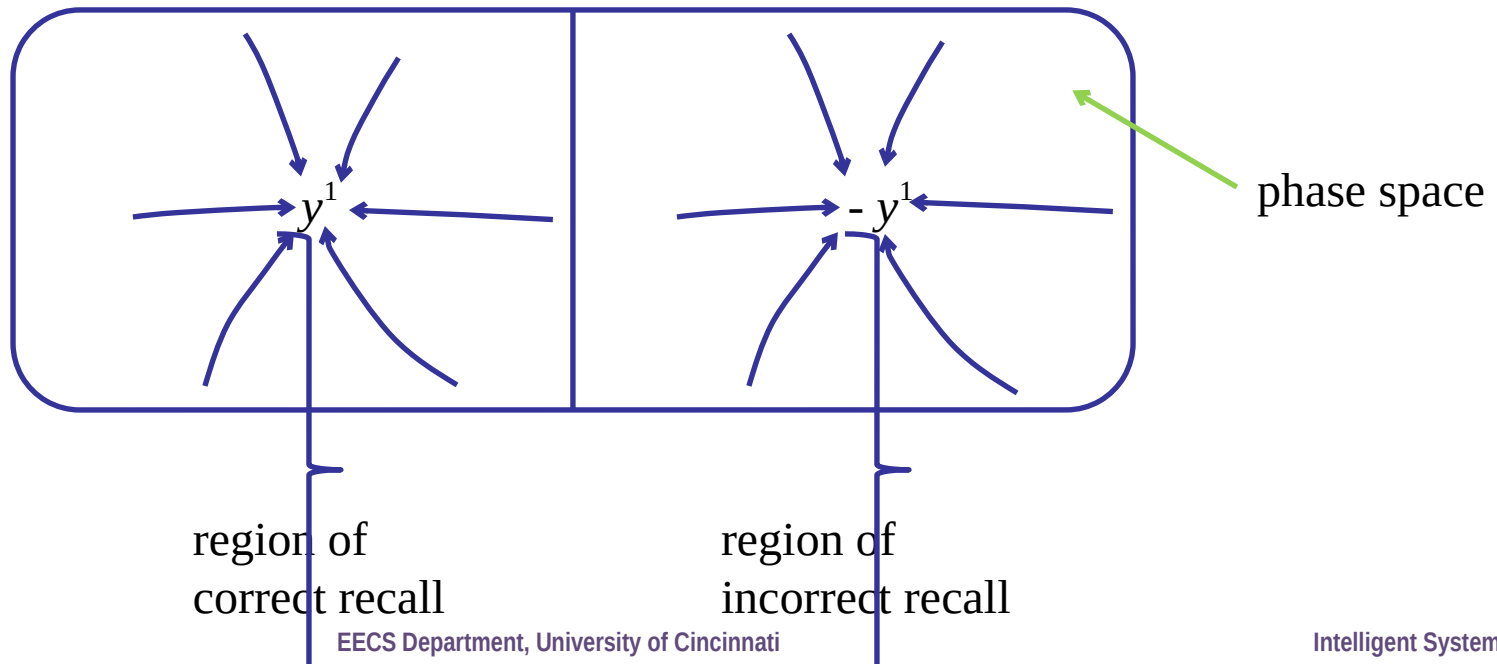
∴ if $\hat{y}(0)$ matches y^1 in $> \frac{n}{2}$ bits then $\hat{y}(1) = y^1 \longrightarrow$ recall

Are the stored memories the *only* attractors?

What happens if $\hat{y}(0)$ matches y^1 in $< \frac{n}{2}$ bits?

In that case $\hat{y}_i(1) = -y_i^1 \quad \forall i$

and $-y_i^1$ is also stable ← spurious memory



Case II: Multiple Patterns

$$w_{ij} = \frac{1}{n} \sum_{q=1}^N y_i^q y_j^q$$

Let us examine the recall of a particular pattern, y^r

Let
$$\frac{1}{n} \sum_{i=1}^n \hat{y}_i(t) y_i^r = \phi^r(t)$$

$$\begin{aligned} s_i(t+1) &= \sum_j w_{ij} \hat{y}_j(t) \\ &= \frac{1}{n} \sum_{j=1}^n \sum_{q=1}^N y_i^q y_j^q \hat{y}_j(t) \end{aligned}$$

$$\begin{aligned} &= \frac{1}{n} y_i^r \sum_j y_j^r \hat{y}_j(t) + \frac{1}{n} \sum_{q \neq r} y_i^q \sum_j y_j^q \hat{y}_j(t) \\ &= \phi^r(t) y_i^r + C_i(t) \end{aligned}$$

