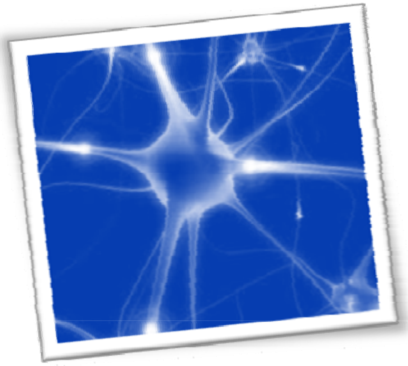
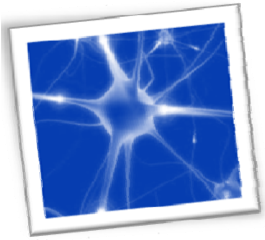
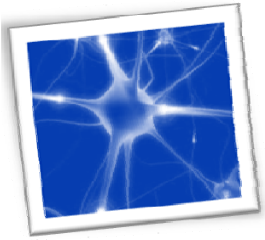
Lecture 23: Associative memory & Hopfield Networks





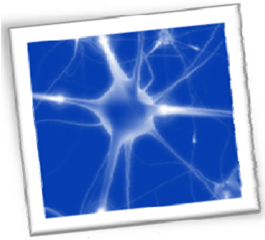
Feedforward/Feedback NNs

- Feedforward NNs
  - The connections between units do *not* form cycles.
  - Usually produce a response to an input quickly.
  - Most feedforward NNs can be trained using a wide variety of efficient algorithms.
- Feedback or recurrent NNs
  - There are cycles in the connections.
  - In some feedback NNs, each time an input is presented, the NN must iterate for a potentially long time before it produces a response.
  - Usually more difficult to train than feedforward NNs.



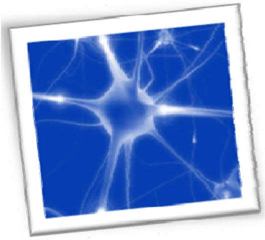
Supervised-Learning NNs

- Feedforward NNs
  - Perceptron
  - Adaline, Madaline
  - Backpropagation (BP)
  - Artmap
  - Learning Vector Quantization (LVQ)
  - Probabilistic Neural Network (PNN)
  - General Regression Neural Network (GRNN)
- Feedback or recurrent NNs
  - Brain-State-in-a-Box (BSB)
  - Fuzzy Conigitive Map (FCM)
  - Boltzmann Machine (BM)
  - Backpropagation through time (BPTT)



Unsupervised-Learning NNs

- Feedforward NNs
  - Learning Matrix (LM)
  - Sparse Distributed Associative Memory (SDM)
  - Fuzzy Associative Memory (FAM)
  - Counterprogation (CPN)
- Feedback or Recurrent NNs
  - Binary Adaptive Resonance Theory (ART1)
  - Analog Adaptive Resonance Theory (ART2, ART2a)
  - Discrete Hopfield (DH)
  - Continuous Hopfield (CH)
  - Discrete Bidirectional Associative Memory (BAM)

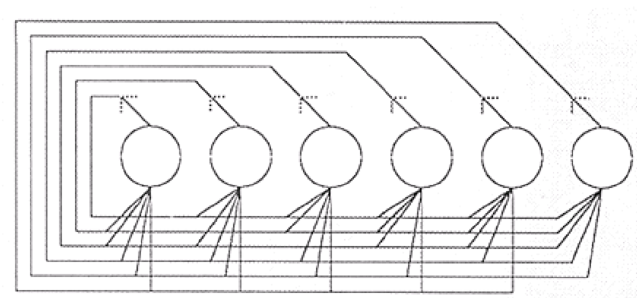


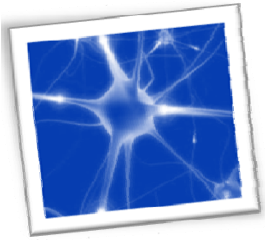
Neural Networks with Temporal Behavior

- Inclusion of feedback gives temporal characteristics to neural networks: **recurrent networks.**
- Two ways to add feedback:
  - **Local feedback**
  - **Global feedback**
- Recurrent networks can become unstable or stable.
- Main interest is in recurrent network's **stability: neurodynamics.**
- Stability is a property of the whole system: **coordination between parts is necessary.**

The Hopfield NNs

- In 1982, Hopfield, a Caltech physicist, mathematically tied together many of the ideas from previous research.
- A fully connected, symmetrically weighted network where each node functions both as input and output node.
- Used for
  - Associated memories
  - Combinatorial optimization



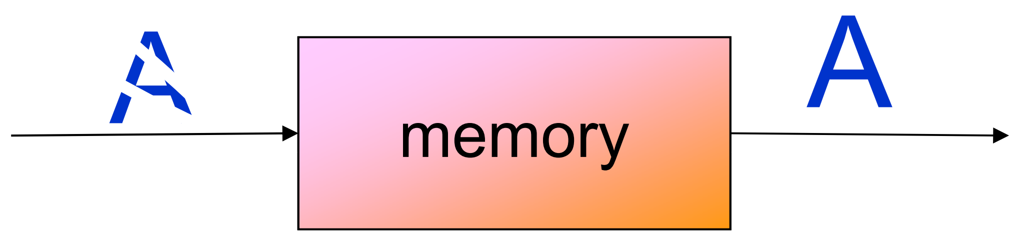


Associative Memories

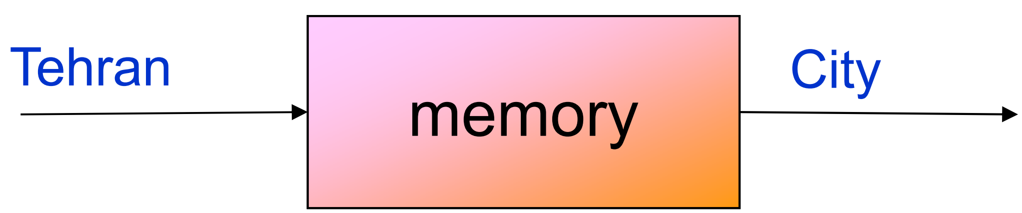
- An associative memory is a content-addressable structure that maps a set of input patterns to a set of output patterns.
- Two types of associative memory: *autoassociative* and *heteroassociative*.
- Auto-association
  - retrieves a previously stored pattern that most closely resembles the current pattern.
- Hetero-association
  - the retrieved pattern is, in general, different from the input pattern not only in content but possibly also in type and format.

Associative Memories

Auto-association

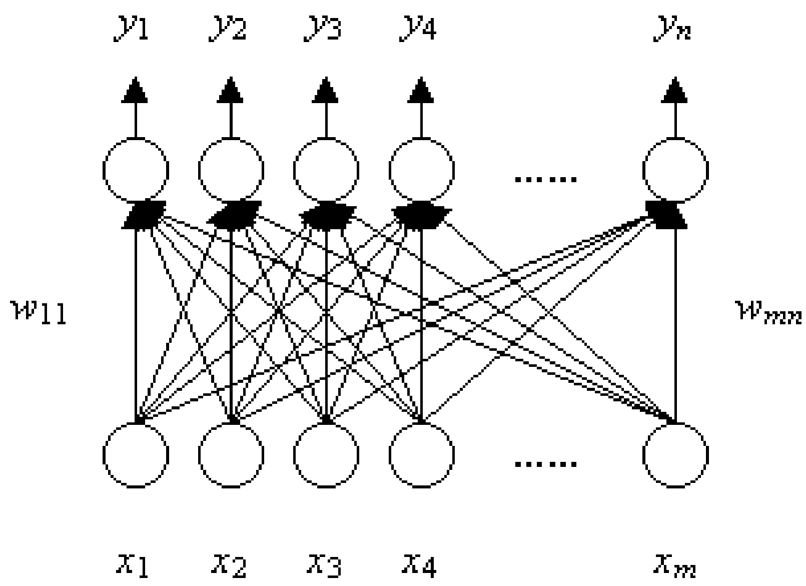


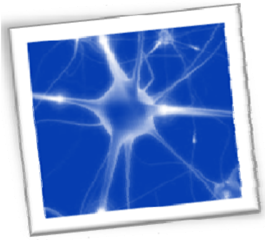
Hetero-association



Linear associative memory

- The linear associator is one of the simplest and first studied associative memory models
- A feedforward type network where the output is produced in a single feedforward computation
- The inputs connected to the outputs via the connection weight matrix $\mathbf{W} = [w_{ij}]_{m \times n}$
- It is **W** that stores the N different associated pattern pairs $\{(\mathbf{X}_k, \mathbf{Y}_k) \mid k = 1, 2, ..., N\}$ where the inputs and outputs are either -1 or +1





Linear associative memory

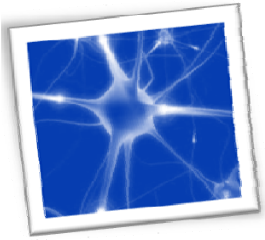
- Building an associative memory is constructing **W** such that when an input pattern is presented, the stored pattern associated with the input pattern is retrieved → encoding
- $\mathbf{W}_k$'s for a particular associated pattern pair ($\mathbf{X}_k$, $\mathbf{Y}_k$) are computed as $(w_{ij})_k = (x_i)_k(y_j)_k$
- Then

$$W = a\Sigma_{k=1}^{N} W_k$$

- a is the proportionality or normalizing constant to prevent the synaptic values from going too large when there are a number of associated pattern pairs to be memorized, usually $a = 1/N$.
- The connection weight matrix construction above simultaneously stores or remembers N different associated pattern pairs in a distributed manner.

Linear associative memory

- After encoding or memorization, the network can be used for *retrieval* → *decoding* ($X = W^{-1}Y$)
- Given a stimulus input pattern ***X***, decoding or recollection is accomplished by computing the net input to the output units using the previous formula.
- Then, y_j (-1 or +1) can be computed by thresholding the neuron j
- The input pattern may contain **errors** and **noise**, or may be an **incomplete version** of some previously encoded pattern.
- Nevertheless, when presented with such a corrupted input pattern, the network will retrieve the stored pattern that is closest to actual input pattern.
- So, the model is **robust** and **fault tolerant**, i.e., the presence of noise or errors results only in a mere decrease rather than total degradation in the performance of the network.
- Associative memories being robust and fault tolerant are the byproducts of having a number of processing elements performing highly parallel and distributed computations.



Linear associative memory

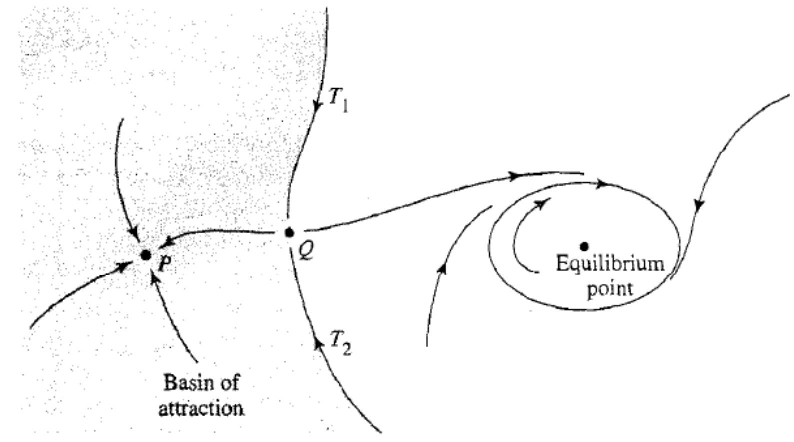
- Traditional measures of associative memory performance are its **memory capacity** and **content-addressability**.
- Memory capacity refers to the maximum number of associated pattern pairs that can be stored and correctly retrieved
- Content-addressability is the ability of the network to retrieve the correct stored pattern
- It can be shown that using Hebb's learning rule in building the connection weight matrix of an associative memory yields a significantly low memory capacity.
- Due to the limitation brought about by using Hebb's learning rule, several modifications and variants have been proposed to maximize the memory capacity.

Intuition

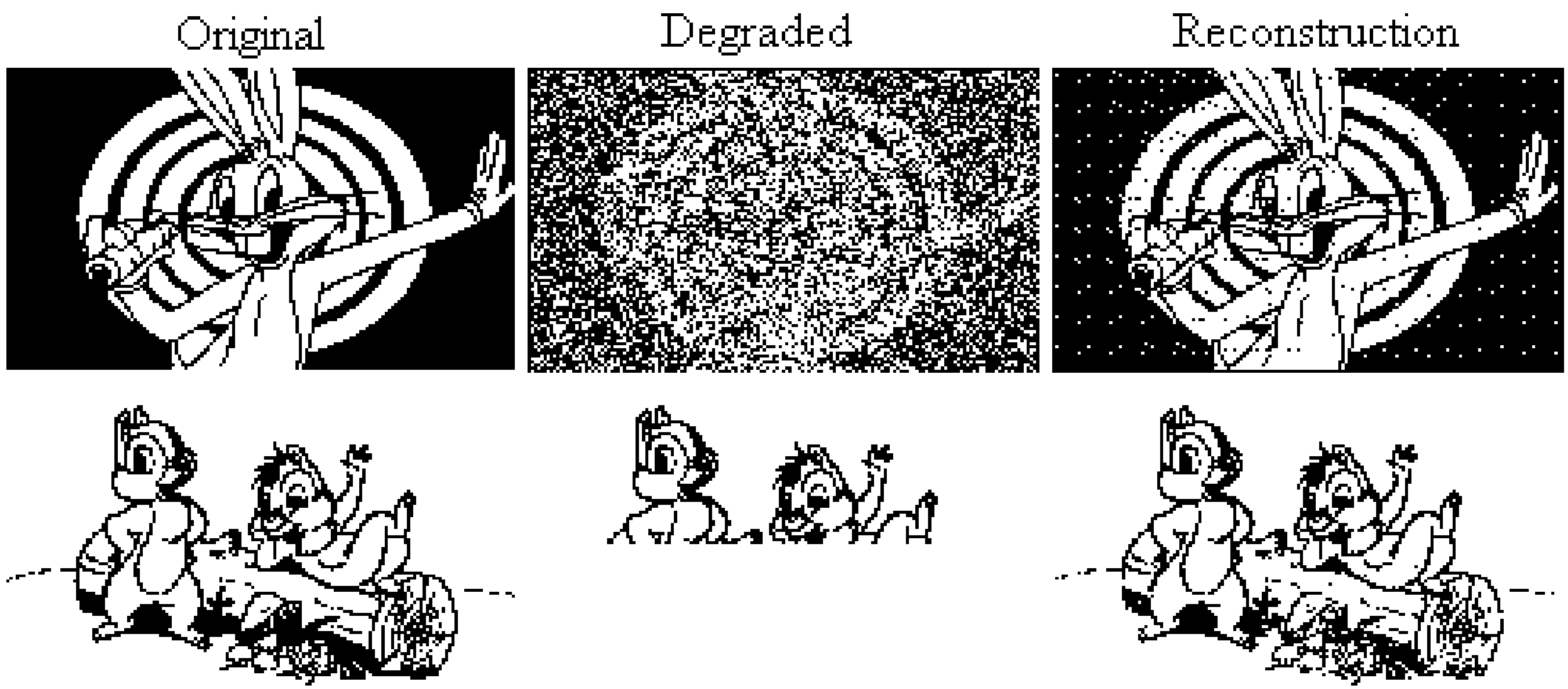
- **Manipulation of attractors as a recurrent neural network paradigm.**
- We can identify attractors with computational objects.
- In order to do so, we must exercise control over the location of the attractors in the state space of the system.
- A learning algorithm will manipulate the equations governing the dynamical behavior so that a desired location of attractors are set.
- One good way to do this is to use the **energy minimization** paradigm (e.g., by Hopfield).

