Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. HOPFIELD

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

Luke Frankiw and Jonathan Kenny

Topics in Systems Neuroscience
California Institute of Technology
April 26th, 2017

John Hopfield

- Professor of Chemistry and Biology at Caltech from 1980 to 1997.

John Hopfield

- Professor of Chemistry and Biology at Caltech from 1980 to 1997.
- Co-founded the *Computation and Neural Systems* option at Caltech in 1986.

John Hopfield

- Professor of Chemistry and Biology at Caltech from 1980 to 1997.
- Co-founded the *Computation and Neural Systems* option at Caltech in 1986.
- Best known for his proposal to use a neural network to understand brain function (associative memory).

John Hopfield

- Professor of Chemistry and Biology at Caltech from 1980 to 1997.
- Co-founded the *Computation and Neural Systems* option at Caltech in 1986.
- Best known for his proposal to use a neural network to understand brain function (associative memory).
- Revitalized interest into neural network research during the early 1980s.

John Hopfield

- Professor of Chemistry and Biology at Caltech from 1980 to 1997.
- Co-founded the *Computation and Neural Systems* option at Caltech in 1986.
- Best known for his proposal to use a neural network to understand brain function (associative memory).
- Revitalized interest into neural network research during the early 1980s.
- Still an active researcher, proposed a model uniting associative memory and deep learning networks at *NIPS* 2016.

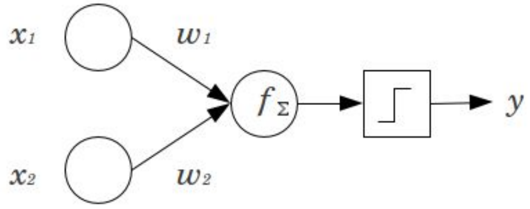
Neurons performing complex computational tasks in nature

- Do collective phenomena simply result from having many simple units working together?
- In Sea Slugs: “Few neurons to obtain elementary useful biological behavior.”

Neurons performing complex computational tasks in nature

- Do collective phenomena simply result from having many simple units working together?
- In Sea Slugs: “Few neurons to obtain elementary useful biological behavior.”

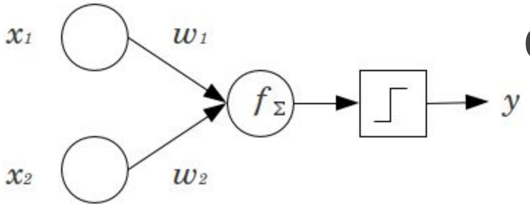
~8000 simple units



Neurons performing complex computational tasks in nature

- Do collective phenomena simply result from having many simple units working together?
- In Sea Slugs: “Few neurons to obtain elementary useful biological behavior.”

~8000 simple units



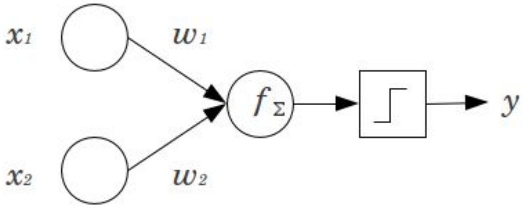
Organization

Neurons performing complex computational tasks in nature

- Do collective phenomena simply result from having many simple units working together?
- In Sea Slugs: “Few neurons to obtain elementary useful biological behavior.”

~8000 simple units

Complex behavior

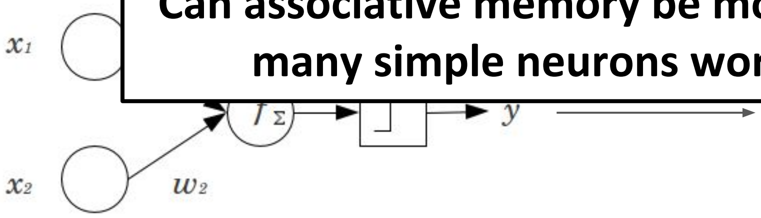


Organization

Neurons performing complex computational tasks in nature

- Do collective phenomena simply result from having many simple units working together?
- In Sea Slugs: “Few neurons to obtain elementary useful biological behavior.”

~8000 simple units



Complex behavior

Can associative memory be modeled by having many simple neurons work together?