if $-y_i^r C_i(t) < \phi^r(t)$ (i.e. if $-y_i^r C_i(t)$ is "small enough")
 $\text{sgn}(s_i(t+1)) = y_i^r \Rightarrow \hat{y}_i(t+1) = y_i^r$

if $-y_i^r C_i(t)$ is "small enough" for all i

\equiv if $\phi^r(t)$ is "sufficiently large" to overcome $-y_i^r C_i(t) \quad \forall i$

y_i^r is recalled in one step from $\hat{y}(t)$

$\phi^r(t)$ "sufficiently large" $\equiv \hat{y}(t)$ overlaps sufficiently with y^r

If this occurs for all i , then starting with a state sufficiently similar to y^r , the network will recall y^r

→ y^r has a finite basin of attraction in phase space

$\equiv \underline{y^r \text{ is an attractor}}$

Note that the basin size will be much smaller than in the single memory case.

Stability of Hopfield Networks

We have shown that the embedded memories are fixed points,
but not that the system always converges from any initial condition

To prove this, we can use a Lyapunov Function (or an energy function)

A Lyapunov function is a function of the system state that is:

- Constant at system equilibrium points
- A decreasing function of time when the system is not at equilibrium

Existence of a Lyapunov Function \rightarrow stable system

Lyapunov function for Hopfield network:

$$J = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \hat{y}_i \hat{y}_j$$

J.J. Hopfield (1982) Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences, USA*, 79: 2554-2558.

$$J = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \hat{y}_i \hat{y}_j$$

Each ij combination occurs twice

For symmetric networks with $w_{ij} = w_{ji}$ (e.g., Hopfield networks):

$$J = -\sum_{i=1}^n \sum_{\substack{j=1 \\ j>i}}^n w_{ij} \hat{y}_i \hat{y}_j - \frac{1}{2} \sum_{i=1}^n w_{ii}$$

Each ij combination occurs once

$$= -\sum_{i=1}^n \sum_{j>i}^n w_{ij} \hat{y}_i \hat{y}_j - K$$

- Hopfield: $K = 0$
- Hebb: $K = N/2n$

Consider the case with asynchronous update – only bit k updated at time t

$$J(t) = - \sum_{i=1}^n \sum_{j>i}^n w_{ij} \hat{y}_i(t) \hat{y}_j(t) - K$$

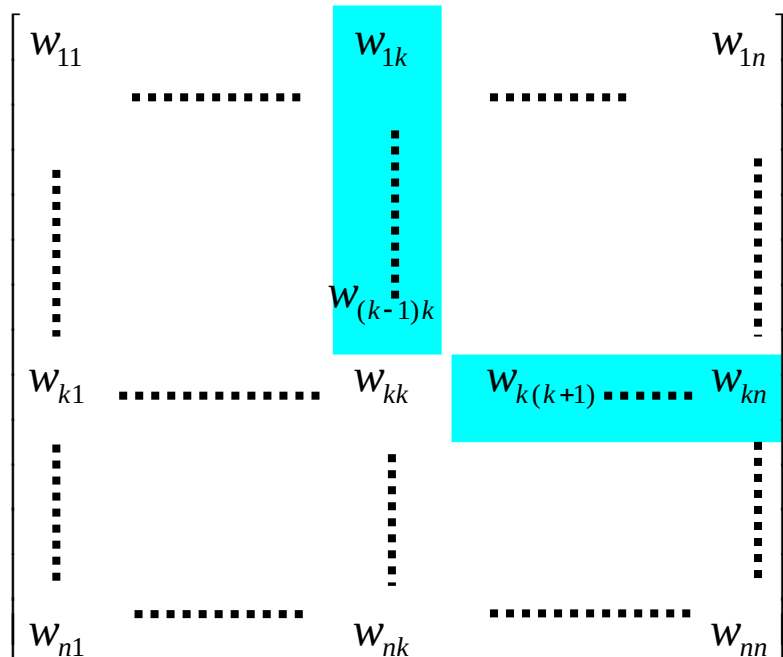
This remained unchanged

$$J(t+1) = J(t) - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t+1) \right)$$

Remove the old terms
involving neuron k

Add the new terms
involving neuron k

Terms affected by the update



Note that the w_{kk} term is excluded
because it is covered in the constant
 K which remains unchanged.

The weights involved are highlighted.