Intuition

- N units with full connection among every node (no self-feedback).
- Given M input patterns, each having the same dimensionality as the network, can be memorized in attractors of the network.
- Starting with an initial pattern, the dynamic will converge toward the attractor of the basin of attraction where the initial pattern was placed.



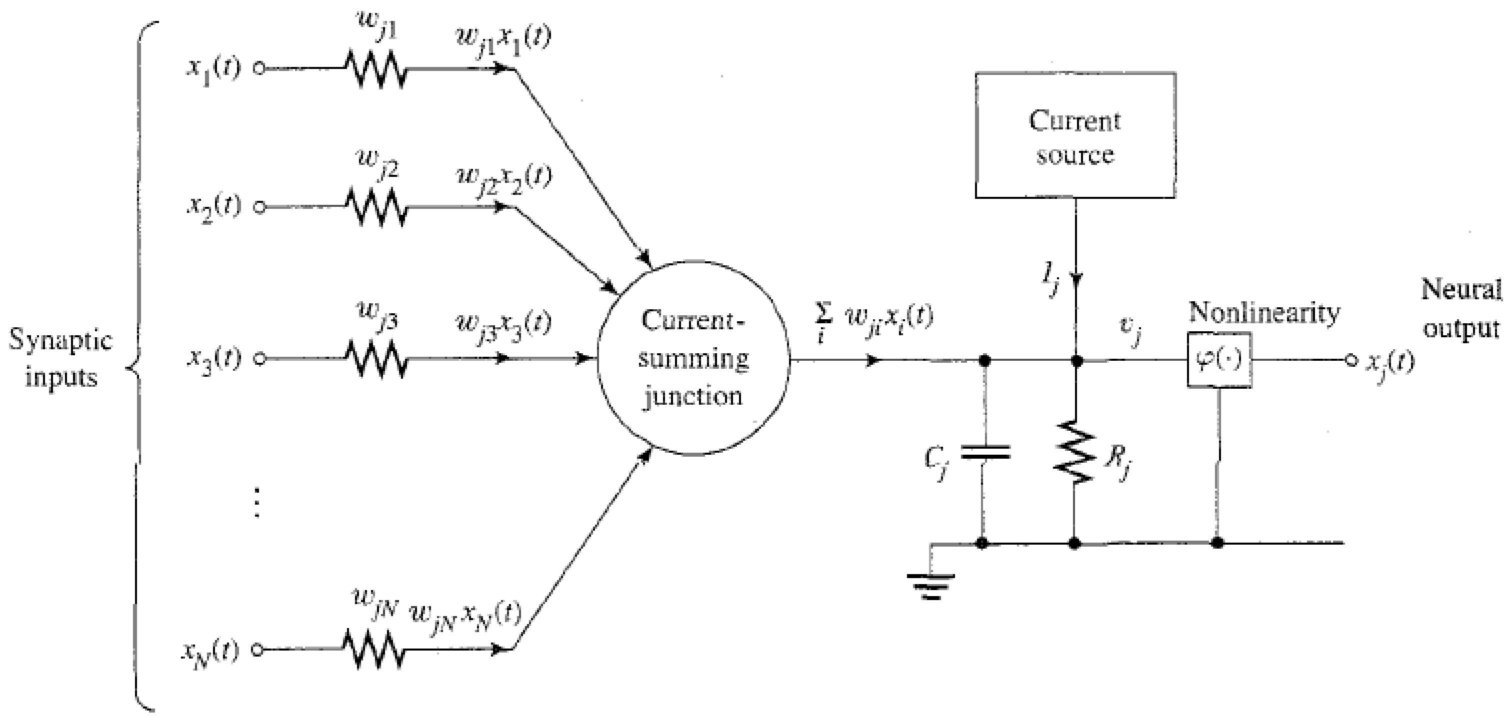
Example of using Hopfield NNs

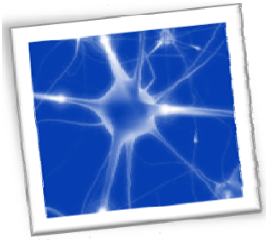


Hopfield network reconstructing degraded images from noisy (top) or partial (bottom) cues.

Additive model of a neuron

- Low input resistance
- Unity current gain
- High output resistance





Additive model of a neuron

- The total current flowing toward the input node of the nonlinear element is:

$$\Sigma_{i=1}^{N} w_{ji}\, x_i(t) + I_j$$

- The total current flowing away from the input nodes of the nonlinear element is

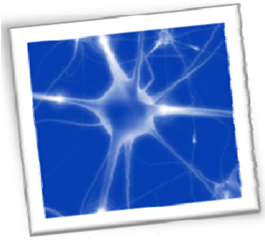
$$v_j(t)/R_j + C_j\, dv_j(t)/dt$$

- By applying Kirchoff's current law, we get

$$C_j\, dv_j(t)/dt + v_j(t)/R_j = \Sigma_{i=1}^{N} w_{ji}\, x_i(t) + I_j$$

- Given the induced local field $v_j(t)$, we may determine the output of neuron j by using the nonlinear relation

$$x_j(t) = \phi\ (v_j(n)\ )$$



Additive model of a neuron

- So, ignoring interneuron propagation time delays, we may define the dynamics of the network by the following system of **coupled first-order ODEs**:

$$C_j \, dv_j(t)/dt = -v_j(t)/R_j + \Sigma_{i=1}^{N} w_{ji} \, x_i(t) + I_j \quad ; \; j = 1,\ldots,N$$

$$x_j(t) = \phi\,(v_j(n))$$

For example: $\phi(v_j(n)) = 1/(1+\exp(-\,v_j(n))$

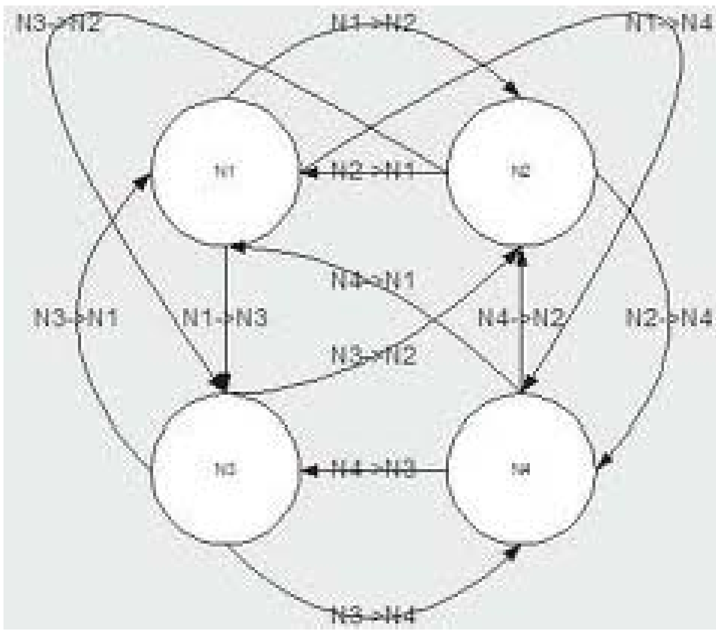
- This can be converted to:

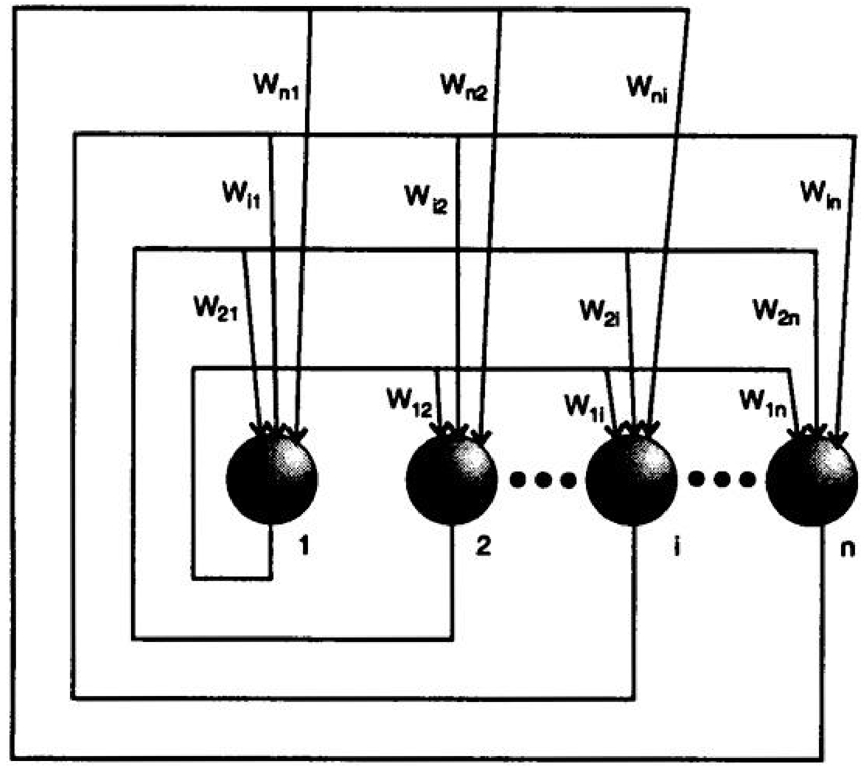
$$\mathbf{dx_j(t)/dt = -x_j(t)/R_j + \phi(\Sigma_{i=1}^{N} w_{ji} \, x_i(t)) + K_j}$$

Hopfield Model

- The Hopfield network (model) consists of a set of neurons and a corresponding set of unit delays, forming a multiple-loop feedback system
- The number of feedback loops is equal to the number of neurons.
- The output of each neuron is fed back via a unit delay element, to each of the other neurons in the network.
    - i.e., there is no self-feedback in the network.

Hopfield Architecture





Hopfield Model: learning

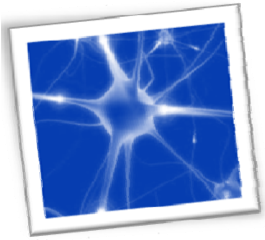
- Let t_1, t_2, …t_M denote a known set of N-dimensional fundamental memories
- The weights are computed using Hebb's rule

$$W_{ji} = (1/M) \Sigma_{q=1}^{M} t_{q,j} t_{q,i} \quad ; \quad j \neq i$$

$$W_{ji} = 0 \quad ; \quad j = i$$

$t_{q,i}$: the i-th element of t_q

- W becomes an N×N matrix
- The elements of vector t_q are equal to -1 or +1
- Once computed, the synaptic weights are kept fixed

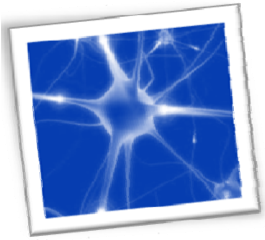


Hopfield Model: initialization

- Let t_{probe} denote an unknown N-dimensional input vector presented to the network
- The algorithm is initialized by setting

$$x_j(0) = t_{j,probe} \quad ; \quad j = 1,\ldots,N$$

- where $x_j(0)$ is the state of neuron j at time n = 0, and $t_{j,probe}$ is the j-th element of the evctor t_{probe}



Hopfield Model: iteration

- The states of the neurons (i.e., randomly and one at a time) are iterated asynchronously (difference equation for discrete-time and differential equations for continuous-time) until convergence. For discrete-time it is

$$x_j(n + 1) = \text{sgn}\,[\Sigma_{i=1}^{N} W_{ji} x_i(n)] \quad ; \quad j = 1,2, \ldots,N$$

- The convergance of the above rule is gauranteed (we will see why!)

