Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program

605-447/625-438

Dr. Mark Fleischer

Module 8.1: Hebbian Learning

This Sub-Module Covers …

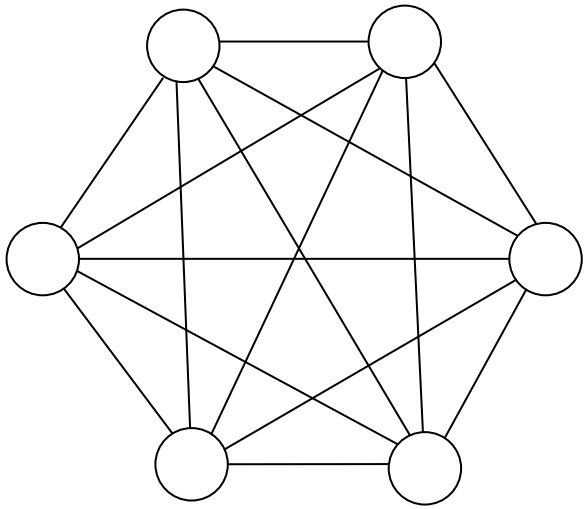
- The concept of Hebbian Learning.
- How to implement Hebbian learning paradigms with recurrent neural networks.
- How to model recurrent networks using matrix methods.

Dynamical Systems

- Neural Networks can be fashioned into dynamical systems.
- Feeding outputs back as inputs and cycling them ala fixed point exercise.
- How does such a system behave?
- Let's examine the most basic formulation.

Associative Neural Networks

- Examine the Hebbian Learning paradigm.
- Examine how and why the Hopfield Network works.



For n nodes, there are $\frac{n(n-1)}{2}$ connections which can be associated as the weights between each node.

With the addition of edges from each node to itself, and the symmetry of the nodes, we get n^2 weights.

Hebbian Learning

- In this type of net, and the FFBP algorithm, once the weights are established they are treated as fixed for the given input/output pairs that are used to train the net.
- Experiments with physiological neurons however reveal that the post-synaptic potential varies with time (as I mentioned during our first class).
- Moreover, the "weights" are also affected temporally by other efficacies for the same node. Some excite, others inhibit and if inhibitory potentials (negative weights) have "fired" in close temporal proximity to another excitatory potential, both may then have modified efficacies closer to their average (zero for example).
- Long-term potentiation of synapses, long-term depression of synapses occur if the potentials remain increased or decreased for extended periods of time.
- Thus, repeated temporal firings over time can lead to a more permanent modification of weights while short-term modifications do not last as long.

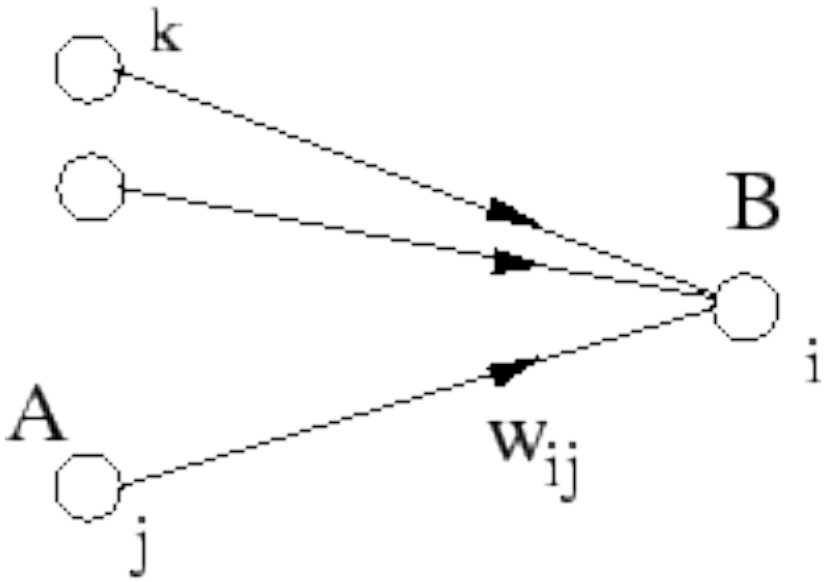
Hebbian Learning

- “Synaptic changes that are driven by correlated activity of pre- and post-synaptic neurons. This class of learning rule can be motivated by Hebb's principle and is therefore often called `Hebbian learning'.”
- This is often described in terms of *synaptic plasticity*.

Hebbian Learning

- When an axon of cell *A* is near enough to excite cell *B* or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that *A*'s synaptic efficacy, as one of the cells firing *B*, is increased.

Hebbian Learning



The change at synapse w_{ij} depends on the state of the presynaptic neuron *j* and the postsynaptic neuron *i* and the present efficacy w_{ij}.

Hebbian Learning

- Correlation-based learning is now generally called *Hebbian learning*. Correlations with firings on either side of the synapse are the basis of this type of learning which helps to stabilize the behavior of the neuron. Leads to the "cascade of associations" I mentioned in the first class.
- Long-lasting changes of synaptic efficacies were found experimentally by many researchers.
- How can we model this type of behavior?
- There are two aspects in Hebb's postulate that are particularly important, viz. ***locality*** and ***cooperativity***.
  - Locality means that the change of the synaptic efficacy can only depend on local variables, i.e., on information that is available at the site of the synapse, such as pre- and postsynaptic firing rate, and the actual value of the synaptic efficacy, but not on the activity of other neurons. Based on the locality of Hebbian plasticity we can make a rather general ansatz for the change of the synaptic efficacy,

Hebbian Idea

$$\frac{dw_{ij}}{dt} = f(w_{ij}, v_i, v_j)$$

The main idea of Hebbian Learning is that the behavior of other nodes can affect the synaptic weights of a given node in a manner that is either **reinforcing** or **inhibiting**.

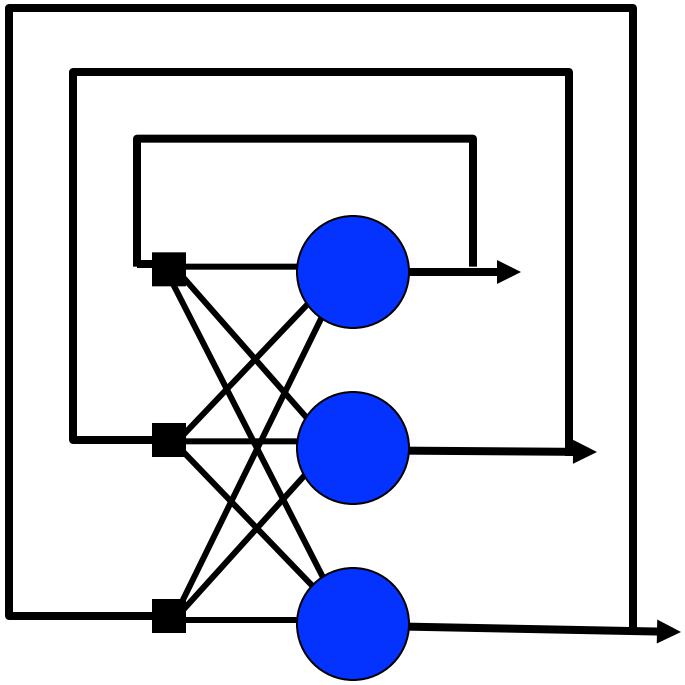
Network Structure

Let's start with simple ideas:

- One node for every input element of an exemplar.
- Two arcs between any two nodes (symmetry).
- No arc from a node to itself.
- Should map an exemplar to itself.