Consider the case with asynchronous update – only bit k updated at time t

$$J(t) = - \sum_{i=1}^n \sum_{j>i}^n w_{ij} \hat{y}_i(t) \hat{y}_j(t) - K$$

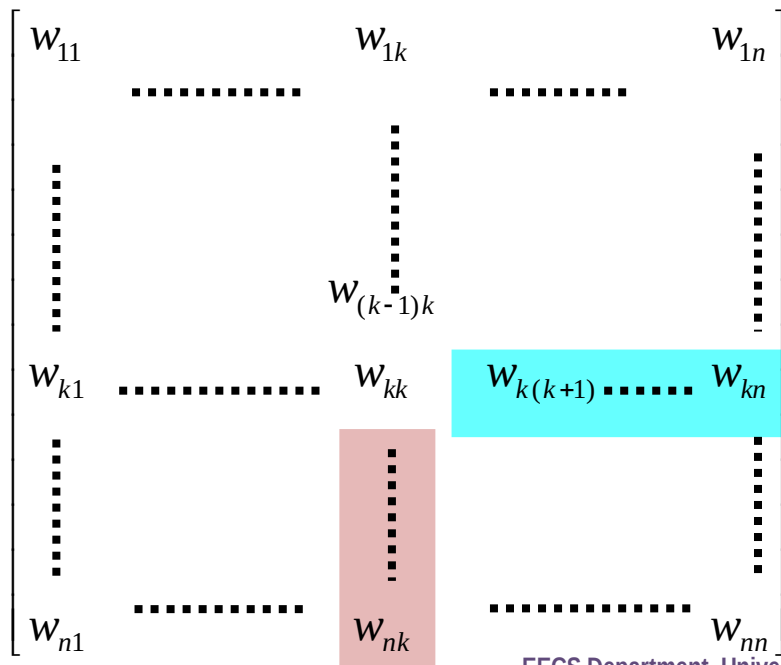
This remained unchanged

$$J(t+1) = J(t) - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t+1) \right)$$

Remove the old terms
involving neuron k

Add the new terms
involving neuron k

Terms affected by the update



Note that the w_{kk} term is excluded because it is covered in the constant K which remains unchanged.

The weights involved are highlighted.

Also, in the equation, we have chosen to count the $w_{(k+1)k} \dots w_{nk}$ terms instead of the $w_{k(k+1)} \dots w_{kn}$ terms, which are the same

$$J(t+1) - J(t) = - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t+1) \right)$$

Case I: $\hat{y}_k(t+1) = \hat{y}_k(t)$ (bit k remains unchanged by the update)

$$J(t+1) - J(t) = - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) = 0$$

J remains unchanged

Case II: $\hat{y}_k(t+1) = -\hat{y}_k(t)$ (bit k is flipped by the update)

$$\begin{aligned} J(t+1) - J(t) &= - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t+1) \right) \\ &= - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) (-\hat{y}_k(t)) \right) \end{aligned}$$

$$J(t+1) - J(t) = - \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(- \sum_{i \neq k} w_{ik} \hat{y}_i(t) (-\hat{y}_k(t)) \right)$$

$$= \left(\sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right) + \left(\sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) \right)$$

$$= 2 \sum_{i \neq k} w_{ik} \hat{y}_i(t) \hat{y}_k(t) = 2 \sum_{i \neq k} w_{ki} \hat{y}_i(t) \hat{y}_k(t) \quad \text{Replace } w_{ik} \text{ with } w_{ki}$$

$$= 2 \hat{y}_k(t) \sum_{i \neq k} w_{ki} \hat{y}_i(t)$$

$$= 2 \hat{y}_k(t) \left(\sum_{i=1}^n w_{ki} \hat{y}_i(t) - w_{kk} \hat{y}_k(t) \right) \quad \text{Summing over all } i \text{ including } k$$

$$= 2 \hat{y}_k(t) \sum_{i=1}^n w_{ki} \hat{y}_i(t) - 2 w_{kk}$$

$$J(t+1) - J(t) = 2\hat{y}_k(t) \underbrace{\sum_{i=1}^n w_{ki} \hat{y}_i(t)}_{\text{sum of weighted inputs}} - 2w_{kk}$$

$$\text{But } \hat{y}_k(t+1) = \text{sgn} \left(\sum_{i=1}^n w_{ki} \hat{y}_i(t) \right)$$

and $\hat{y}_k(t)$ and $\hat{y}_k(t+1)$ have opposite signs (Case II)

$$\Rightarrow 2\hat{y}_k(t) \sum_{i \neq k}^n w_{ki} \hat{y}_i(t) < 0$$

$$\text{Also, } w_{kk} = N/n > 0 \quad \Rightarrow \quad -2w_{kk} < 0$$

(Hebb)

$$\text{or } w_{kk} = 0 \quad \Rightarrow \quad -2w_{kk} = 0$$

(Hopfield)