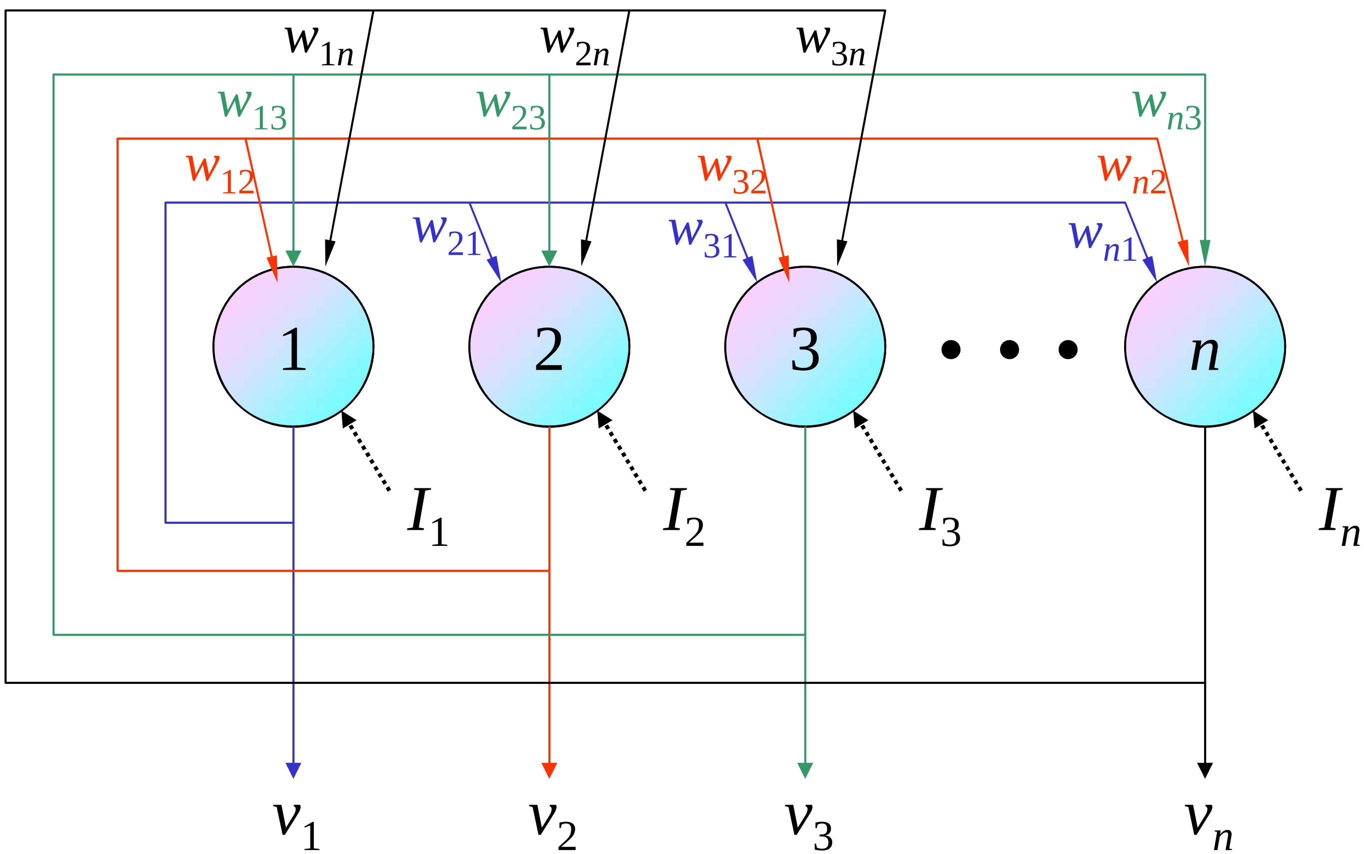
Hopfield Model: Outputting

- Let X_{fixed} denote the fixed point (stable state) computed at the end of the previous step
- The resulting output vector y of the network is set as

$$y = X_{fixed}$$

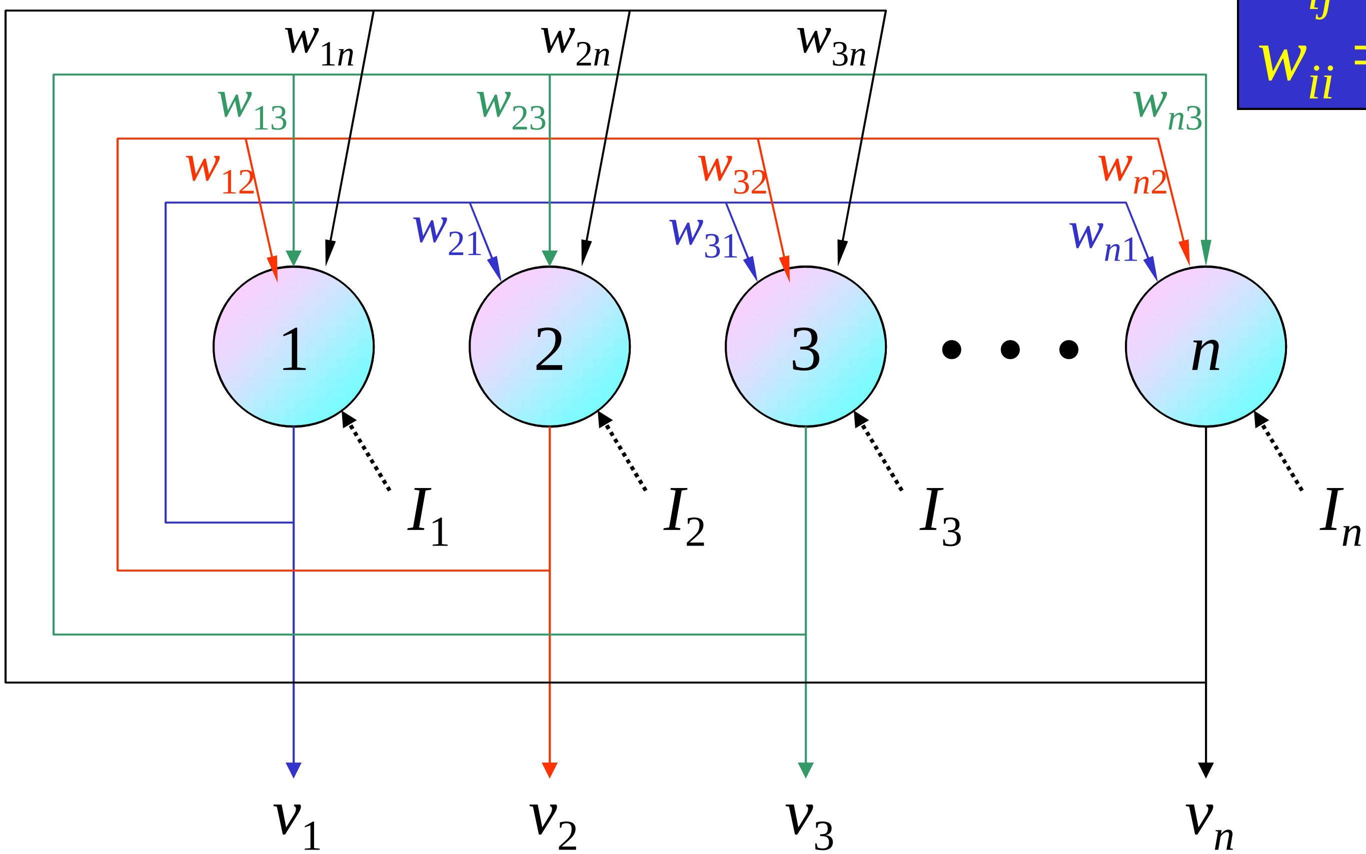
- Step 1 is the storage phase, while the last three steps constitute the retrieval phase

The Discrete Hopfield NNs



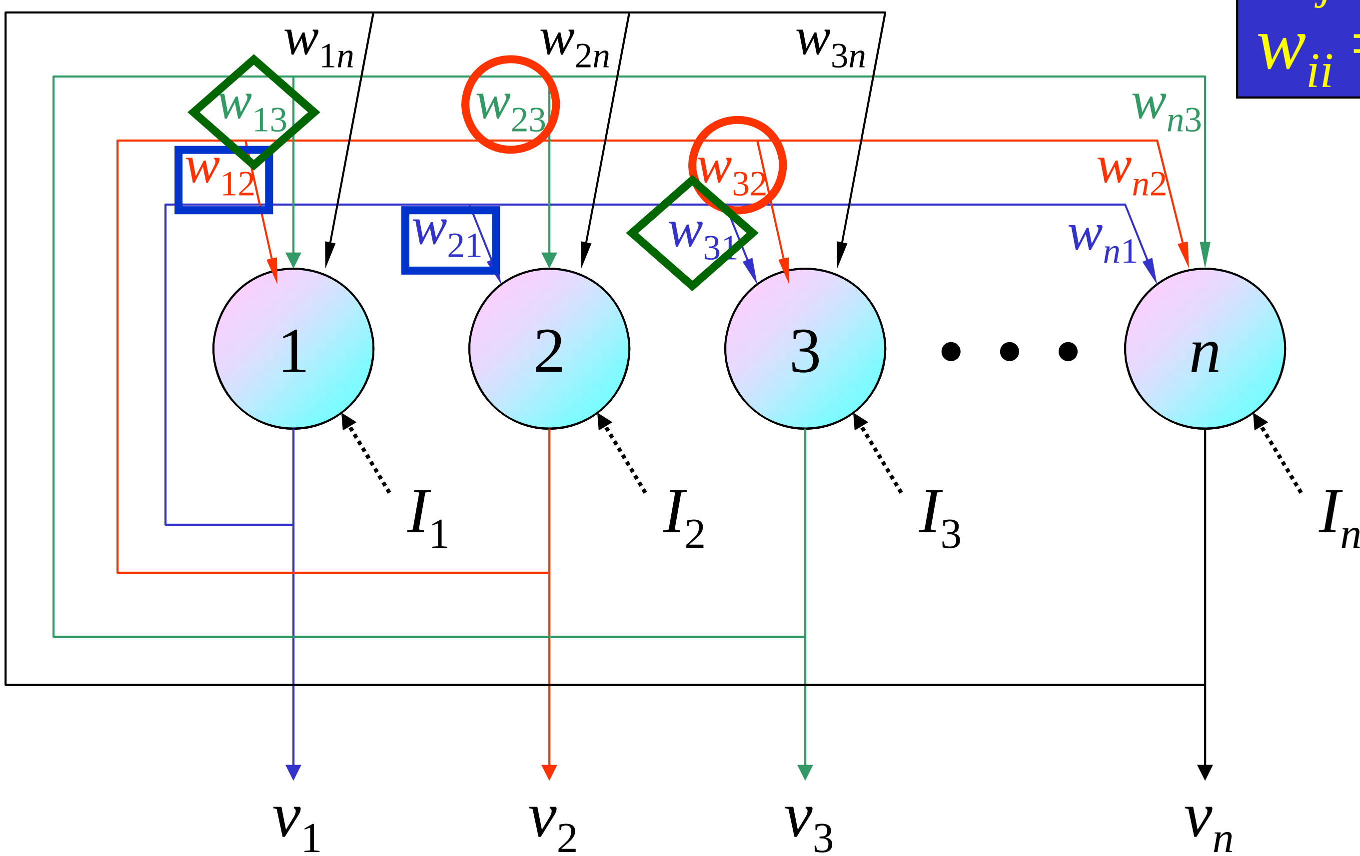
The Discrete Hopfield NNs

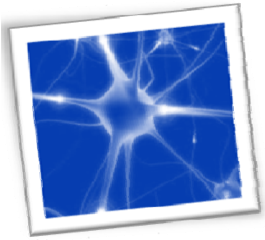
$$w_{ij} = w_{ji}$$
$$w_{ii} = 0$$



The Discrete Hopfield NNs

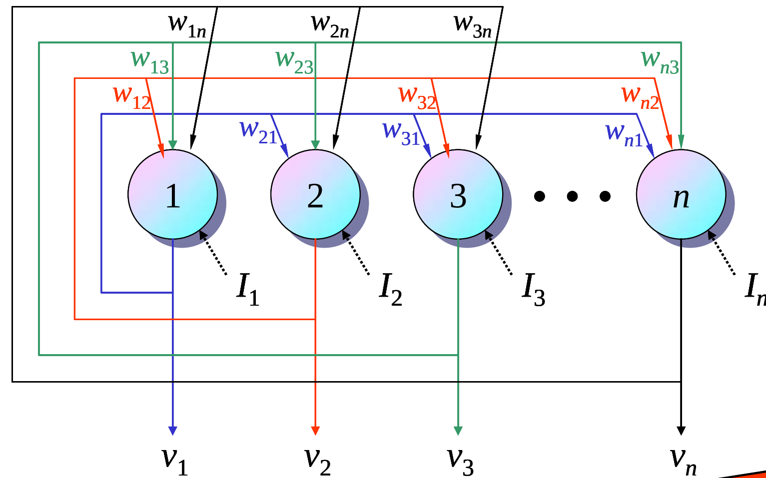
$$w_{ij} = w_{ji}$$
$$w_{ii} = 0$$





State Update Rule

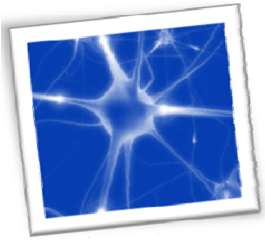
- Asynchronous mode
- Update rule



$$H_i(t+1) = \sum_{\substack{j=1 \\ j\neq i}}^{n} w_{ij} v_j(t) + I_i$$

Stable?

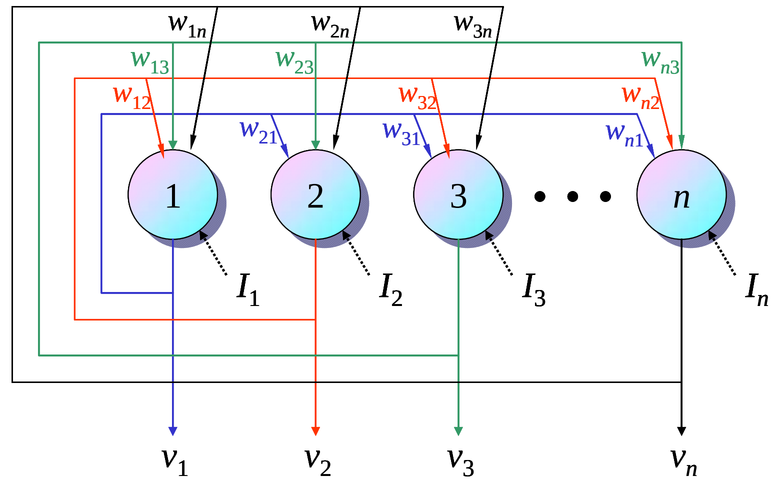
$$v_i(t+1) = \operatorname{sgn}\left[H_i(t+1)\right] = \begin{cases} 1 & H_i(t+1) \geq 0 \\ -1 & H_i(t+1) < 0 \end{cases}$$



Energy Function

$$H_i(t+1) = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j(t) + I_i$$

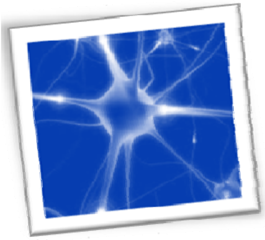
$$v_i(t+1) = \begin{cases} 1 & H_i(t+1) \geq 0 \\ -1 & H_i(t+1) < 0 \end{cases}$$



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

If *E* is *monotonically decreasing*, the system is stable.