The Recurrent Network Topology

Let's view the network a bit differently.



Exemplar:

(1, -1, -1)

What is the weight from Node 1 to Node 2?

Network to Matrix

Each output x_i for a node i in a network of N perceptrons can be written as:

$$x_i = \sum_{j=1}^{N} w_{ij} x_j$$

But this is just the definition of the i^{th} element of a vector after a matrix/vector multiplication!

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k$$

where index k corresponds to the iterate number.

Network to Matrix

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k$$

But how should we define **W**?

Sometimes in order to answer a question relating to a problem, we must change the problem!

$$\mathbf{p} = \mathbf{W}\mathbf{p}$$

Instead of deciphering a general dynamical system, let's examine a special case --- a fixed point!

Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447/625-438
Dr. Mark Fleischer

Module 8.2: Hebbian Learning Continued

This Sub-Module Covers …

- Performance evaluation.
- Proper training does not merely involve minimizing the error function using the training data.
- Must have some appropriate way of gauging how well the network performs using data it hasn't seen during the training phase.

Network to Matrix

Each output x_i for a node i with N perceptrons can be written as:

$$x_i = \sum_{j=1}^{N} w_{ij} x_j$$

But this is just the definition of the i^{th} element of a vector after a matrix/vector multiplication!

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k$$

where index k corresponds to the iterate number.

Network to Matrix

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k$$

But how should we define **W**?

Sometimes in order to answer a question relating to a problem, we must change the problem!

$$\mathbf{p} = \mathbf{W}\mathbf{p}$$

Instead of deciphering a general dynamical system, let's examine a special case --- a fixed point!

A Test Exemplar

Given some input pattern (or exemplar)

$$p = \left\{ \begin{array}{ccc} \oplus & \cdot & \cdot \\ \cdot & \oplus & \cdot \\ \cdot & \cdot & \oplus \end{array} \right\},$$

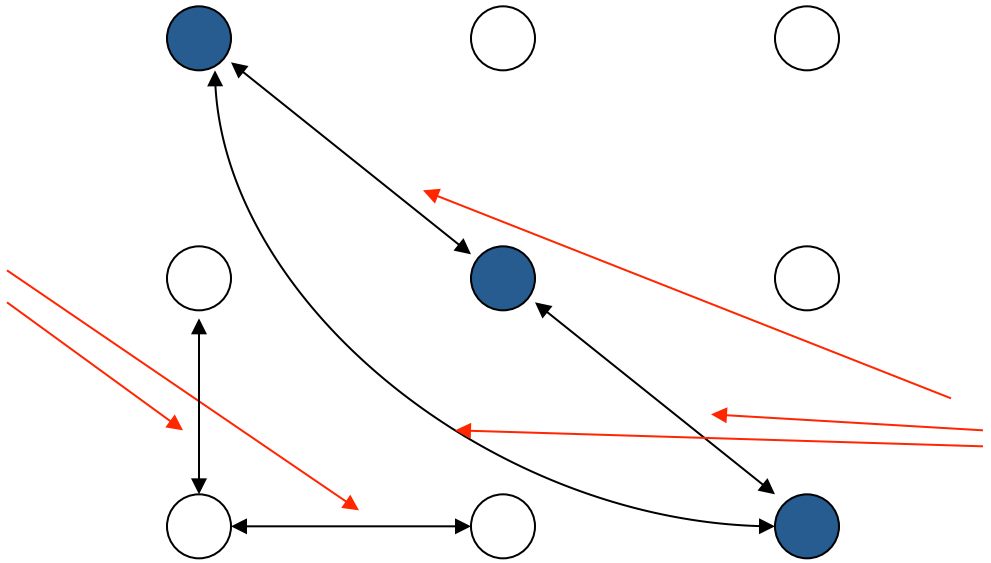
where $\oplus$ = On and $\cdot$ = Off

We want to design a network that will converge to a set of exemplars given a partial or noisy representation of the exemplars.

Mutual Reinforcement ala Hebbian Learning

The ● represent a 1 input, the ○ represent a 0 input.

Initially, all weights are 0.



These weights stay 0.

These weights are increased by 1.

Note: This doesn't show all the edges (weights).

How can we model this mathematically?

Change the input pattern into a vector of 1's and 0's.

$$\mathbf{p}^{\mathbf{T}} = \{\oplus\cdot\cdot|\cdot\oplus\cdot|\cdot\cdot\oplus\}$$

$$= \begin{pmatrix} 100 & 010 & 001 \end{pmatrix}$$

Now we compute a matrix **W** such that **Wp** = **p**.

How should such a matrix be defined?

W = ?

W = **I**?

Trivial and generic!

Hint: Must be somehow linked to the vector **p** to facilitate the calculation **Wp** = **p.**

Can we create a matrix from the vector **p**?

$$\mathbf{p}\mathbf{p}^{\mathrm{T}} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

We will need to get rid of the 1's on the diagonal. Can't have a node reinforce itself. Why?

We will create/introduce a function Z which operates on matrices and sets diagonal elements to 0.

So let $\Delta \mathbf{W} = Z\left(\mathbf{p}\mathbf{p}^{\mathbf{T}}\right)$ and

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + Z\left(\mathbf{p}\mathbf{p}^{\mathbf{T}}\right)$$

We will also use a hard-limiting activation function: $F_h(x) = \begin{cases} 1, \text{if } x > 0 \\ 0, \text{if } x \leq 0 \end{cases}$

So does $\mathbf{W}_{\text{new}}\mathbf{p} = \mathbf{p}$?

What is the Z function?

- For any 1, 0 vector **p** producing $\mathbf{pp}^T$ we want to subtract a modified identity matrix to get rid of entries on the diagonal with a 1.

E.g., p = (1 0 1) then pp^T =

$$\begin{matrix}1&0&1\\0&0&0\\1&0&1\end{matrix} \longrightarrow \begin{matrix}0&0&1\\0&0&0\\1&0&0\end{matrix}$$

Thus we must subtract the I^* = $\begin{matrix}1&0&0\\0&0&0\\0&0&1\end{matrix}$

Note the diagonal is the same as the vector p and that $I^*p = p$ for any p.