A

B

$$\Rightarrow J(t+1) < J(t)$$

Thus, J either remains unchanged or decreases until there are no bit flips
→ **Equilibrium**

As we know, for a stored memory y^q

$$\begin{aligned}\hat{y}(t) = y^q &\Rightarrow \hat{y}(t+1) = y^q \\ &\Rightarrow J(t+1) = J(t) \quad \text{at} \quad \hat{y} = y^q\end{aligned}$$

When the system is not at equilibrium, i.e., bits are still flipping
 $\rightarrow J$ is still decreasing

J will keep decreasing until the system reaches equilibrium.

$\Rightarrow J$ is a Lyapunov function \Rightarrow System is stable

Conclusions about Hopfield network dynamics:

- Symmetric networks with asynchronous dynamics converge to fixed points
- Stored memories are not the only equilibria.
- Setting $w_{ii} = 0$ can reduce spurious equilibria.

Capacity of Hopfield Networks

How many memories can a Hopfield network store?

Suppose the network is already in state $y(t) = y^r$

The condition for an erroneous flip in bit i is:

$$Q_i^r = -y_i^r C_i(t) \geq 1 \quad \leftarrow \text{Since we are starting in } y^r, \phi^r = 1$$

$$Q_i^r = -y_i^r \left(\frac{1}{n} \sum_j \sum_{q \neq r} y_i^q y_j^q \hat{y}_j(t) \right) \geq 1$$

$$Q_i^r \approx \frac{1}{n} [\text{sum of } n(N-1) \text{ random } \pm 1 \text{ terms}]$$

$$\Rightarrow \text{mean of } Q_i^r \approx 0$$

This assumes that +1 and -1 are equally probable, which may not always be the case.

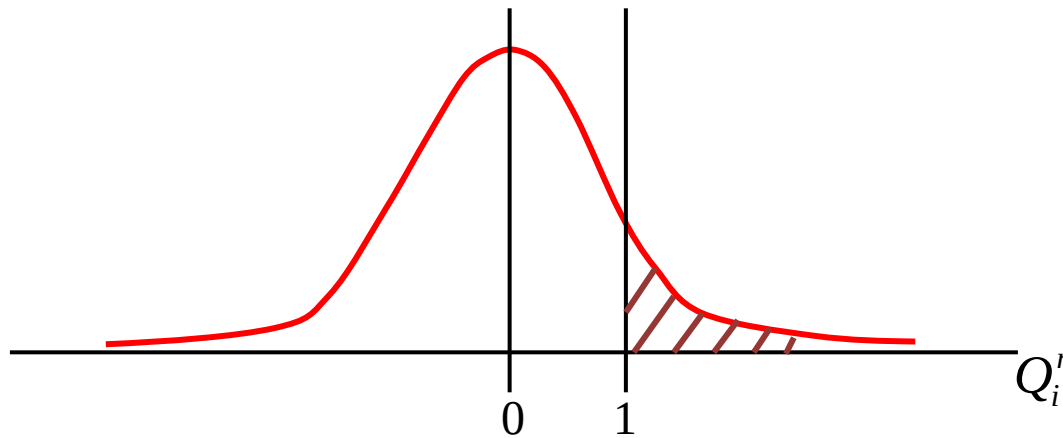
$$\text{variance of } Q_i^r \approx \frac{n(N-1)}{n^2} \approx \frac{N}{n}$$

Since each sample has a variance of 1 and there are $N(n-1)$ such samples

distribution of $Q_i^r \approx \text{Gaussian (Central Limit Theorem)}$

$$Q_i^r \sim G(0, N/n)$$

$$P(\text{error in bit } i) = P(Q_i^r \geq 1) \approx \frac{1}{\sqrt{2\pi N/n}} \int_1^{\infty} e^{-\frac{x^2}{2N/n}} dx$$