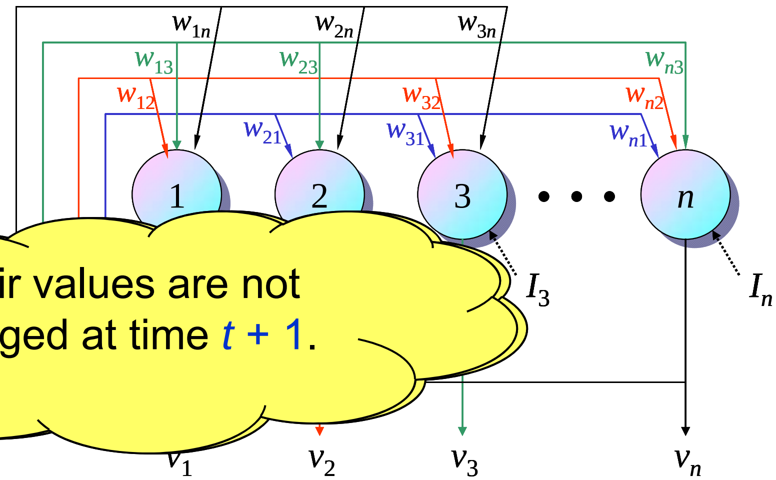
(Due to Lyapunov Theorem)



The Proof

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

Suppose that at time $t + 1$, the k^{th} neuron is selected for update.



Their values are not changed at time $t + 1$.

$$E(t) = -\frac{1}{2}\sum_{\substack{i=1 \\ i\neq k}}^{n}\sum_{\substack{j=1 \\ j\neq k}}^{n} w_{ij} v_i(t) v_j(t) - \sum_{\substack{i=1 \\ i\neq k}}^{n} I_i v_i(t) - \sum_{i=1}^{n} w_{ki} v_k(t) v_i(t) - I_k v_k(t)$$

$$E(t+1) = -\frac{1}{2}\sum_{\substack{i=1 \\ i\neq k}}^{n}\sum_{\substack{j=1 \\ j\neq k}}^{n} w_{ij} v_i(t+1) v_j(t+1) - \sum_{\substack{i=1 \\ i\neq k}}^{n} I_i v_i(t+1) - \sum_{i=1}^{n} w_{ki} v_k(t+1) v_i(t+1) - I_k v_k(t+1)$$

The Proof

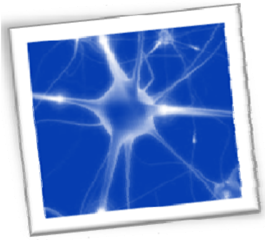
$$\Delta E = E(t+1) - E(t) = -\sum_{i=1}^{n} w_{ki} v_k(t+1) v_i(t) - I_k v_k(t+1) + \sum_{i=1}^{n} w_{ki} v_k(t) v_i(t) + I_k v_k(t)$$

$$= -\left(\sum_{i=1}^{n} w_{ki} v_i(t) + I_k \right) [v_k(t+1) - v_k(t)]$$

$$= -H_k(t+1)[v_k(t+1) - v_k(t)]$$

$$E(t) = -\tfrac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{n} \sum_{\substack{j=1 \\ j \neq k}}^{n} w_{ij} v_i(t) v_j(t) - \sum_{\substack{i=1 \\ i \neq k}}^{n} I_i v_i(t) - \sum_{i=1}^{n} w_{ki} v_k(t) v_i(t) - I_k v_k(t)$$

$$E(t+1) = -\tfrac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^{n} \sum_{\substack{j=1 \\ j \neq k}}^{n} w_{ij} v_i(t+1) v_j(t+1) - \sum_{\substack{i=1 \\ i \neq k}}^{n} I_i v_i(t+1) - \sum_{i=1}^{n} w_{ki} v_k(t+1) v_i(t+1) - I_k v_k(t+1)$$



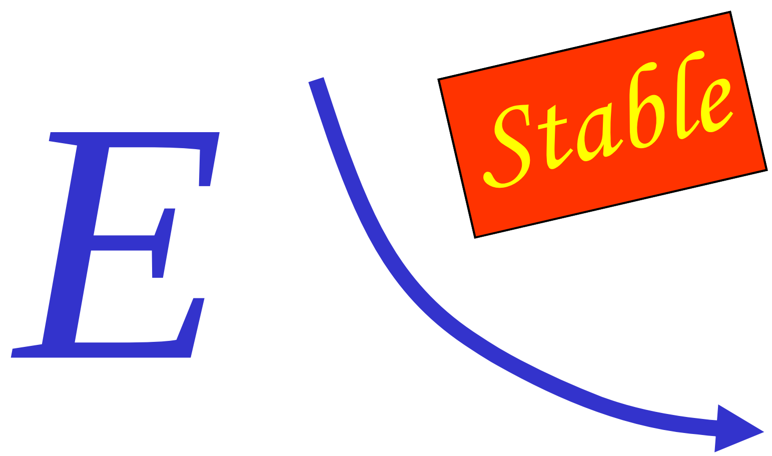
The Proof

$$\Delta E = E(t+1) - E(t) = -\sum_{i=1}^{n} w_{ki} v_k(t+1) v_i(t) - I_k v_k(t+1) + \sum_{i=1}^{n} w_{ki} v_k(t) v_i(t) + I_k v_k(t)$$

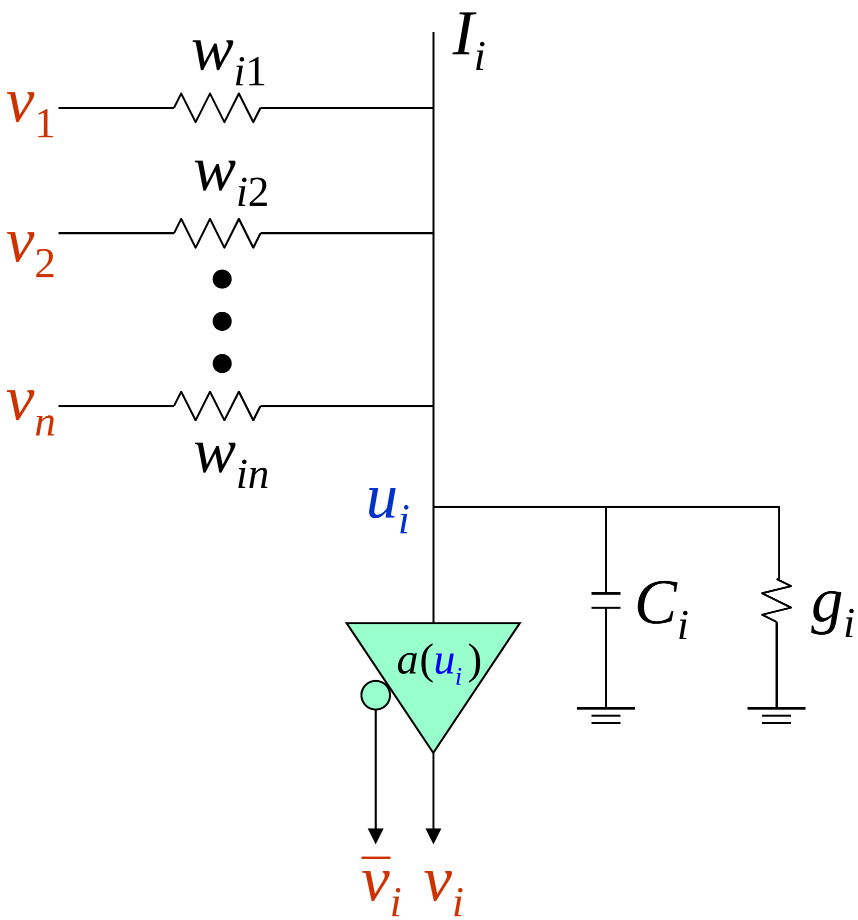
$$= -\left(\sum_{i=1}^{n} w_{ki} v_i(t) + I_k \right) [v_k(t+1) - v_k(t)]$$

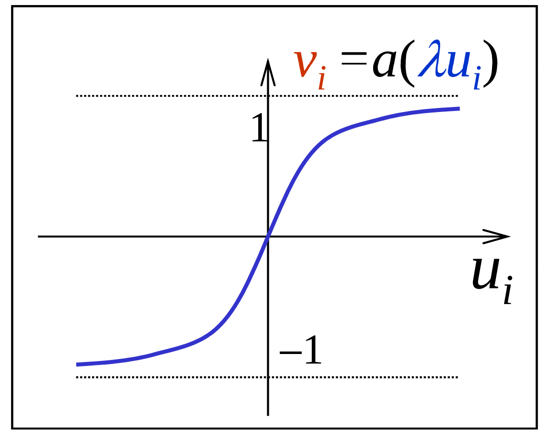
$$= -H_k(t+1)[v_k(t+1) - v_k(t)]$$

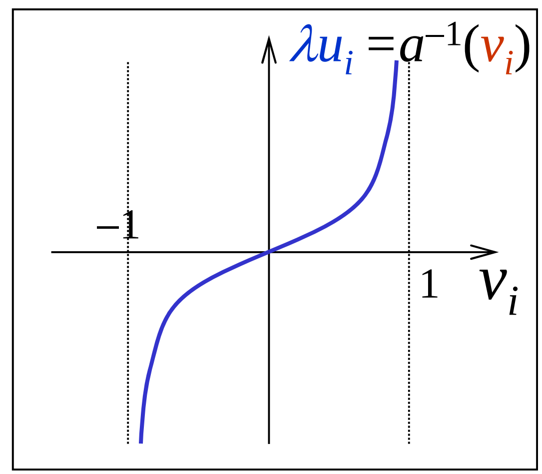
$v_k(t)$	$H_k(t+1)$	$v_k(t+1)$	ΔE
1	≥ 0	1	0
1	< 0	-1	< 0
-1	≥ 0	1	< 0
-1	< 0	-1	0



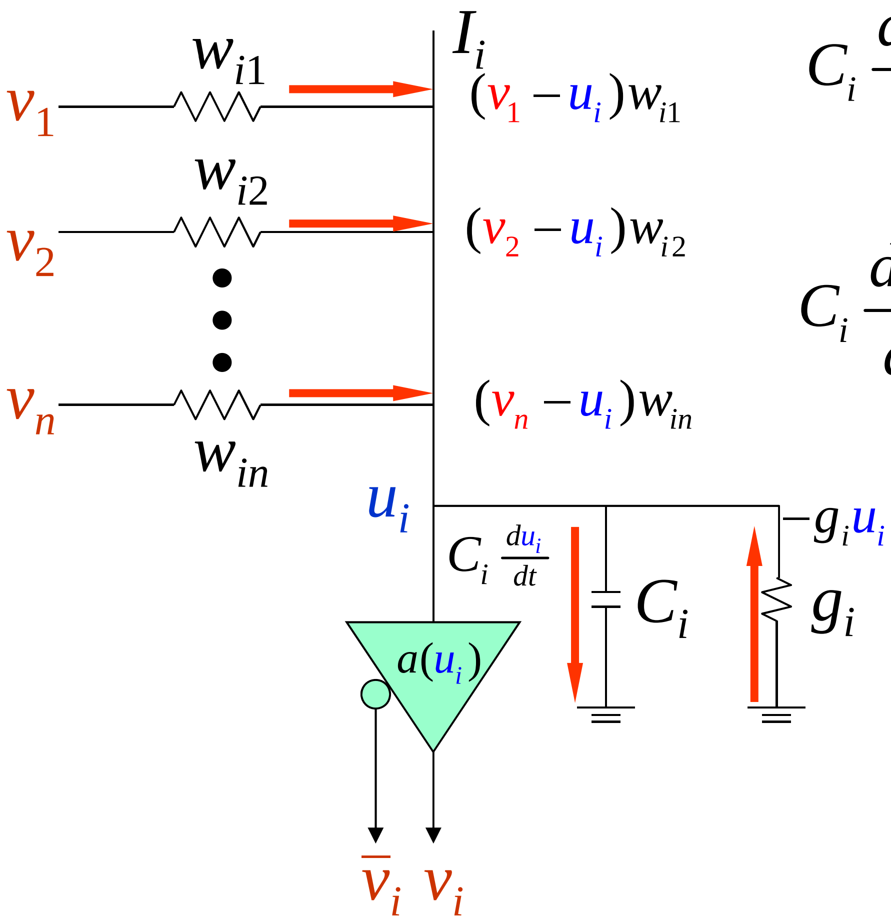
The Neuron of Continuous Hopfield NNs







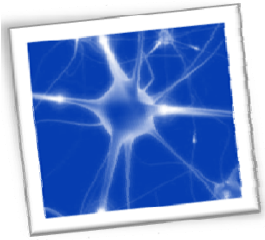
The Dynamics



$$C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij}(v_j - u_i) - g_i u_i + I_i$$