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

Access with partial inputs

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

Neural networks and physical systems... PNAS. 1982

Access with partial inputs

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

Neural networks and physical systems... PNAS. 1982

Access with partial inputs

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

Neural networks and physical systems... PNAS. 1982

Hopfield JJ. PNAS.

Access with partial inputs

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

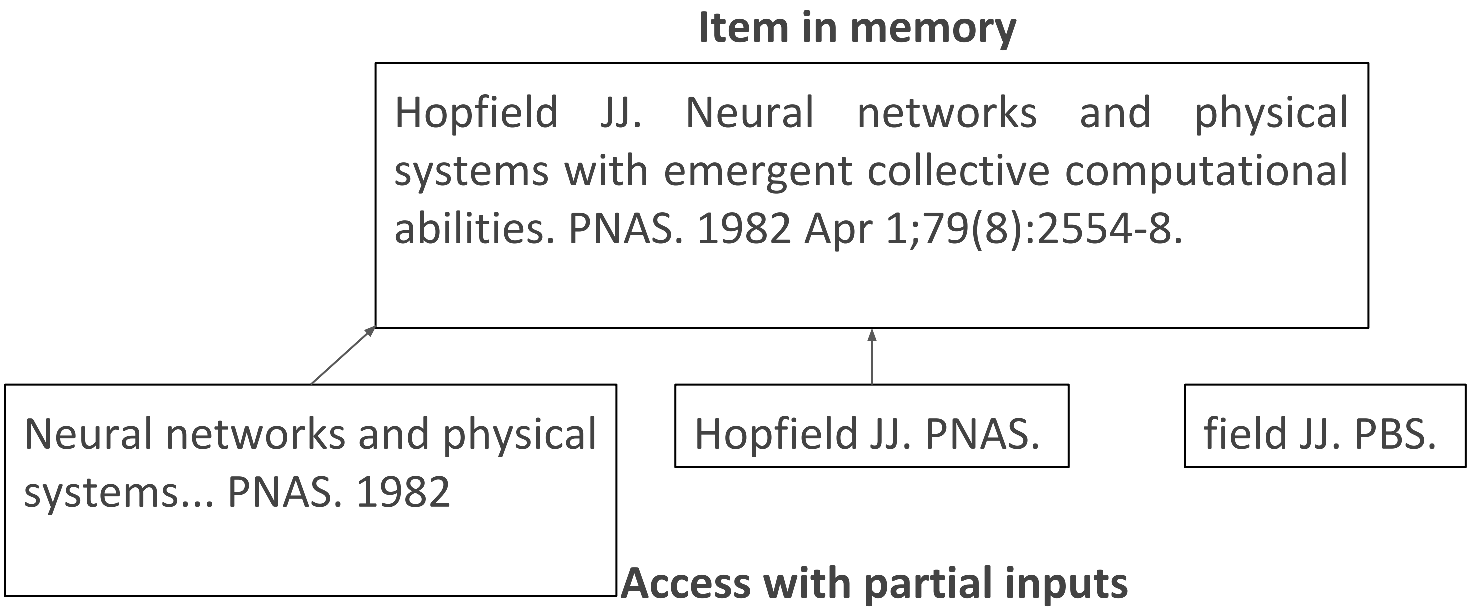
Neural networks and physical systems... PNAS. 1982

Hopfield JJ. PNAS.

Access with partial inputs

Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:



Associative memory

- Also known as *content-addressable memory*, associative memory is crucial for pattern completion in the brain/hardware systems:

Item in memory

Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982 Apr 1;79(8):2554-8.

Neural networks and physical systems... PNAS. 1982

Hopfield JJ. PNAS.

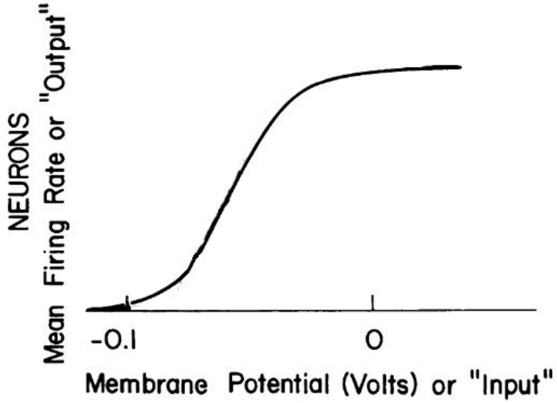
field JJ. PBS.