Another example

$p^T = (1\ 0\ 1\ 0\ 1)$ then $pp^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$ $I^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

and $I^*p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$

Define $\Delta\tilde{\mathbf{W}} = \mathbf{p}\mathbf{p}^{\mathbf{T}}$; hence $\Delta\mathbf{W} = Z\left(\mathbf{p}\mathbf{p}^{\mathbf{T}}\right)$

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + Z\left(\mathbf{p}\mathbf{p}^{\mathbf{T}}\right)$$

$$= \Delta\mathbf{W} + \mathbf{W}(0)$$

$$= \Delta\mathbf{W}$$

So does $F_h\left(\mathbf{W}_{\text{new}}\mathbf{p}\right) = \mathbf{p}$?

$$F_h\left(\mathbf{W}_{\text{new}}\mathbf{p}^{\mathbf{T}}\right) = F_h\left[\Delta\mathbf{W}\mathbf{p}\right]$$

$$= F_h\left[\left(\Delta\tilde{\mathbf{W}} - \mathbf{I}_{\mathbf{n}}^{*}\right)\mathbf{p}\right]$$

$$= F_h\left[\Delta\tilde{\mathbf{W}}\mathbf{p} - \mathbf{p}\right]$$

$$= F_h\left[\mathbf{p}\mathbf{p}^{\mathbf{T}}\mathbf{p} - \mathbf{p}\right]$$

$$F_h\left(\mathbf{W}_{\text{new}}\mathbf{p}\right) = F_h\left[\mathbf{p}\mathbf{p}^{\mathrm{T}}\mathbf{p} - \mathbf{p}\right]$$

$$= F_h\left[d\mathbf{p} - \mathbf{p}\right]$$

$$= F_h\left[\left(d-1\right)\mathbf{p}\right]$$

where $\mathbf{p}^{\mathrm{T}}\mathbf{p} = d$, a scalar equal to the number of 1's in the vector (do you see why?)

Note that $d \geq 1$ in non-trivial cases. Therefore, the above

$$= \mathbf{p}$$

What About Noisy Inputs?

$$I_3 = \sum_{j=1}^{9} w_{3j} x_j \quad \text{with } w_{33} = 0. \quad \text{In general, } I_i = \sum_{j=1}^{N} w_{ij} x_j$$

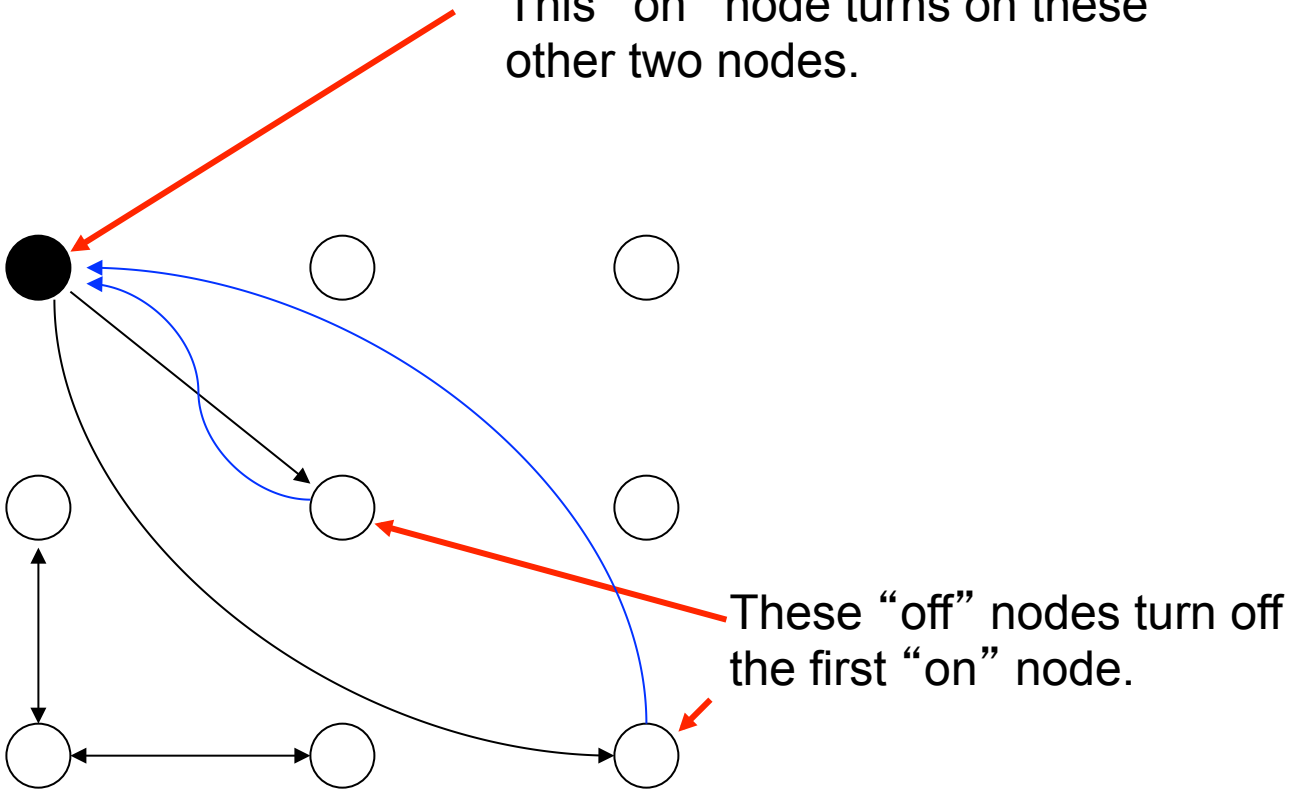
for N = number of neurons = number of elements in a pattern.

Now make a noisy version of the pattern p:

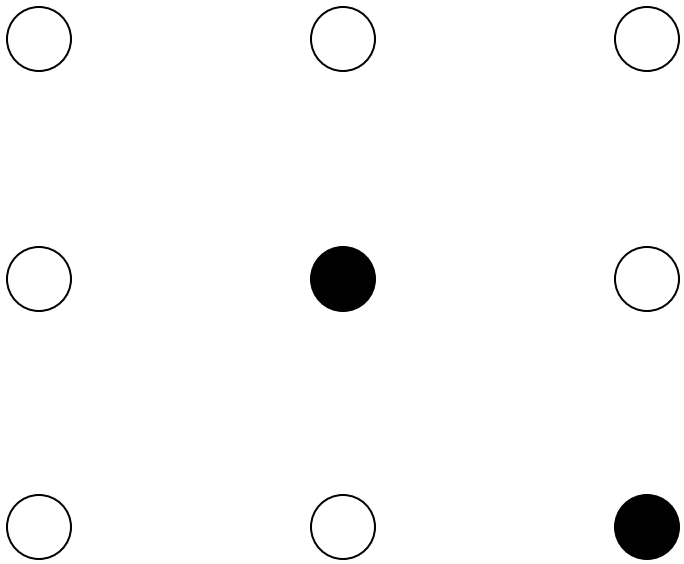
$$\tilde{p} = \left\{ \begin{array}{ccc} \oplus & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array} \right\} = (100\,|\,000\,|\,000)^{\mathbf{T}}$$

$$\mathbf{W}_{\text{new}}\tilde{\mathbf{p}} = \left(\begin{array}{ccc|ccc|ccc} 0&0&0&0&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&0 \end{array}\right) \left(\begin{array}{c} 1\\0\\0\\0\\0\\0\\0\\0\\0 \end{array}\right) = \left(\begin{array}{c} 0\\0\\0\\0\\1\\0\\0\\0\\1 \end{array}\right)$$

Putting this into F_h produces no change.