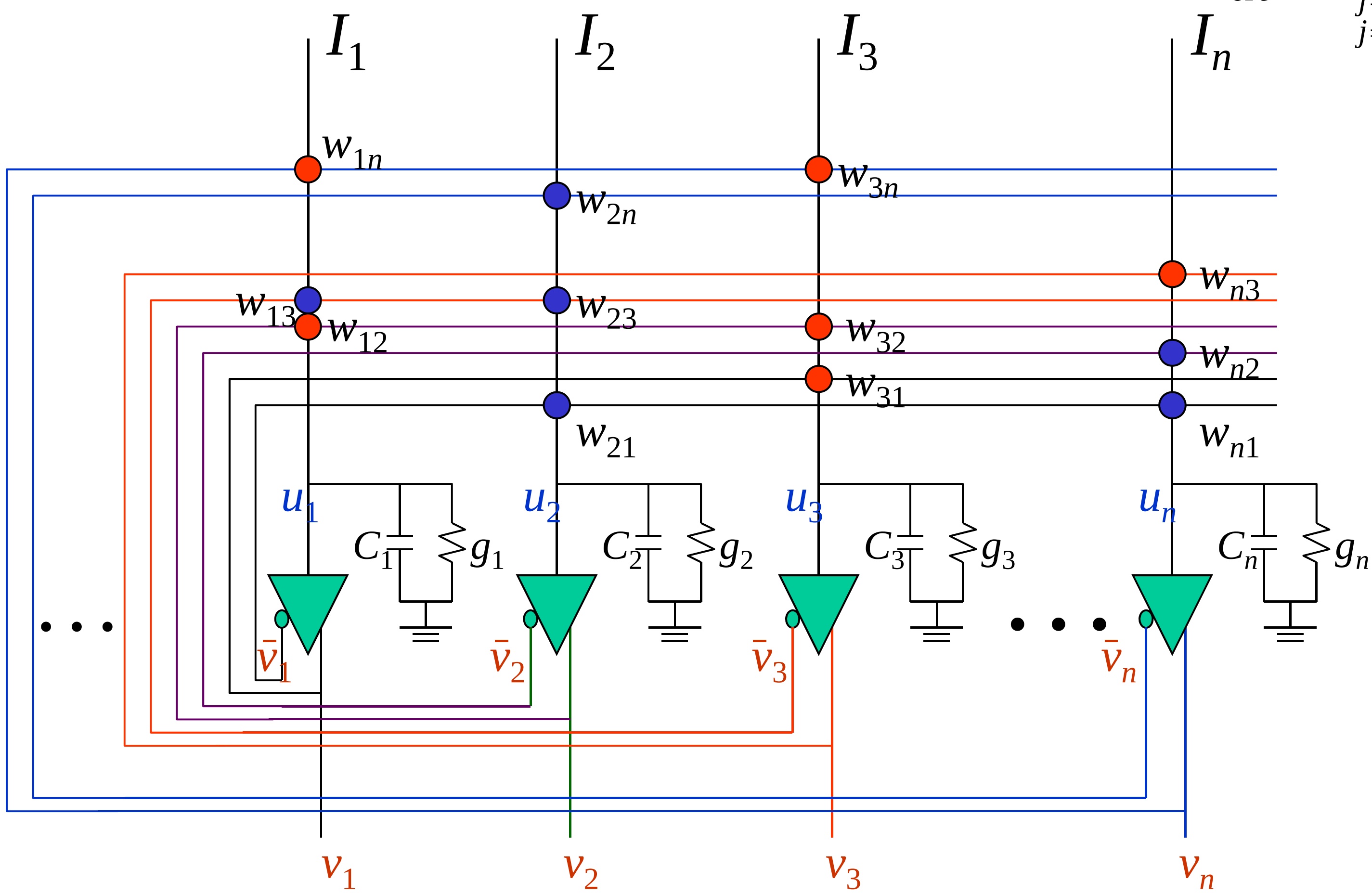
$$C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - \underbrace{\left(\sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} + g_i \right)}_{G_i} u_i + I_i$$

$$C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - G_i u_i + I_i$$



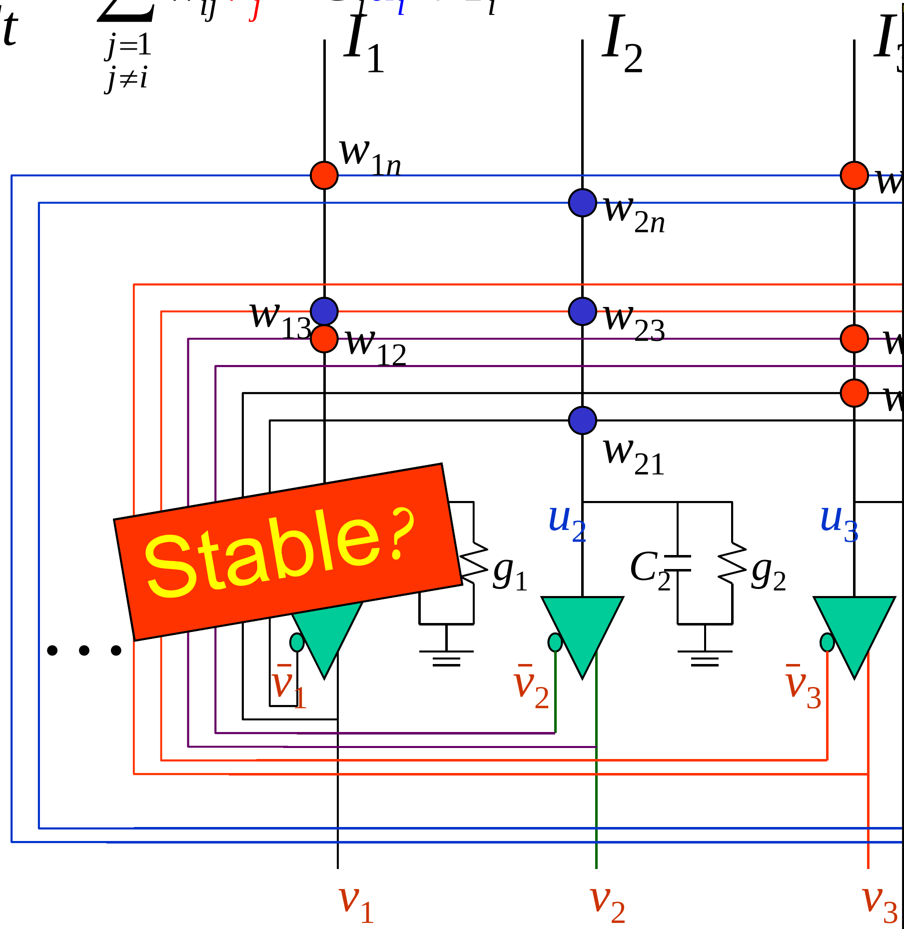
The Continuous Hopfield NNs

$$C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - G_i u_i + I_i$$



The Continuous Hopfield NNs

$$C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - G_i u_i + I_i$$



$$C_1 \frac{du_1}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{1j} v_j - G_1 u_1 + I_1$$

$$C_2 \frac{du_2}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{2j} v_j - G_2 u_2 + I_2$$

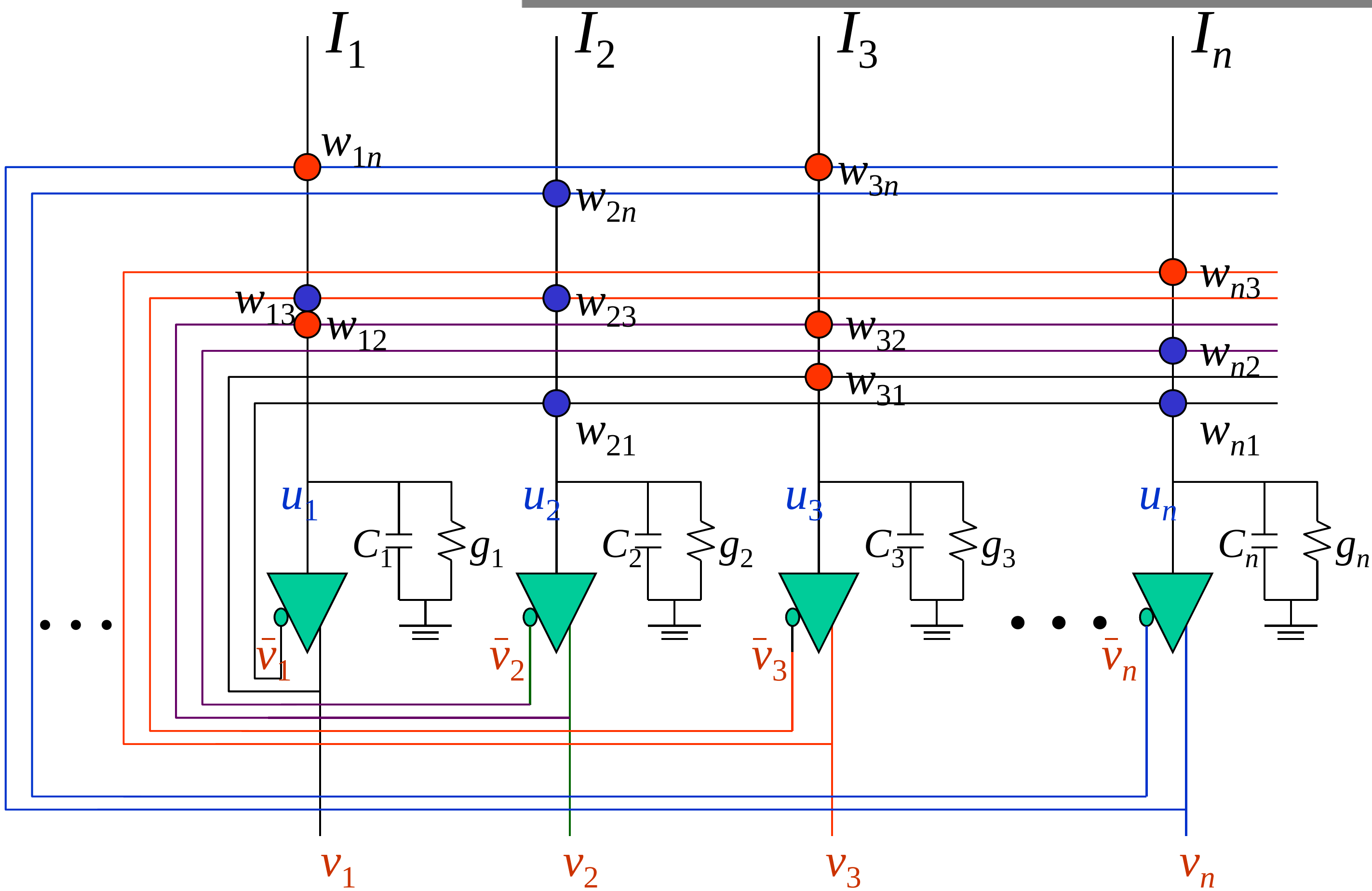
$$C_3 \frac{du_3}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{3j} v_j - G_3 u_3 + I_3$$

$$\vdots$$

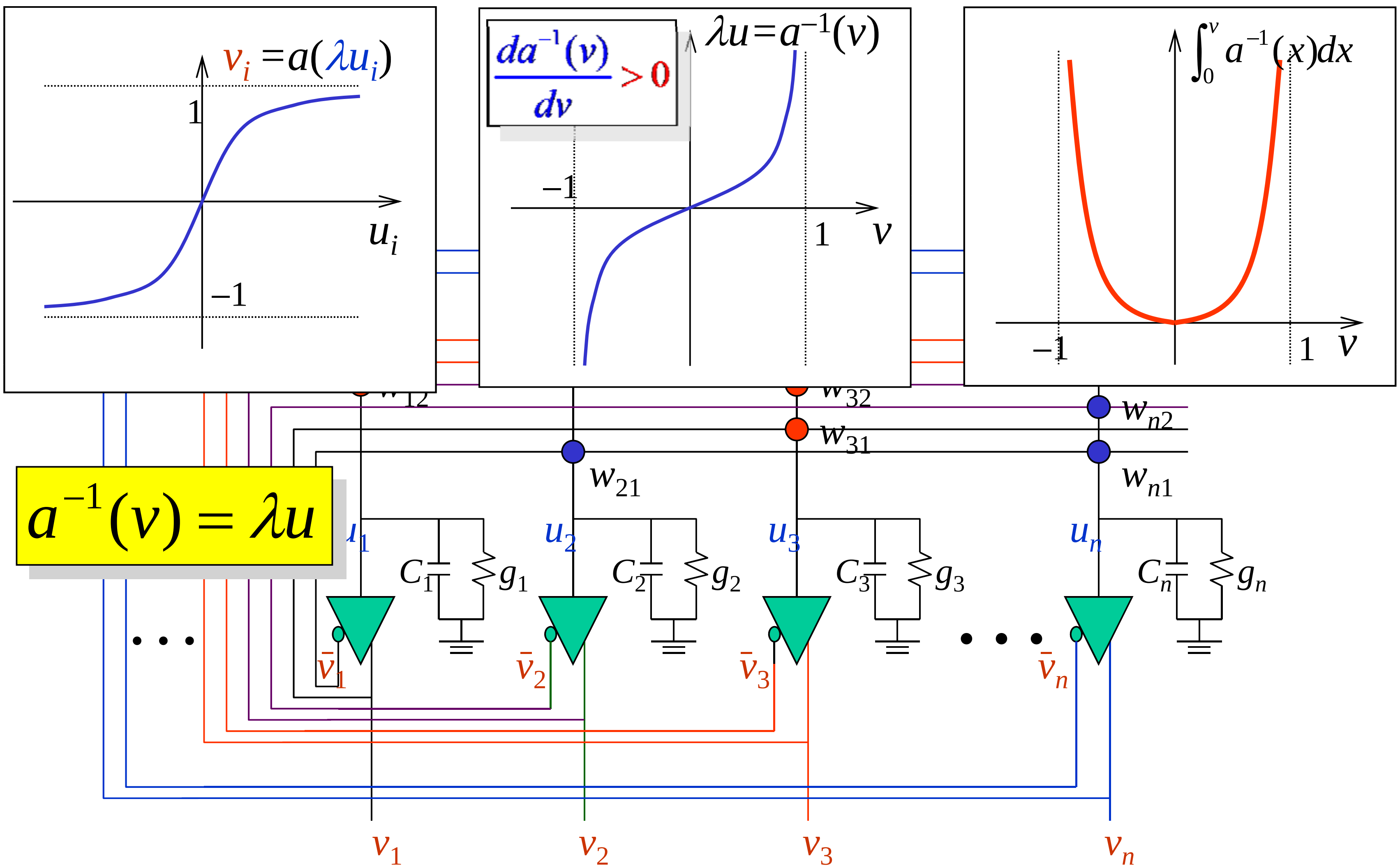
$$C_n \frac{du_n}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{nj} v_j - G_n u_n + I_n$$

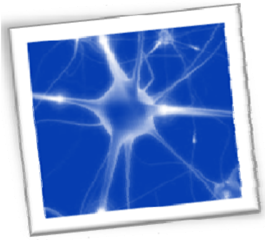
Lyapunov Energy Function

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq n}}^{n} w_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i + \frac{1}{\lambda}\sum_{i=1}^{n} G_i \int_0^{v_i} a^{-1}(v)dv$$



Lyapunov Energy Function





Stability of Continuous Hopfield NNs

Dynamics $C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - G_i u_i + I_i$

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j \neq n}}^{n} w_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i + \frac{1}{\lambda}\sum_{i=1}^{n} G_i \int_0^{v_i} a_i^{-1}(v)\,dv$$

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{dE}{dv_i}\frac{dv_i}{dt} = \sum_{i=1}^{n}\left(-\sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j + G_i u_i - I_i\right)\frac{dv_i}{dt}$$

$$= -\sum_{i=1}^{n} C_i \frac{du_i}{dt}\frac{dv_i}{dt}$$

$$= -\frac{1}{\lambda}\sum_{i=1}^{n} C_i \frac{da_i^{-1}(v_i)}{dv_i}\left(\frac{dv_i}{dt}\right)^2$$