Access with partial inputs

Associative memory (Hopfield) network structure

Short-term mean firing rate (neglect individual spikes)

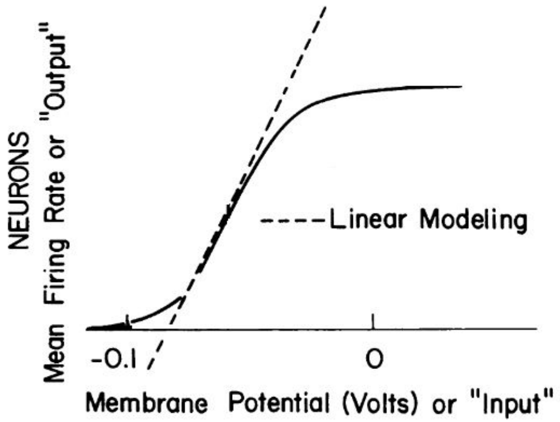
Neuron/unit (s)



Associative memory (Hopfield) network structure

Short-term mean firing rate (neglect individual spikes)

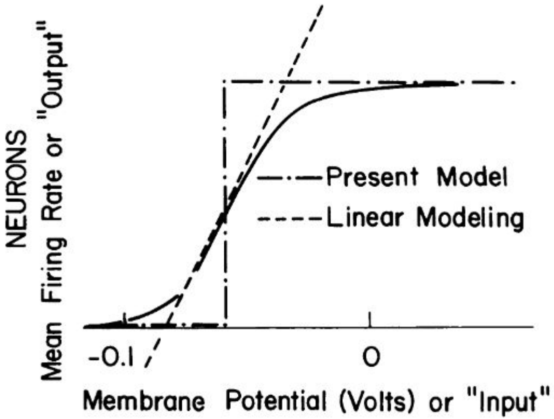
Neuron/unit (s)



Associative memory (Hopfield) network structure

Short-term mean firing rate (neglect individual spikes)

Neuron/unit (s)

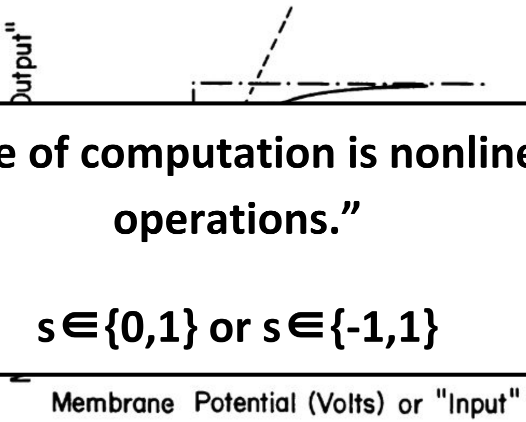


Associative memory (Hopfield) network structure

Short-term mean firing rate (neglect individual spikes)

Neuron/unit (

s_1

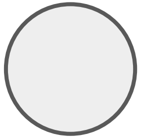


"The essence of computation is nonlinear logical operations."

$s \in \{0,1\}$ or $s \in \{-1,1\}$

Associative memory (Hopfield) network structure

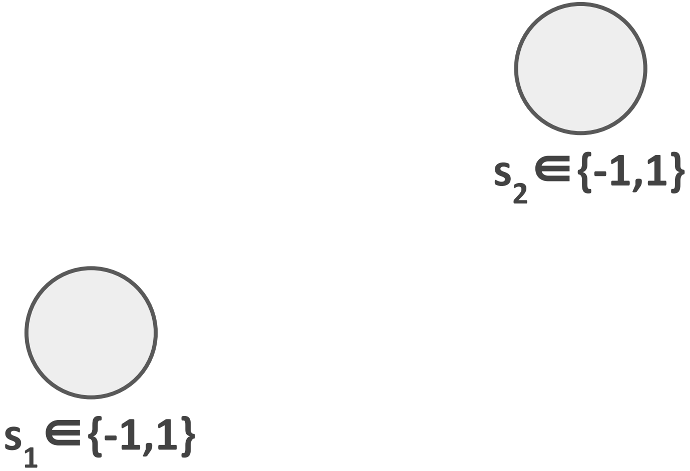
Neuron/unit (s)