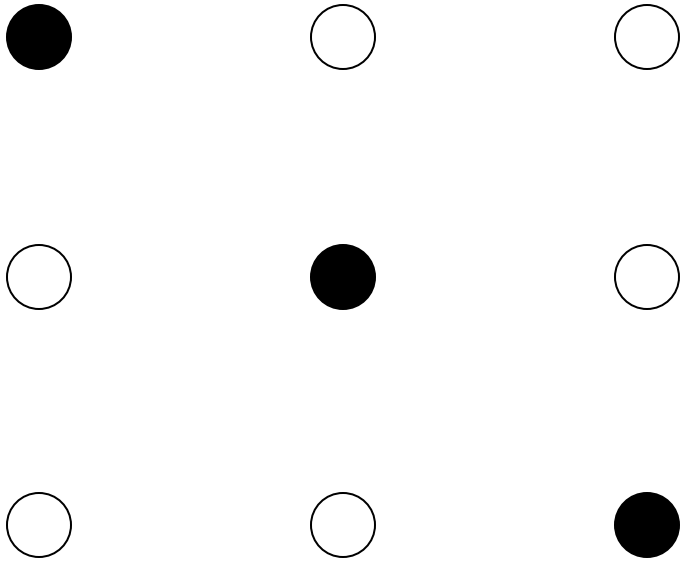

This “on” node turns on these other two nodes.
These “off” nodes turn off the first “on” node.

The resulting pattern:



Now using this as the input in a second iteration, and using the same logic, we get

Our original pattern.

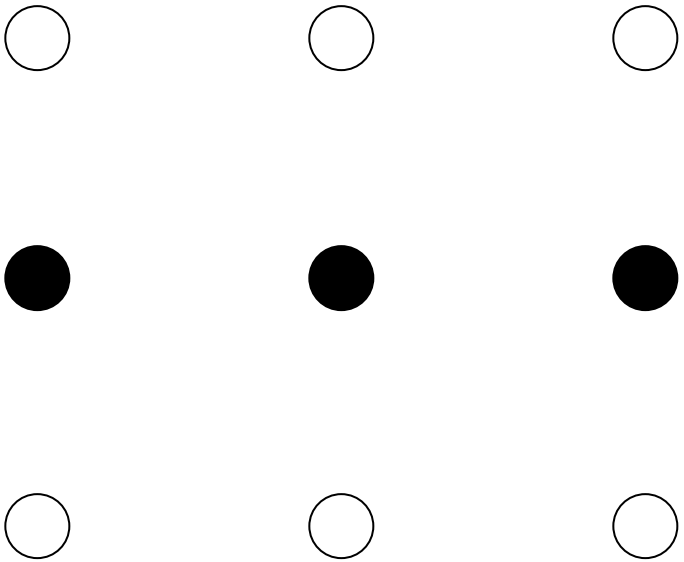


But hold on. Not all input patterns result in our "trained" pattern.

$$\mathbf{W}\begin{pmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}\quad\text{and in fact}\quad\mathbf{W}\begin{pmatrix}0\\\times\\\times\\\times\\0\\\times\\\times\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

This is because the 1 in the first vector was not associated with any 1 in the exemplar, hence, the other 0s turned this 1 to off (0).

What About Another Exemplar?



$$= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\Delta \mathbf{W} = Z\left(\mathbf{q}\mathbf{q}^{\mathbf{T}}\right)$$

$$\mathbf{q}\mathbf{q}^{\mathbf{T}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 000111000 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

$$Z\left(\mathbf{q}\,\mathbf{q}^{\mathbf{T}}\right)=\left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + Z\left(\mathbf{q}\mathbf{q}^{\mathrm{T}}\right) = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right)$$

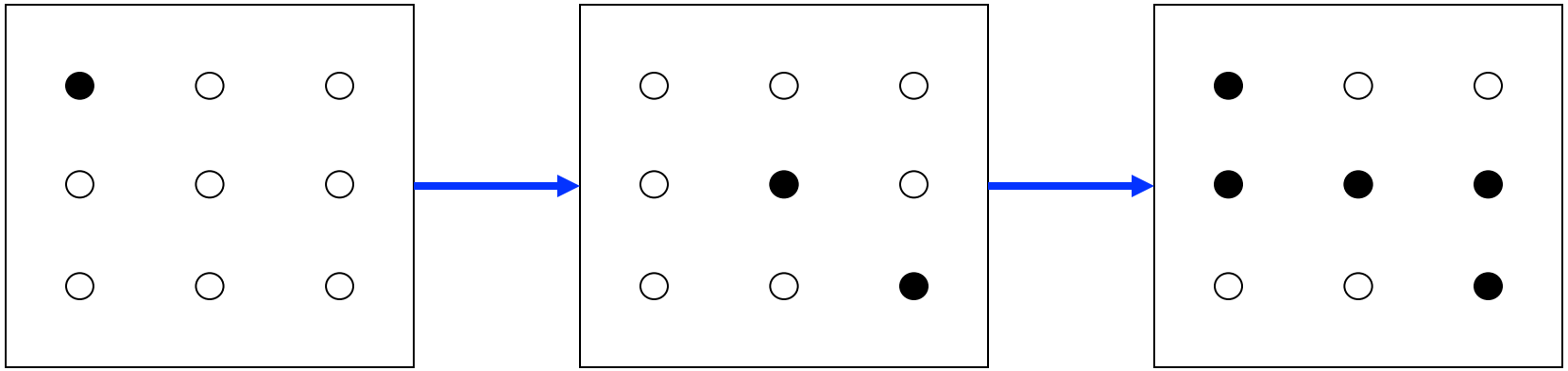
This system “knows” (is trained for) the two exemplars.

$$F_h\left(\mathbf{W}_{\text{new}}\tilde{\mathbf{p}}\right) = F_h\left(\left(\begin{array}{ccc|ccc|ccc} 0&0&0&0&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&1&1&0&0&0\\ 1&0&0&1&0&1&0&0&1\\ 0&0&0&1&1&0&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&0 \end{array}\right)\left(\begin{array}{c}1\\0\\0\\0\\0\\0\\0\\0\\0\end{array}\right)\right) = \left(\begin{array}{c}0\\0\\0\\0\\1\\0\\0\\0\\1\end{array}\right)$$

$$F_h\left(\mathbf{W}_{\text{new}}\tilde{\mathbf{p}}\right) = F_h\left(\left(\begin{array}{ccc|ccc|ccc} 0&0&0&0&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&1&1&0&0&0\\ 1&0&0&1&0&1&0&0&1\\ 0&0&0&1&1&0&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&0 \end{array}\right)\left(\begin{array}{c} 0\\0\\0\\0\\1\\0\\0\\0\\1 \end{array}\right)\right) = \left(\begin{array}{c} \cancel{2}\ 1\\0\\0\\1\\1\\1\\0\\0\\1 \end{array}\right)$$

$$F_h\left(\mathbf{W}_{\text{new}}\tilde{\mathbf{p}}_1\right) = F_h\left(\begin{pmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}\right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \tilde{\mathbf{p}}_2$$

$$F_h\left(\mathbf{W}_{\text{new}}\tilde{\mathbf{p}}_2\right) = F_h\left(\left(\begin{array}{ccc|ccc|ccc} 0&0&0&0&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&1&1&0&0&0\\ 1&0&0&1&0&1&0&0&1\\ 0&0&0&1&1&0&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&0 \end{array}\right)\left(\begin{array}{c}1\\0\\0\\1\\1\\1\\0\\0\\1\end{array}\right)\right) = \left(\begin{array}{c}1\\0\\0\\1\\1\\1\\0\\0\\1\end{array}\right)$$



All exemplar patterns “on”. This is an example of “heat death”.

Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447/625-438
Dr. Mark Fleischer

Module 8.3: Hopfield Networks

This Sub-Module Covers …

- We've examined Hebbian Learning and the notion of 'heat death'.
- Now we examine ways that can provide inhibition and not just excitation to recurrent networks using bipolar values.

Beyond The Hebbian Paradigm

- Node states should reinforce each other if their pattern elements are the same.
- Node states should also inhibit each other if their pattern elements are different.

Here, 'reinforce' means increasing the magnitude of their connecting weights. Inhibition means increasing the magnitude of their connecting weights, but in a negative or opposite direction.

How can this be effected?

The Hopfield Algorithm

$$x_i \in \{-1, 1\}, \quad f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Initialize with M Exemplars and $1 \leq i, j \leq M$:

$$w_{ij} = \begin{cases} \sum_{m=1}^{M} x_i^{[m]} x_j^{[m]} & i \neq j \\ 0 & i = j \end{cases}$$

$x_i(0) = x_i^*$ for all inputs i,

$$x_j(t+1) = f\left[\sum_{i=1}^{M} w_{ij} x_i(t)\right]$$

Instead of using binary values, let's use "bi-polar" values.

$$\mathbf{p} = \{\oplus\cdot\cdot\,|\,\cdot\oplus\cdot\,|\,\cdot\cdot\oplus\}$$
$$= (1\,-1\,-1 \quad -1\;1\,-1 \quad -1\,-1\;1)$$
$$\mathbf{q} = \{\cdot\cdot\cdot\,|\,\oplus\oplus\oplus\,|\,\cdot\cdot\cdot\}$$
$$= (-1\,-1\,-1 \quad 1\;1\;1 \quad -1\,-1\,-1)$$

$$F_h(x) = \begin{cases} 1, \text{if } x > 0 \\ -1, \text{if } x \le 0 \end{cases}$$

Benefits of Bipolar Values

- They magically provide inhibition *and* reinforcement:
- Consider: $I/O_A \times$ link weight = I/O_B
  - If we want to *reinforce* negative values:
    - $-1 \times 1 = -1$
  - If we want to *reinforce positive values:*
    - $1 \times 1 = 1$
  - If we want to inhibit negative values:
    - $-1 \times -1 = 1$
  - If we want to inhibit positive values:
    - $1 \times -1 = -1$

$$\Delta\tilde{\mathbf{W}} = \mathbf{p}\mathbf{p}^{\mathrm{T}} = \begin{pmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

Each row is either **p** or -**p.**

$$\begin{array}{c} \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \end{pmatrix} \\ \Delta\mathbf{W} = \Delta\tilde{\mathbf{W}} - \mathbf{I}_9 = \begin{pmatrix} 0 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \\ -1 & 0 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 0 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 0 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 0 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 0 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 0 \end{pmatrix} \end{array}$$

$$\mathbf{W}_{\text{new}} = \Delta\mathbf{W} + \mathbf{W}_{\text{old}}$$

and so $$F_h\left(\mathbf{W}_{\text{new}}\mathbf{p}\right) = \mathbf{p}$$

Whereas before we had:

$$F_h\left(\mathbf{W}_{\text{new}}\mathbf{p}\right) = F_h\left[\mathbf{p}\mathbf{p}^{\mathrm{T}}\mathbf{p} - \mathbf{p}\right]$$
$$= F_h\left[d\mathbf{p} - \mathbf{p}\right]$$
$$= F_h\left[\left(d-1\right)\mathbf{p}\right]$$

Remember that

$$\mathbf{W} = \mathbf{p}\mathbf{p}^{\mathrm{T}} - \mathbf{I}^*$$

Now, we have this:

$$F_h\left(\mathbf{W}_{\text{new}}\mathbf{p}\right) = F_h\left[\mathbf{p}\mathbf{p}^{\mathrm{T}}\mathbf{p} - \mathbf{p}\right]$$
$$= F_h\left[n\mathbf{p} - \mathbf{p}\right]$$
$$= F_h\left[\left(n-1\right)\mathbf{p}\right]$$

where *n* is the size (length) of the vector **p.**

Do you see where the difference comes from?

And for the second exemplar…

$$\mathbf{q}\mathbf{q}^{\mathrm{T}} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ \hline -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ \hline 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \end{array}\right)$$

$$\Delta\tilde{\mathbf{W}}$$

And for the second exemplar…

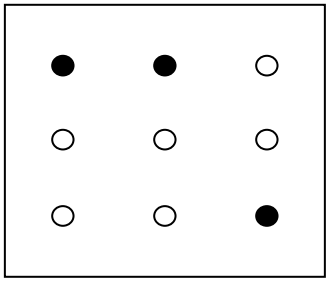
$$Z\left(\mathbf{q}\mathbf{q}^{\mathrm{T}}\right) = \left(\begin{array}{ccc|ccc|ccc} 0 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & -1 & -1 & -1 & 1 & 1 & 1 \\ \hline -1 & -1 & -1 & 0 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 0 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 0 & -1 & -1 & -1 \\ \hline 1 & 1 & 1 & -1 & -1 & -1 & 0 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 0 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 0 \end{array}\right)$$

$$\Delta \mathbf{W}$$

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \Delta\mathbf{W}$$

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

Now examine a noisy representation of **p.**



$$\tilde{\mathbf{p}}^{\mathbf{T}} = \begin{pmatrix} 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

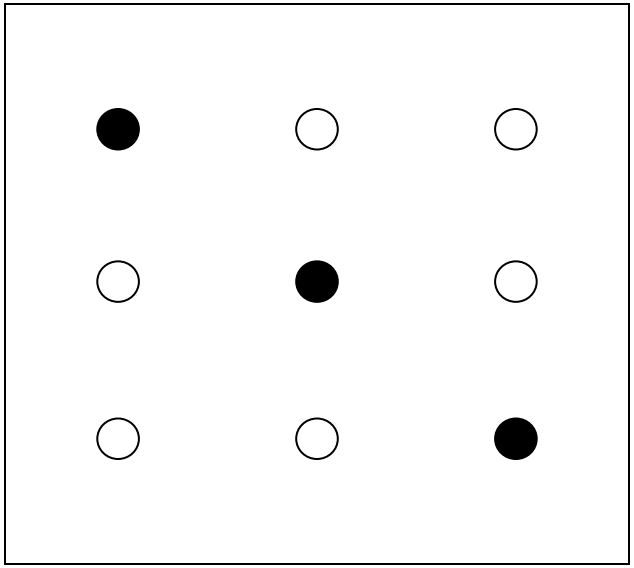
$$F_h\left(\mathbf{W}\tilde{\mathbf{p}}\right) = \mathbf{p}'_1$$

$$F_h\left(\mathbf{W}\mathbf{p}'_1\right) = \mathbf{p}'_2$$

$$\vdots$$

$$F_h\left(\mathbf{W}\mathbf{p}'_{n-1}\right) = \mathbf{p}'_n$$

This will converge to exemplar 1 after only two iterations and does not reflect “heat death”.