$\lambda u_i = a_i^{-1}(v_i)$

$u_i = \frac{1}{\lambda} a_i^{-1}(v_i)$

$$\frac{du_i}{dt} = \frac{1}{\lambda}\frac{da_i^{-1}(v_i)}{dv_i}\frac{dv_i}{dt}$$

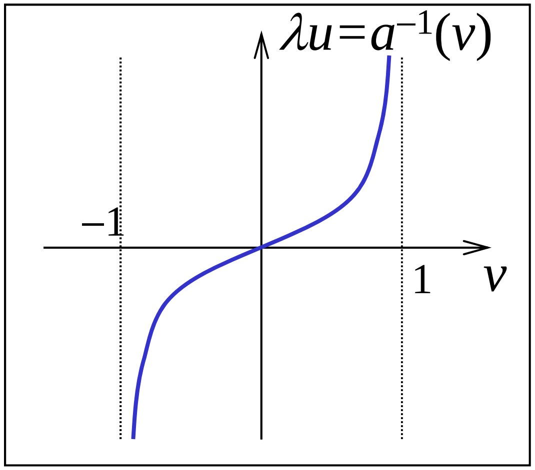
Stability of Continuous Hopfield NNs

Dynamics $C_i \frac{du_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} v_j - G_i u_i + I_i$

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq n}}^{n} w_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i + \frac{1}{\lambda} \sum_{i=1}^{n} G_i \int_0^{v_i} a_i^{-1}(v) dv$$

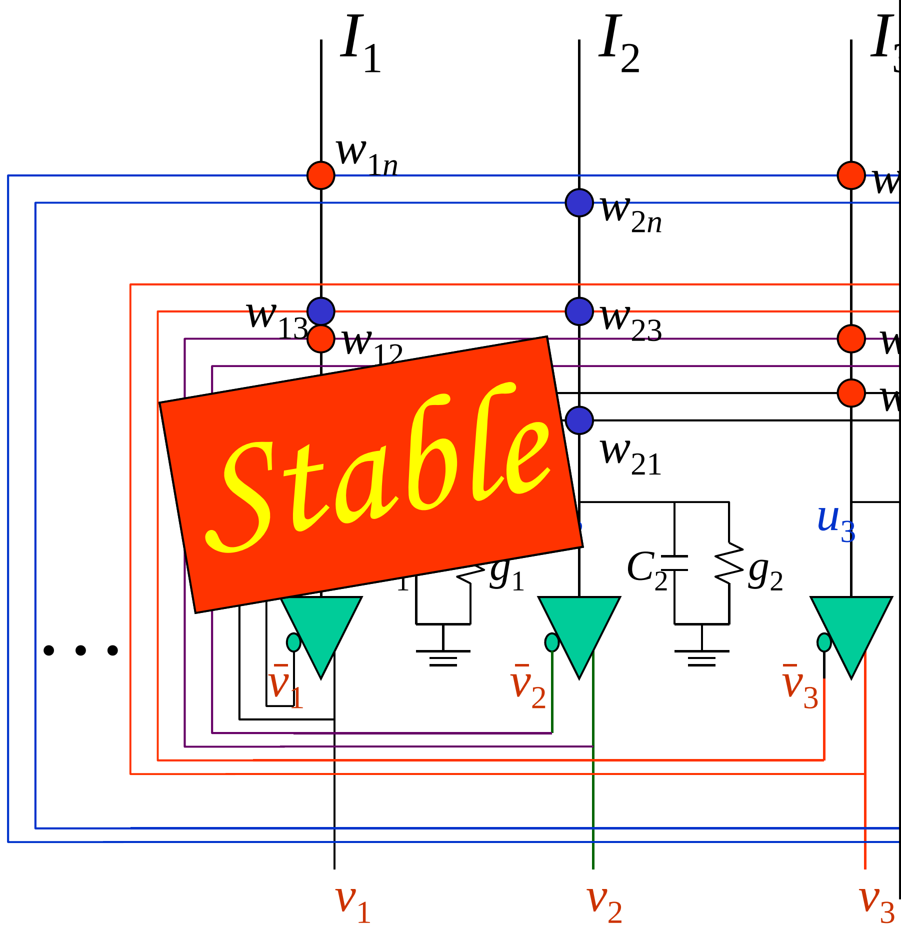
$$\frac{dE}{dt} = -\frac{1}{\lambda} \sum_{i=1}^{n} C_i \underbrace{\frac{da_i^{-1}(v_i)}{dv_i}}_{>0} \left(\frac{dv_i}{dt} \right)^2$$

$$\frac{da^{-1}(v)}{dv} > 0$$



$$\frac{dE}{dt} < 0$$

Stability of Continuous Hopfield NNs



$$C_1 \frac{du_1}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{1j} v_j - G_1 u_1 + I_1$$

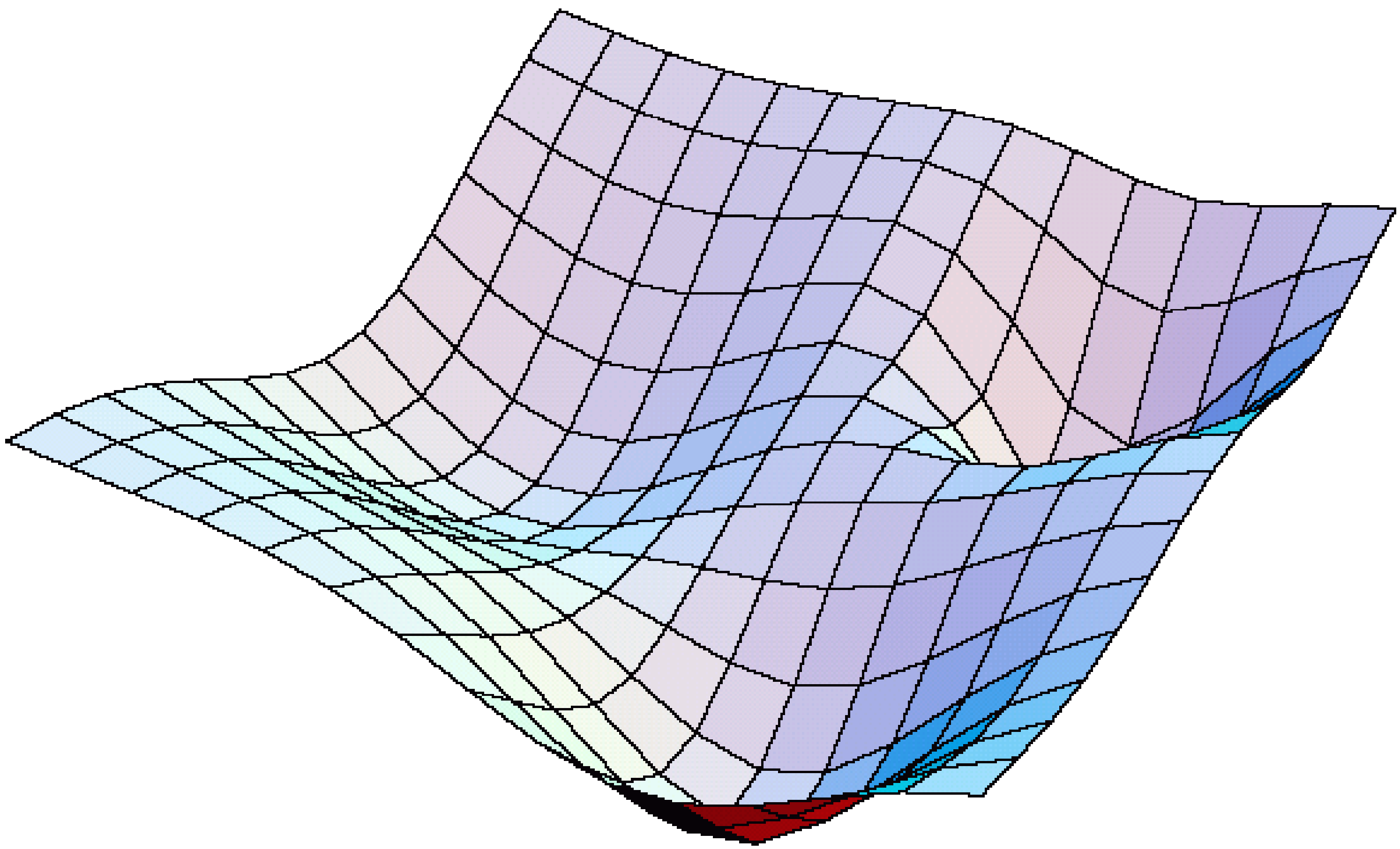
$$C_2 \frac{du_2}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{2j} v_j - G_2 u_2 + I_2$$

$$C_3 \frac{du_3}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{3j} v_j - G_3 u_3 + I_3$$

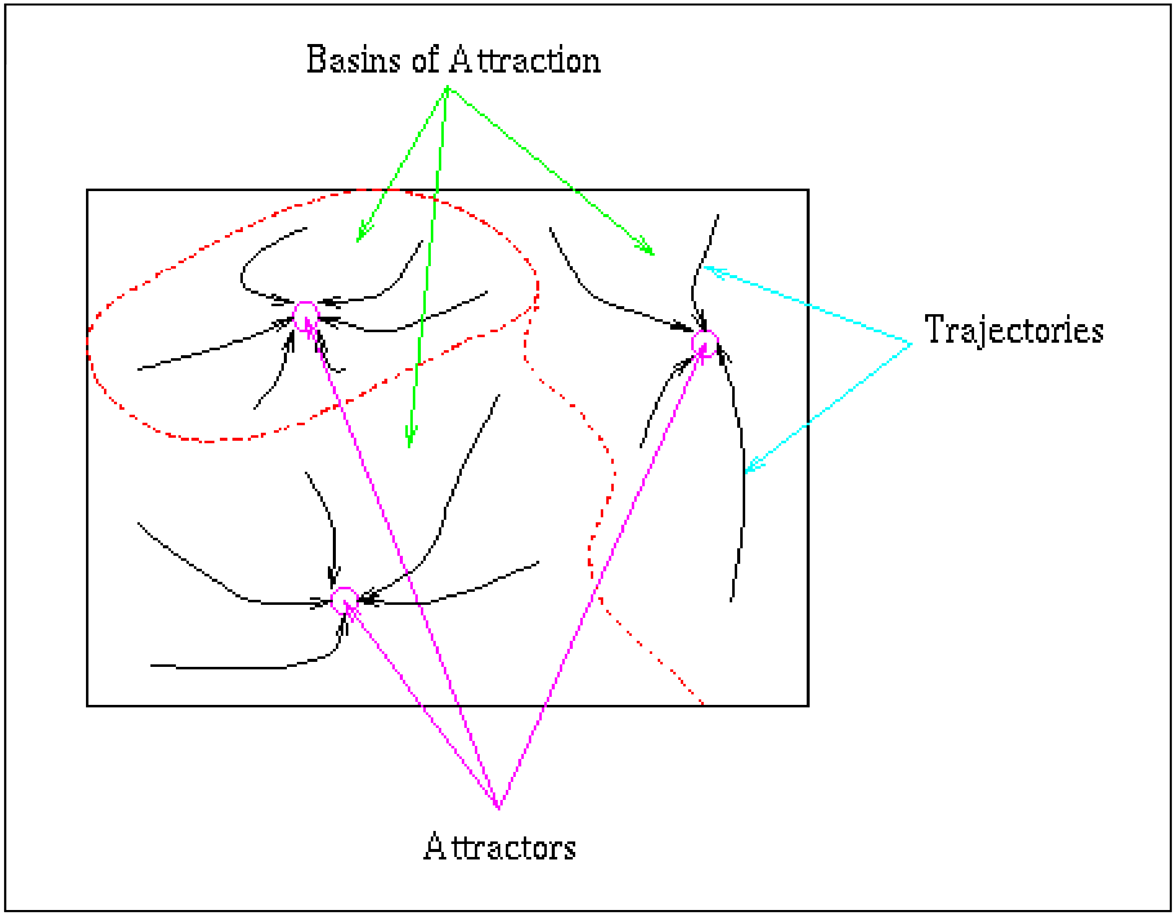
$$\vdots$$

$$C_n \frac{du_n}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{nj} v_j - G_n u_n + I_n$$

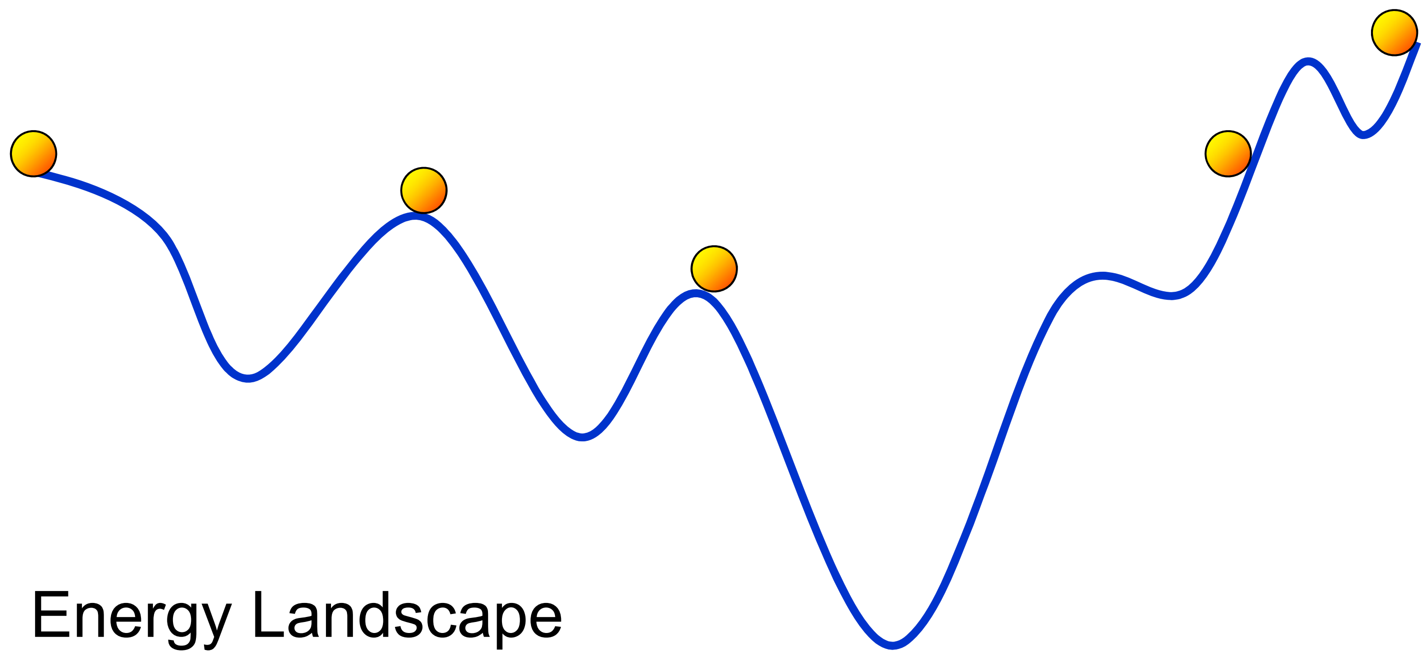
Basins of Attraction

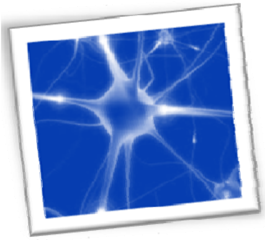


Basins of Attraction



Local/Global Minima





Hopfield Model (Summary)

- The Hopfield network may be operated in a continuous mode or discrete mode
- The continuous mode of operation is based on an additive model
- The discrete mode of operation is based on the McCulloch-Pitts model
- We may establish the relationship between the stable states of the continuous and discrete Hopfield models with the following assumptions:
  - The output of a neuron has the asymptotic values

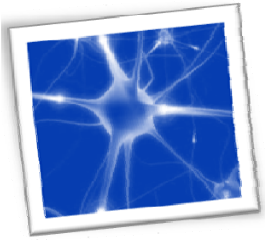
    $x_j = 1$ when $v_j \rightarrow \infty$ and $x_j = -1$ when $v_j \rightarrow -\infty$
  - The midpoint of the activation function of a neuron lies at the origin, that is $\phi(0) = 0$
  - Correspondingly, we may set the bias I_j equal to zero for all j.