$s_1 \in \{-1,1\}$

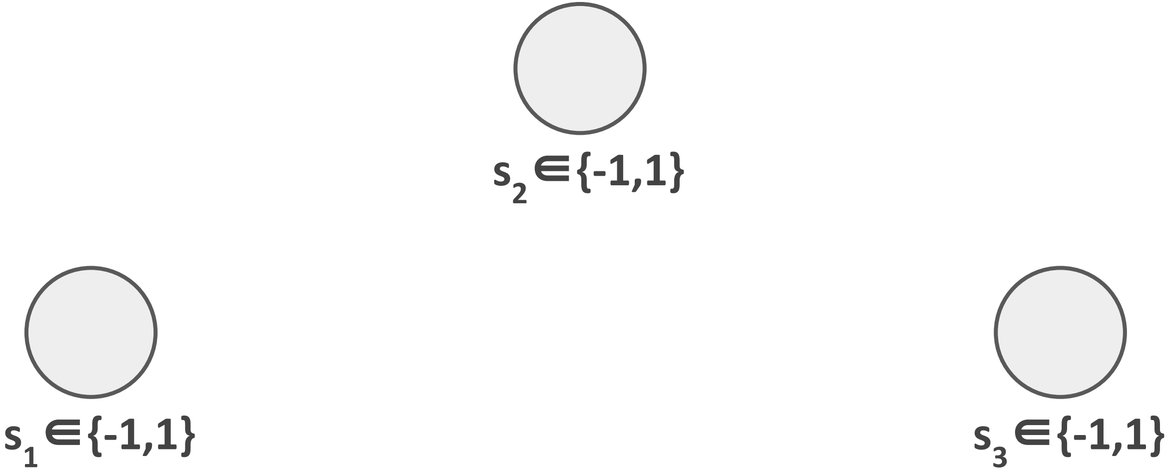
Associative memory (Hopfield) network structure

Neuron/unit (s)



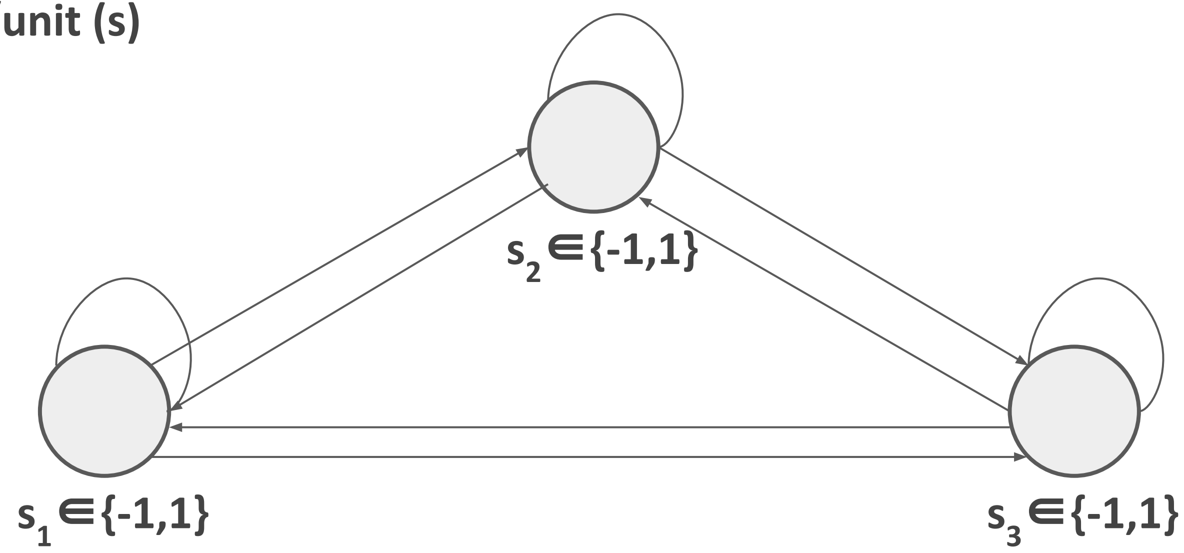
Associative memory (Hopfield) network structure

Neuron/unit (s)