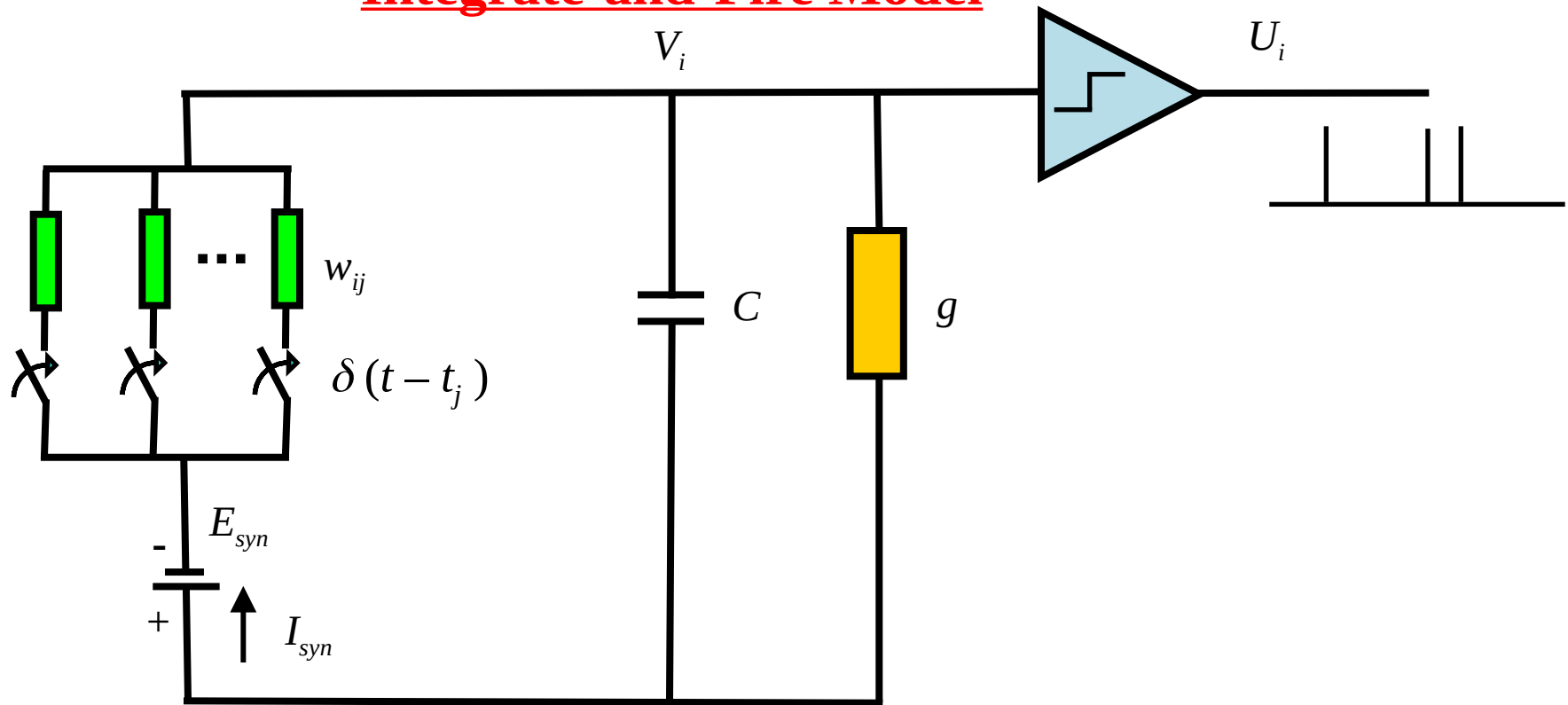
Solving for $P(\text{error}) < 0.01$, this gives