Hopfield Model

- If the gain of the activation function is very large (infinity), we can show that the energy function of the formulating the energy function E of the discrete (and continuous) Hopfield models can be rewritten as

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji}x_i x_j (i \neq j)$$

- The models tried to minimize the above energy function



Hopfield Model: Storage

- The learning is similar to Hebbian learning:

$$w_{ji} = \frac{1}{N} \sum_{\mu=1}^{M} \xi_{\mu,j} \xi_{\mu,i}$$

- $w_{ji} = 0$ if i = j. (Learning is **one-shot**.)
- In matrix form the above becomes:

$$\mathbf{W} = \frac{1}{N} \sum_{\mu=1}^{M} \boldsymbol{\xi}_{\mu} \boldsymbol{\xi}_{\mu}^{T} - M\mathbf{I}$$

- The resulting weight matrix W is symmetric:

$$W = W^T$$

Hopfield Model: Activation (Retrieval)

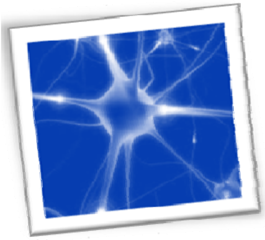
- Initialize the network with a **probe pattern** probe.

$$x_j(0) = \xi_{\text{probe},j}$$

- Update output of each neuron (picking them by random)

$$x_j(n+1) = \text{sgn}\left(\sum_{i=1}^{N} w_{ji} x_i(n)\right)$$

- until x reaches a fixed point.
- The fixed point is the retrieved output.



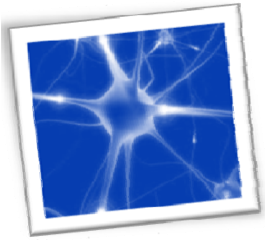
Storage Capacity of Hopfield Network

- Given a probe (unknown input) equal to the stored pattern, the activation of the j^th neuron can be decomposed into the signal term and the noise term (The unit hypercube is N-dimensional and we have M fundamental memory):

$$
\begin{aligned}
v_j &= \sum_{i=1}^{N} w_{ji}\xi_{v,i} \\
&= \frac{1}{N}\sum_{\mu=1}^{M}\xi_{\mu,j}\sum_{i=1}^{N}\xi_{\mu,i}\xi_{\nu,i} \\
&= \underbrace{\xi_{\nu,j}}_{signal} + \underbrace{\frac{1}{N}\sum_{\mu=1,\mu\neq\nu}^{M}\xi_{\mu,j}\sum_{i=1}^{N}\xi_{\mu,i}\xi_{\nu,i}}_{noise}
\end{aligned}
$$

Storage Capacity of Hopfield Network

- The signal to noise ratio is obtained as b = N/M
- It has been shown that memory recall of the Hopfield network deteriorates with increasing load parameter b, and breaks down at the critical value $b_c = 0.14$
- Thus, $M < 0.15N$ in order that Hopfield retrieve correctly



Cohen-Grossberg Theorem

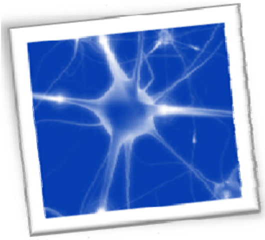
- Cohen & Grossberg defined a general principle for assessing the stability of a certain class of neural networks described by the following equations (Hopfiled model is a special case):

$$du_j(t)/dt = a_j(u_j)[b_j(u_j) - \Sigma_{i=1}^{N} c_{ji}\, \phi_i\, (u_i)\,] \quad ; \quad j = 1,\ldots,N$$

- They proved that the energy function E described below can be considered as a Lyapunov function for the above system

$$E = (1/2)\, \Sigma_{i=1}^{N}\, \Sigma_{j=1}^{N}\, c_{ji}\, \phi_i\, (u_i)\, \phi_j\, (u_j)$$
$$- \Sigma_{j=1}^{N} \int_{o}^{uj} b_j(k)\phi'_j(k)dk$$

$$\phi'_j(k) = (d/dk)\phi_j(k)$$



Associative Memories

- Also named content-addressable memory.
- Autoassociative Memory (Hopfield Memory)
- Heteroassociative Memory (Bidirection Associative Memory)

Stored Patterns

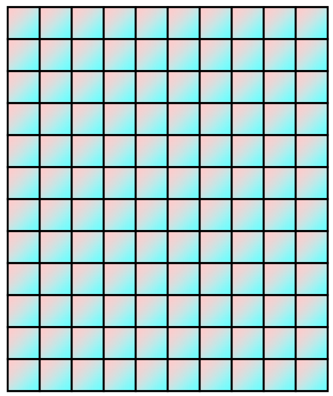
$$(\mathbf{x}^1, \mathbf{y}^1)$$
$$(\mathbf{x}^2, \mathbf{y}^2)$$
$$\vdots$$
$$(\mathbf{x}^p, \mathbf{y}^p)$$

$$\mathbf{x}^i \in R^n$$
$$\mathbf{y}^i \in R^m$$

$\mathbf{x}^i \equiv \mathbf{y}^i$ Autoassociative

$\mathbf{x}^i \neq \mathbf{y}^i$ Heteroassociative

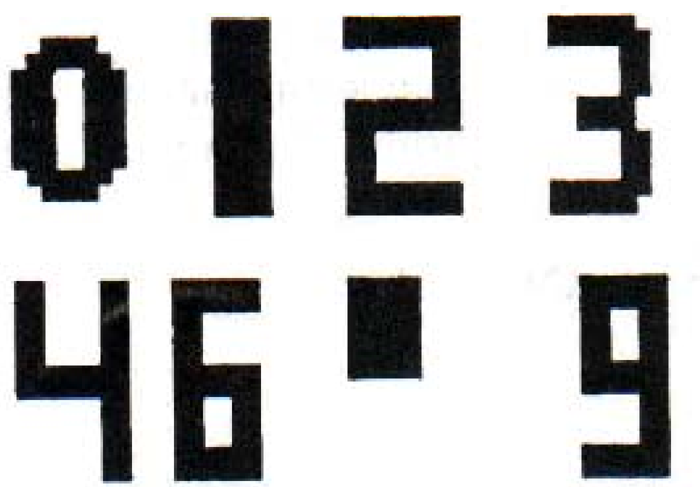
Hopfield Memory



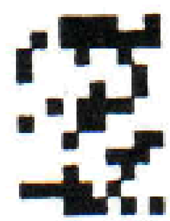
12×10 Neurons

Fully connected

14,400 weights

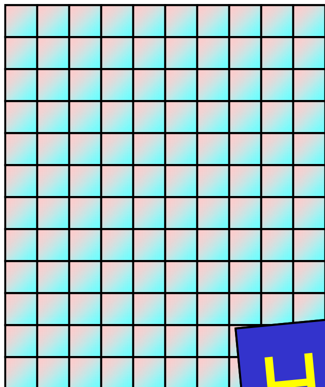


Stored Patterns



Memory Association

Hopfield Memory



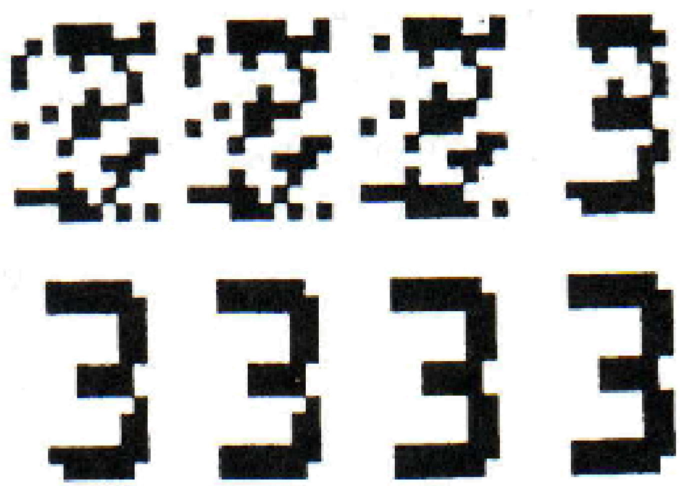
12×10 Neurons

Fully connected

14,400 weights

How to Store Patterns?

Stored Patterns



Memory Association

The Storage Algorithm

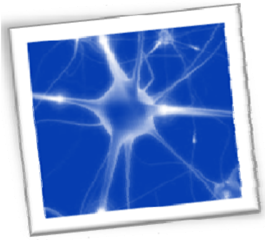
Suppose that the set of stored patterns is of dimension *n*.

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} = ?$$

$$\mathbf{x}^k = (x_1^k, x_2^k, \ldots, x_n^k)^T \qquad k = 1, 2, \ldots p.$$

$$x_i^k \in \{+1, -1\} \quad i = 1, 2, \ldots n.$$

$$\mathbf{W} = \sum_{k=1}^{p} \mathbf{x}^k (\mathbf{x}^k)^T - p\mathbf{I} \quad \Rightarrow \quad w_{ij} = \begin{cases} \sum_{k=1}^{p} x_i^k x_j^k & i \neq j \\ 0 & i = j \end{cases}$$



Analysis

$$\mathbf{W} = \sum_{k=1}^{p} \mathbf{x}^k (\mathbf{x}^k)^T - p\mathbf{I} \qquad w_{ij} = \begin{cases} \sum_{k=1}^{p} x_i^k x_j^k & i \neq j \\ 0 & i = j \end{cases}$$

Suppose that $\mathbf{x} \approx \mathbf{x}^i$.

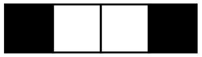
$$\mathbf{W}\mathbf{x} = \left[\sum_{k=1}^{p} \mathbf{x}^k (\mathbf{x}^k)^T - p\mathbf{I} \right] \mathbf{x}$$

$$\approx n\mathbf{x}^i - p\mathbf{x}^i = (n-p)\mathbf{x}^i$$

Example

$$\mathbf{W} = \sum_{k=1}^{p} \mathbf{x}^k (\mathbf{x}^k)^T - p\mathbf{I} \qquad w_{ij} = \begin{cases} \sum_{k=1}^{p} x_i^k x_j^k & i \neq j \\ 0 & i = j \end{cases}$$

$$\mathbf{x}^1 = (1, -1, -1, 1)^T$$



$$\mathbf{x}^2 = (-1, 1, -1, 1)^T$$

$$\mathbf{x}^1(\mathbf{x}^1)^T = \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

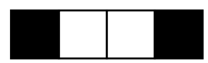
$$\mathbf{x}^2(\mathbf{x}^2)^T = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

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

$$\mathbf{x}^1(\mathbf{x}^1)^T + \mathbf{x}^2(\mathbf{x}^2)^T = \begin{bmatrix} 2 & -2 & 0 & 0 \\ -2 & 2 & 0 & 0 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & -2 & 2 \end{bmatrix}$$