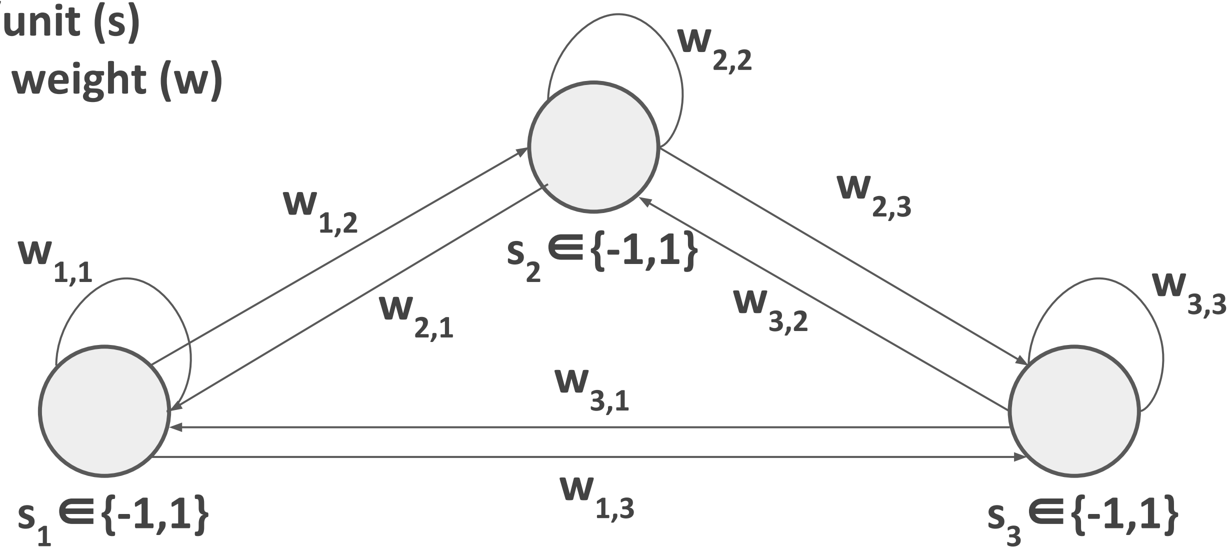
Associative memory (Hopfield) network structure

Neuron/unit (s)



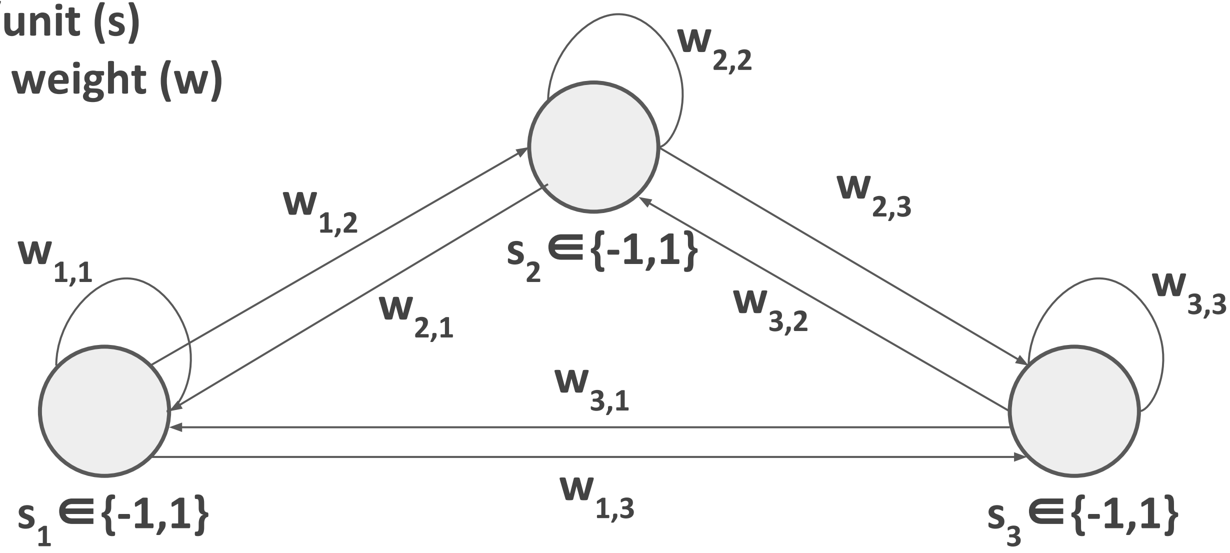
Associative memory (Hopfield) network structure