The Hopfield Algorithm

$$x_i \in \{-1, 1\}, \quad f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Initialize with M Exemplars and $1 \leq i, j \leq M$:

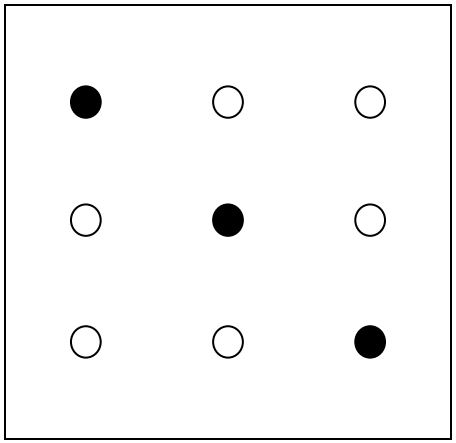
$$w_{ij} = \begin{cases} \sum_{m=1}^{M} x_i^{[m]} x_j^{[m]} & i \neq j \\ 0 & i = j \end{cases}$$

$x_i(0) = x_i^*$ for all inputs i,

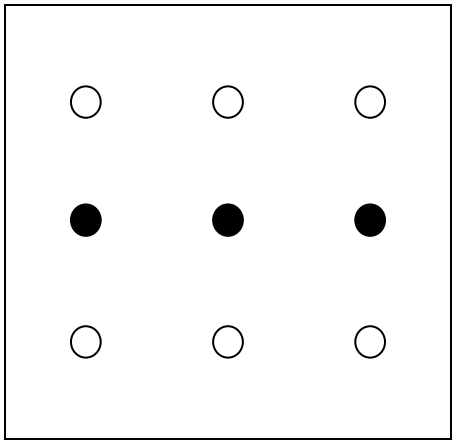
$$x_j(t+1) = f\left[\sum_{i=1}^{M} w_{ij} x_i(t)\right]$$

Suppose we have the following three exemplars.

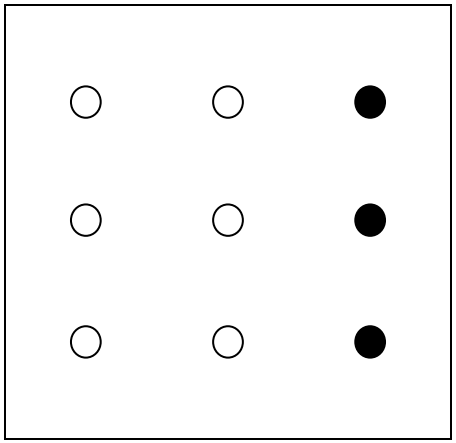
p



q



r



How do you think a Hopfield network trained with these three exemplars will evolve given some part of one of them?

The weight matrix for the first two exemplars was …

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \Delta\mathbf{W}$$

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

And for the third exemplar…the weight matrix is…

$$\mathbf{r}\mathbf{r}^{T} = \begin{pmatrix} -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \end{array}\right)$$

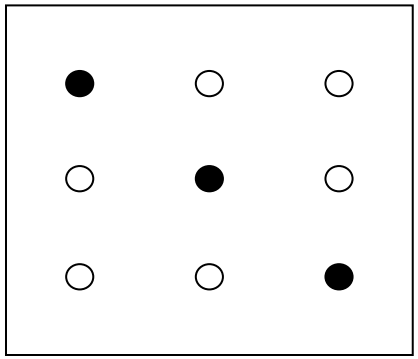
Zeroing out the diagonal …

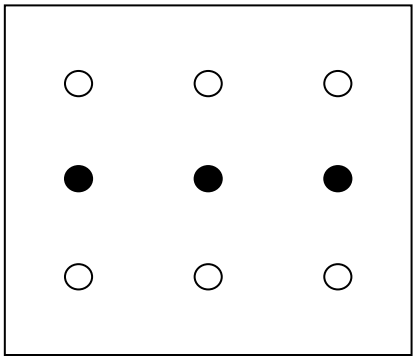
$$Z\left(\mathbf{r}\mathbf{r}^{\mathrm{T}}\right) = \left(\begin{array}{ccc|ccc|ccc} 0 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 0 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 0 & -1 & -1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & -1 & 0 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 0 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 0 & -1 & -1 & 1 \\ \hline 1 & 1 & -1 & 1 & 1 & -1 & 0 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 0 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 0 \end{array}\right)$$

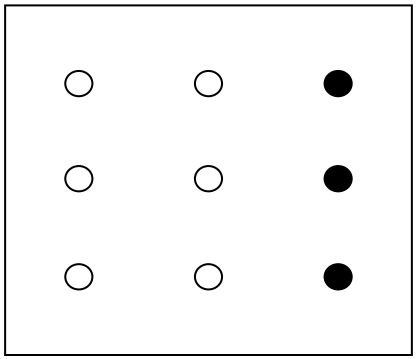
and adding it to the weight matrix for the first two exemplars, we get...

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

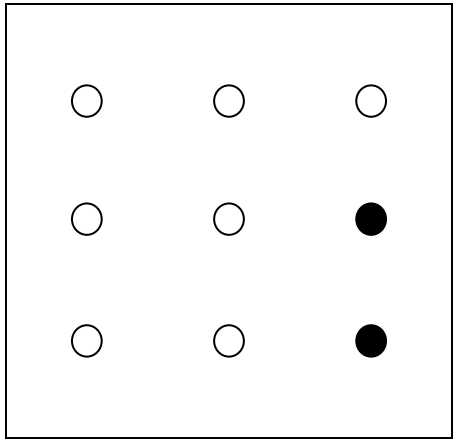
The three exemplars…





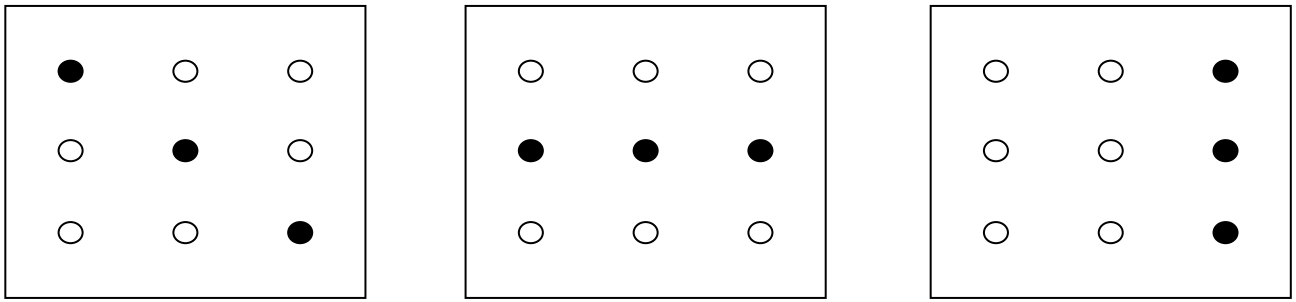


Suppose we present the following pattern to the net.