Example

$$\mathbf{x}^1 = (1,-1,-1,1)^T$$



$$\mathbf{x}^2 = (-1,1,-1,1)^T$$

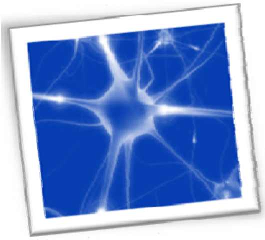
$$\mathbf{W} = \begin{bmatrix} 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \end{bmatrix}$$

$$E(\mathbf{x}) = -\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j - \sum_{i=1}^{n} I_i x_i$$

$$= -\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j$$

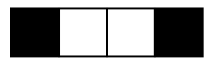
$$= -\tfrac{1}{2}\mathbf{x}^T \mathbf{W} \mathbf{x}$$

$$= 2(x_1 x_2 + x_3 x_4)$$



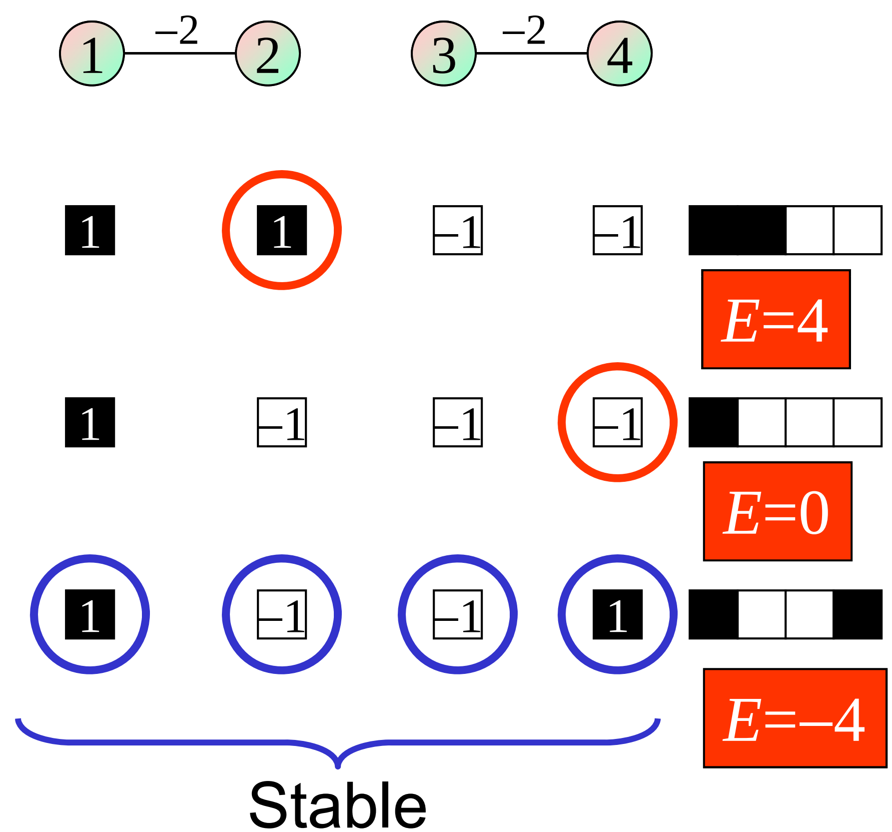
Example

$$\mathbf{x}^1 = (1, -1, -1, 1)^T$$



$$\mathbf{x}^2 = (-1, 1, -1, 1)^T$$

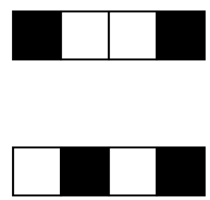
$$E(\mathbf{x}) = 2(x_1 x_2 + x_3 x_4)$$



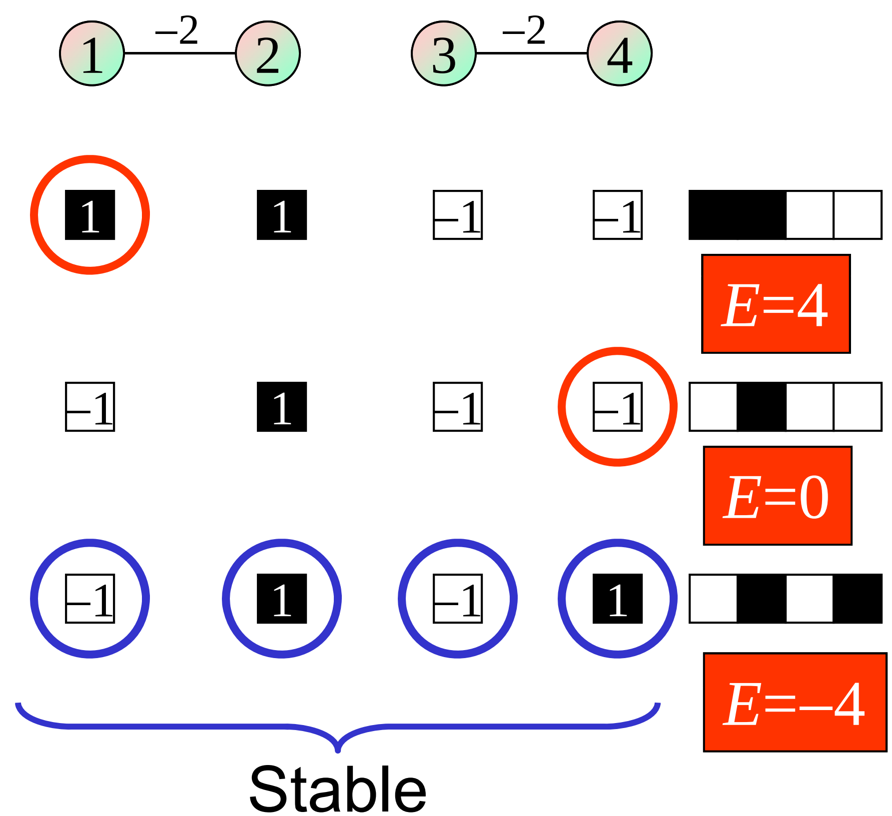
Example

$$\mathbf{x}^1 = (1, -1, -1, 1)^T$$

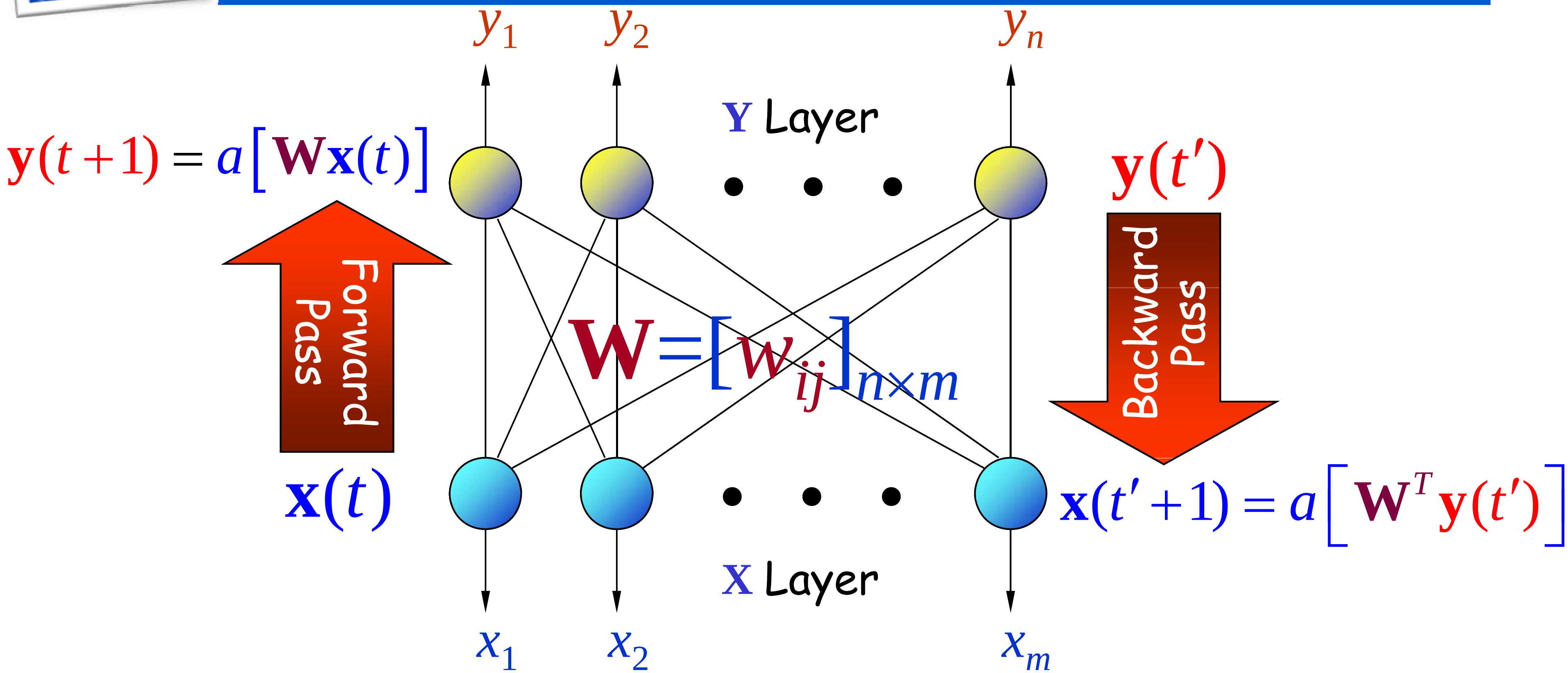
$$\mathbf{x}^2 = (-1, 1, -1, 1)^T$$



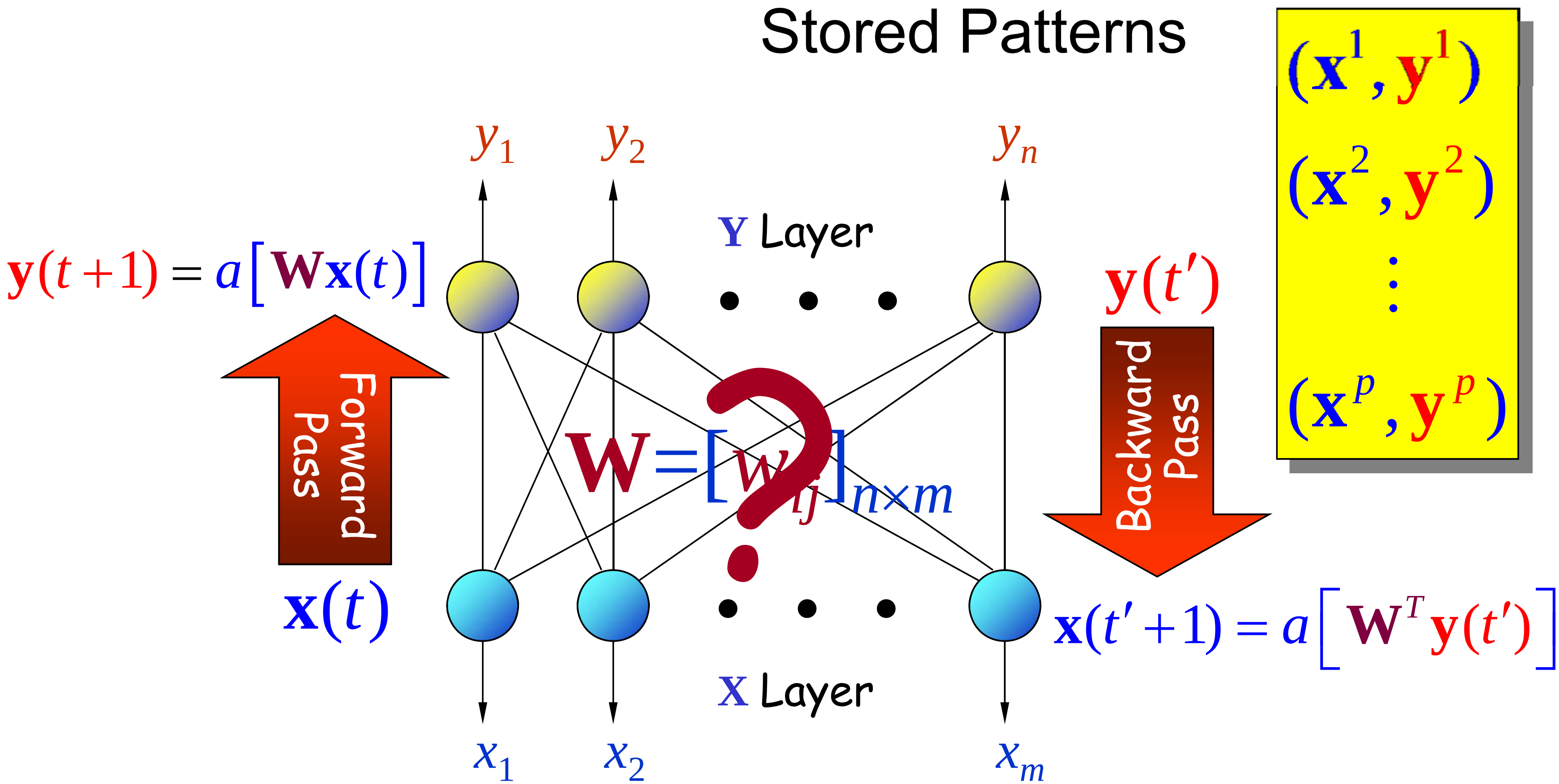
$$E(\mathbf{x}) = 2(x_1 x_2 + x_3 x_4)$$

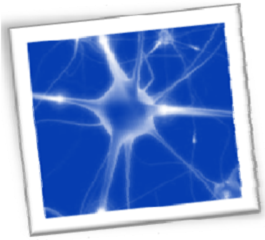


Bidirection Memory



Bidirection Memory





The Storage Algorithm

Stored Patterns

$$\mathbf{x}^k = (x_1, x_2, \ldots, x_m)^T \qquad x_i \in \{-1, 1\}$$

$$\mathbf{y}^k = (y_1, y_2, \ldots, y_n)^T \qquad y_i \in \{-1, 1\}$$

$$(\mathbf{x}^1, \mathbf{y}^1)$$

$$(\mathbf{x}^2, \mathbf{y}^2)$$

$$\vdots$$

$$(\mathbf{x}^p, \mathbf{y}^p)$$

$$\mathbf{W} = \sum_{k=1}^{p} \mathbf{y}^k (\mathbf{x}^k)^T$$

$$w_{ij} = \sum_{k=1}^{p} y_i^k x_j^k$$

Analysis

$$\mathbf{W} = \sum_{k=1}^{p} \mathbf{y}^k (\mathbf{x}^k)^T$$

Suppose $\mathbf{x}^{k'}$ is one of the stored vector:

$$\mathbf{y} = a\left(\mathbf{W}\mathbf{x}^{k'}\right) = a\left(\sum_{k=1}^{p} \mathbf{y}^k (\mathbf{x}^k)^T \mathbf{x}^{k'}\right)$$

$$= a\left(m\mathbf{y}^{k'} + \underbrace{\sum_{\substack{k=1 \\ k \neq k'}}^{p} \mathbf{y}^k (\mathbf{x}^k)^T \mathbf{x}^{k'}}_{\approx 0} \right) \approx a\left(m\mathbf{y}^{k'}\right) = \mathbf{y}^{k'}$$

Energy Function:

$$E(\mathbf{x}, \mathbf{y}) = -\tfrac{1}{2}\mathbf{x}^T \mathbf{W}^T \mathbf{y} - \tfrac{1}{2}\mathbf{y}^T \mathbf{W}\mathbf{x} = -\mathbf{y}^T \mathbf{W}\mathbf{x}$$

Applications of Hopfield Memory

- Pattern restoration
- Pattern completion
- Pattern generalization
- Pattern association

Reading

- S Haykin, Neural Networks: A Comprehensive Foundation, 2007 (Chapter 14).