$$N_{\max} \approx 0.15n$$

In fact, $0.138n$ is a better estimate

\Rightarrow a 1000 neuron Hopfield network can store only about 130 memories

Integrate-and-Fire Model

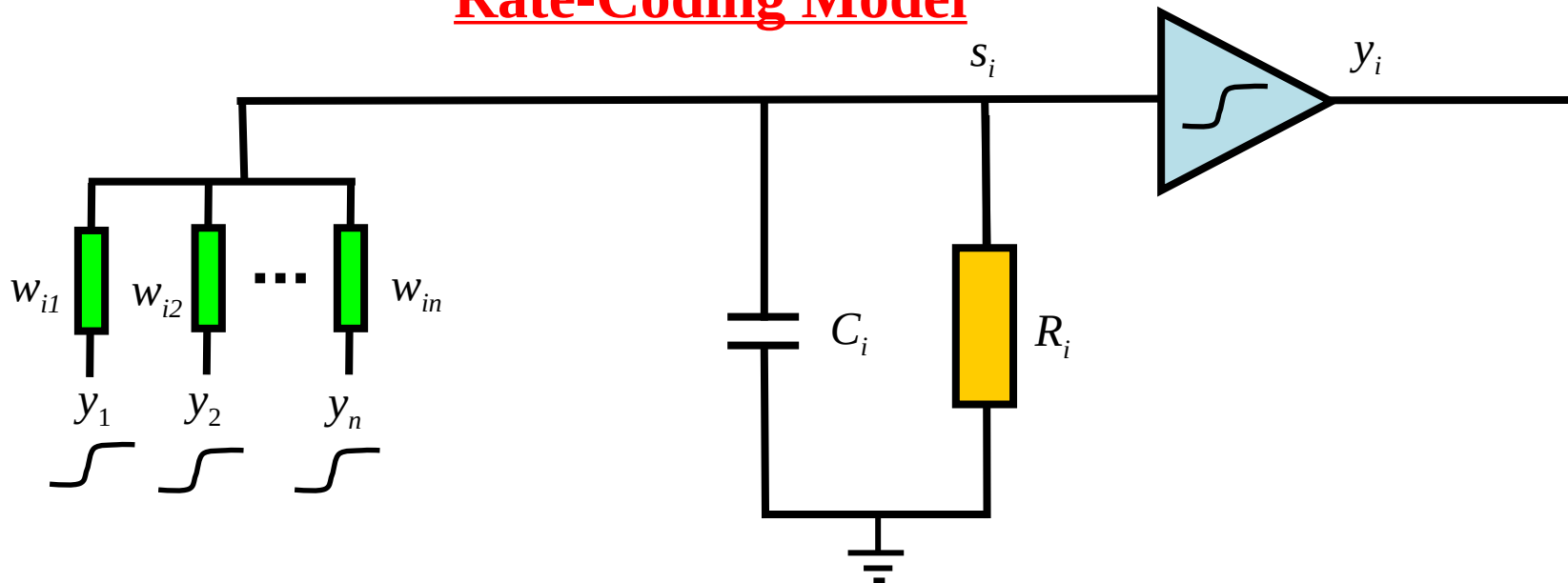


$$C \frac{dV_i}{dt} = -g V_i + \sum_j w_{ij} \sum_k \delta(t - t_{jk}) (E_{syn} - V_i)$$

$$U_i(t) = \begin{cases} 1 & \text{if } V_i(t) > \theta \\ 0 & \text{else} \end{cases}$$

V_i is reset after each spike

Rate-Coding Model



$$C_i \frac{ds_i}{dt} = -\frac{1}{R_i} s_i(t) + \sum_j w_{ij} y_j(t)$$

Defining $\tau_i = R_i C_i$ and measuring time in units of τ_i

State:
$$\frac{ds_i}{dt} = -s_i(t) + \sum_j w_{ij} y_j(t) = -s_i(t) + \sum_j w_{ij} f(s_j(t))$$

Output:
$$y_i(t) = f(s_i(t))$$
 where $f(x)$ is a sigmoid function

Note that this is a **nonlinear** differential equation

Stability and Attractors

It can be shown that if

$$f(x) = \tanh(\alpha x / 2)$$

The continuous-time Hopfield network has a Lyapunov function

$$J = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \hat{y}_i \hat{y}_j + \sum_{j=1}^n \frac{1}{\alpha R_i} \int_0^{y_j} \tanh^{-1}(y/2) dy$$

$$\text{As } \alpha \rightarrow \infty, \quad J \rightarrow -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \hat{y}_i \hat{y}_j$$

which is the Lyapunov function for the discrete-time network with the same weights

\Rightarrow The continuous-time network with extremely steep sigmoids has the same attractors as the equivalent discrete-time network.

Stochastic Model (Boltzmann Machine)

$$C_i \frac{ds_i}{dt} = -\frac{1}{R_i} s_i(t) + \sum_j w_{ij} y_j(t)$$

Note that this is a
nonlinear differential
equation

Defining $\tau_i = R_i C_i$ and measuring time in units of τ_i

State:
$$\frac{ds_i}{dt} = -s_i(t) + \sum_j w_{ij} y_j(t)$$

Output:
$$P(y_i(t)=1) = \frac{1}{1 + \exp(-s_i/T)}$$

where T is a **temperature** parameter.

When T is high, $P(y_i(t)=1) \approx 0.5$

Random update

$$\text{As } T \rightarrow 0 \quad P(y_i(t)=1) \rightarrow \begin{cases} 1 & \text{if } s_i > 0 \\ 0 & \text{if } s_i < 0 \end{cases}$$

Threshold function