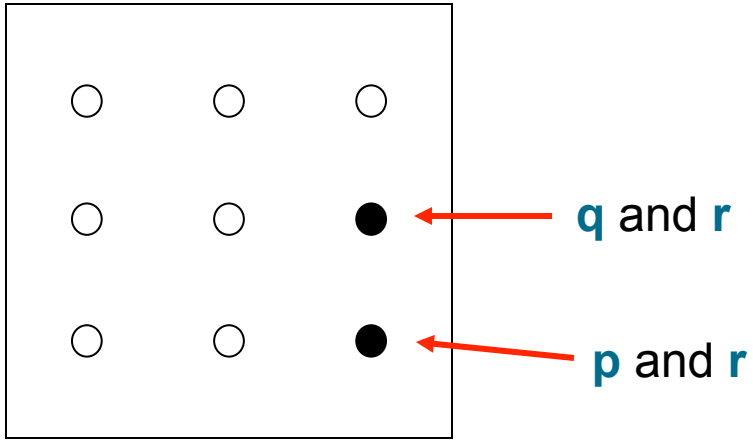


What can we expect the net to converge to? What are some intelligent guess?

The three exemplars…



Suppose we present the following pattern to the net.

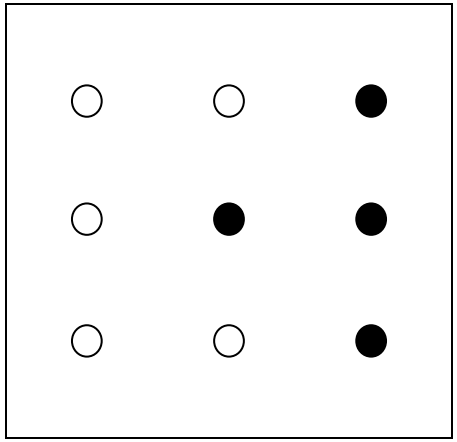


What can we expect the net to converge to? What are some intelligent guess?

and adding it to the weight matrix for the first two exemplars, we get...

$$\begin{pmatrix} 0 & 1 & -1 & -1 & 1 & -3 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & -1 & -1 & 3 & 3 & -1 \\ -1 & 1 & 0 & -1 & -3 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 0 & 1 & 1 & 1 & 1 & -3 \\ 1 & -1 & -3 & 1 & 0 & -1 & -1 & -1 & -1 \\ -3 & -1 & 1 & 1 & -1 & 0 & -1 & -1 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 0 & 3 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 3 & 0 & -1 \\ 1 & -1 & 1 & -3 & -1 & -1 & -1 & -1 & 0 \end{pmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -4 \\ -10 \\ 4 \\ -4 \\ 2 \\ 4 \\ -10 \\ -10 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

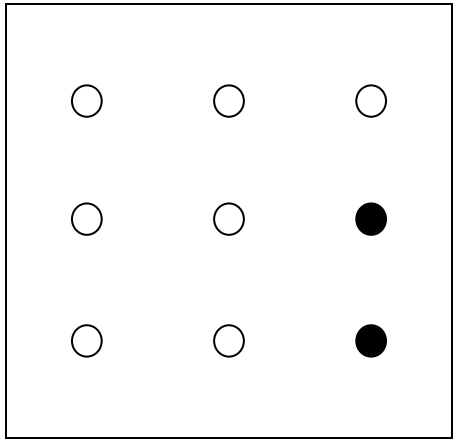
and that vector corresponds to this pattern…



doing another iteration with the last output vector…

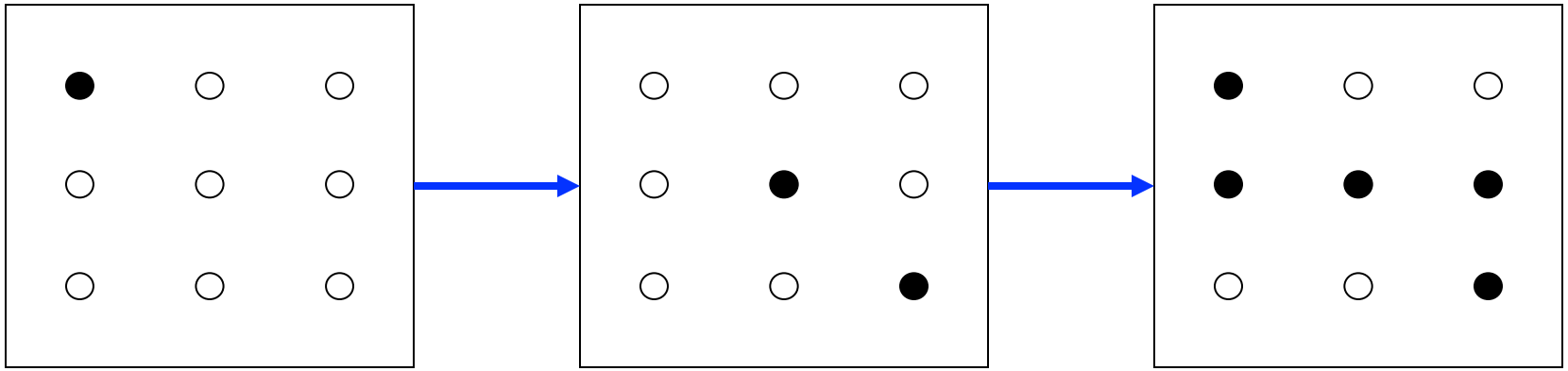
$$\begin{pmatrix} 0 & 1 & -1 & -1 & 1 & -3 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & -1 & -1 & 3 & 3 & -1 \\ -1 & 1 & 0 & -1 & -3 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 0 & 1 & 1 & 1 & 1 & -3 \\ 1 & -1 & -3 & 1 & 0 & -1 & -1 & -1 & -1 \\ -3 & -1 & 1 & 1 & -1 & 0 & -1 & -1 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 0 & 3 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 3 & 0 & -1 \\ 1 & -1 & 1 & -3 & -1 & -1 & -1 & -1 & 0 \end{pmatrix} \begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -4 \\ -10 \\ -2 \\ -4 \\ -4 \\ 4 \\ -10 \\ -10 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

and that vector corresponds to this pattern…



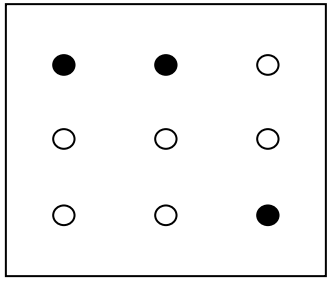
Hmmmm…. what's going on here?

Recall from using the binary values, the net exhibited “heat death”



All exemplar patterns “on”.

Recall a noisy representation of **p**.



$$\tilde{\mathbf{p}}^{\mathbf{T}} = \begin{pmatrix} 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

$$F_h\left(\mathbf{W}\tilde{\mathbf{p}}\right) = \mathbf{p}'_1$$

$$F_h\left(\mathbf{W}\mathbf{p}'_1\right) = \mathbf{p}'_2$$

$$\vdots$$

$$F_h\left(\mathbf{W}\mathbf{p}'_{n-1}\right) = \mathbf{p}'_n$$

Convergence of Hopfield Nets

- Obviously, there is a cycle.
- Why are there chapters concerning convergence of Hopfield Nets?
- If there is a cycle, there is no ‘universal’ convergence.
- Can we consider other modalities?

Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447/625-438
Dr. Mark Fleischer

Module 8.4: Hopfield Convergence

What We've Covered So Far…

- Learned about the Hopfield Network
  - Hebbian Learning
    - Recurrent neural networks
    - Matrix/vector representation of a recurrent network
    - Heat death
  - Excitation and Inhibition in Hopfield Networks via bi-polar values
  - Possibility of cycling in Hopfield networks in part because of inhibition and excitation.
- In this sub-module
  - Another modality to the dynamical system.
  - Prove convergence of the Hopfield network using this modality.

A Peek at the Hopfield Convergence Theorem

- In a Hopfield network, with asynchronous updating, the Hopfield net will always converge.
- This is a **sufficient condition**. It is not necessary—there may be other ways to converge even without the Hopfield conditions (no self connections $w_{ii} = 0$ for all i, symmetric connections $w_{ji} = w_{ij}$).

Convergence

- Want the simplest approach.
- If we simply go down hill, that's easy …
- If there exists some lower-bound to some value --- a bottom of a hill, then we'll eventually reach it.
- In other words, if every possible state has a ranking or metric associated with it (we'll call it energy), then showing that we always reduce that metric is equivalent to showing that we will never get into a cycle.

So What is Asynchronous Updating?

$$\mathbf{x}_{k+1} = F_h\left(\mathbf{W}\mathbf{x}_k\right)$$

$$F_h\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix}$$

$$F_h\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} x_1^* \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

So What is Asynchronous Updating?

$$\mathbf{x}_{k+1} = F_h\left(\mathbf{W}\mathbf{x}_k\right)$$

$$F_h\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix}$$

$$F_h\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix}\begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix}$$

Define the Hecht-Nielsen Function:

$$H(\mathbf{x}) = -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} x_i x_j + \sum_{i=1}^{N} \theta_i x_i$$

Want to prove that

$$\Delta H(\mathbf{x}) \leq 0$$

Define an Update Rule

So, apply the update rule to a given node *k* (the one being updated---remember, asynchronously). *x** refers to the current iteration. Now,

$$x_i^* = \begin{cases} x_k^* & i = k \\ x_i & i \neq k \end{cases}$$

where *k* refers to the one we're updating.

The Update Expression

$$\Delta H = H^* - H = -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}\left(x_i^* x_j^* - x_i x_j\right)$$

Current

Previous Iteration

Remember, the * refers to an updated value.

The Update Expression

$$= \underbrace{-\sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{ij}\left(x_i^* x_j^* - x_i x_j\right)}_{A} \underbrace{-\sum_{i=1}^{N} w_{ik}\left(x_i^* x_k^* - x_i x_k\right)}_{B}$$

Simplifying Part *B* first, we get:

$$B = -\sum_{\substack{i=1 \\ i\neq k}}^{N} w_{ik}\left(x_i x_k^* - x_i x_k\right) = -\sum_{\substack{i=1 \\ i\neq k}}^{N} w_{ik}\left(x_k^* - x_k\right)x_i$$

Why?

The Update Expression

Remember, $$x_i^* = \begin{cases} x_k^* & i = k \\ x_i & i \neq k \end{cases}$$

$$B = -\sum_{\substack{i=1 \\ i \neq k}}^{N} w_{ik}\left(x_i x_k^* - x_i x_k\right) = -\sum_{\substack{i=1 \\ i \neq k}}^{N} w_{ik}\left(x_k^* - x_k\right) x_i$$

First, $w_{kk} = 0$ (no self-connections) and $x_i^* = x_i$ for $i \neq k$.
Which reduces to:

$$= -\sum_{\substack{i=1 \\ i \neq k}}^{N} w_{ik} x_i \Delta x_k$$ for Part B.

The Update Expression

$$= \underbrace{-\cancel{\sum_{i=1}^{N}}\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{ij}\left(x_i^* x_j^* - x_i x_j\right)}_{A} \underbrace{- \sum_{i=1}^{N} w_{ik}\left(x_i^* x_k^* - x_i x_k\right)}_{B}$$

Now Part A, recall that for *i or j* ≠ *k*, $x_i^* = x_i$ so the sum

$$\left(x_i^* x_j^* - x_i x_j\right) = 0$$

so *i* = *k* is the only term left, so

$$A = -\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj}\left(x_k^* x_j - x_k x_j\right) = -\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j \Delta x_k$$

Let's Sum the deltas…

$$\Delta H = A + B = -\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j \Delta x_k - \sum_{\substack{i=1 \\ i\neq k}}^{N} w_{ik} x_i \Delta x_k$$

$$= -\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j \Delta x_k - \sum_{\substack{j=1 \\ j\neq k}}^{N} w_{jk} x_j \Delta x_k$$

$$= -\sum_{\substack{j=1 \\ j\neq k}}^{N} \left(w_{kj} + w_{jk}\right) x_j \Delta x_k = -2 \underbrace{\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j}_{Activity\ a_k} \Delta x_k$$

Implications for cases

$$\Delta H = -2 \underbrace{\sum_{\substack{j=1 \\ j \neq k}}^{N} w_{kj} x_j \Delta x_k}_{Activity\ a_k}$$

Suppose

$$a_k > 0, \quad \text{then} \quad x_k^* = f_h(a_k) = 1 > 0$$

then

$$\Delta x_k = x_k^* - x_k \geq 0.$$

Implications for cases

$$\Delta H = -2\underbrace{\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j \Delta x_k}_{Activity\ a_k}$$

Activity $a_k > 0$

$$\Downarrow$$

$$\Delta x_k = x_k^* - x_k \geq 0.$$

Why? Well, if $x^* = 1$, then x must have been either 1 (unchanged) or -1 (changed) hence 1-1=0 or 1 - -1 = 2 >0 . Thus, $-2a_k\Delta x_k \leq 0$.

Implications for cases

$$\Delta H = -2\underbrace{\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j}_{Activity\ a_k} \Delta x_k$$

Suppose $a_k < 0, \quad \text{then} \quad x_k^* = f_h(a_k) = -1 < 0$

Activity $a_k < 0$

$\Downarrow$

$$\Delta x_k = x_k^* - x_k \leq 0.$$

Implications for cases

$$\Delta H = -2\underbrace{\sum_{\substack{j=1 \\ j\neq k}}^{N} w_{kj} x_j}_{Activity\ a_k} \Delta x_k$$

Activity $a_k < 0$

$$\Delta x_k = x_k^* - x_k \leq 0.$$

Why? Well, if $x^* = -1$, then x must have been either -1 (unchanged) or 1 (changed) hence -1- -1=0 or -1 - 1 = -2 < 0 . So again, $-2a_k\Delta x_k \leq 0$.

H-N Function Never Increases

- This means, the *H* value can only decrease and since there is a lower bound, it will eventually converge such that

$$f_h\left(\sum_j w_{ij} x_j\right) = \mathbf{x}$$

Cannot oscillate between solutions. Why?

Now $$\Delta H = -2a_k \Delta x_k = \begin{cases} 0 \\ -4|a_k| \end{cases}$$

For $a_k > 0$:

If $\Delta x_k = 0$, then no change in state, hence no oscillation.

If $\Delta x_k \neq 0$ then $\Delta x_k = 2$ and $\Delta H < 0$ as we showed above. Can't increase *H.*

For $a_k \leq 0$:

In this case, then $\Delta H = 0$ if $a_k = 0$ or $\Delta x_k = 0$.
The latter means no change. If $a_k = 0$ then $\Delta H = 0$ and
Again no change in *H.*