Neuron/unit (s)
Synaptic weight (w)



Associative memory (Hopfield) network structure

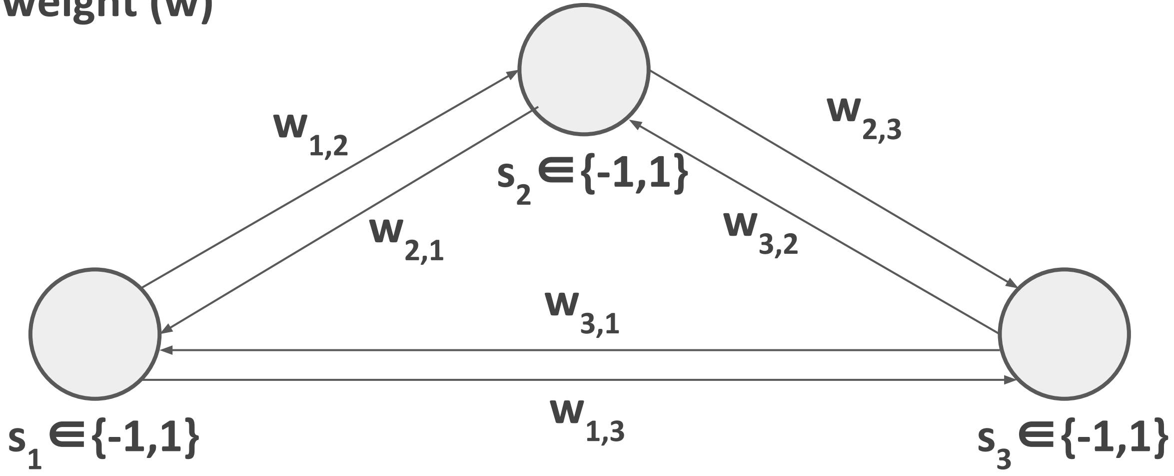
Neuron/unit (s)
Synaptic weight (w)



Some restrictions on weights (w) and neurons (s):

Associative memory (Hopfield) network structure

Neuron/unit (s)
Synaptic weight (w)

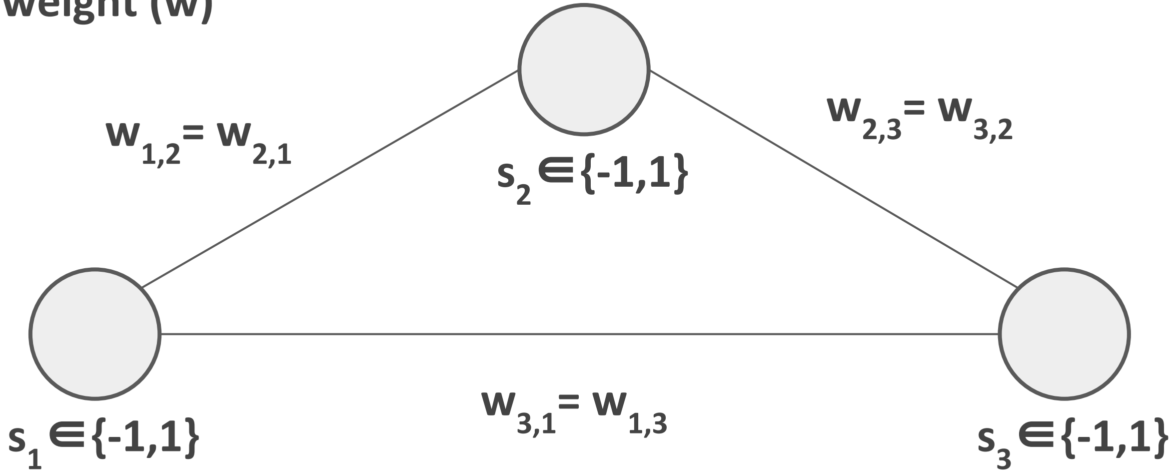


Some restrictions on weights (w) and neurons (s):

- No recursive connections

Associative memory (Hopfield) network structure

Neuron/unit (s)
Synaptic weight (w)

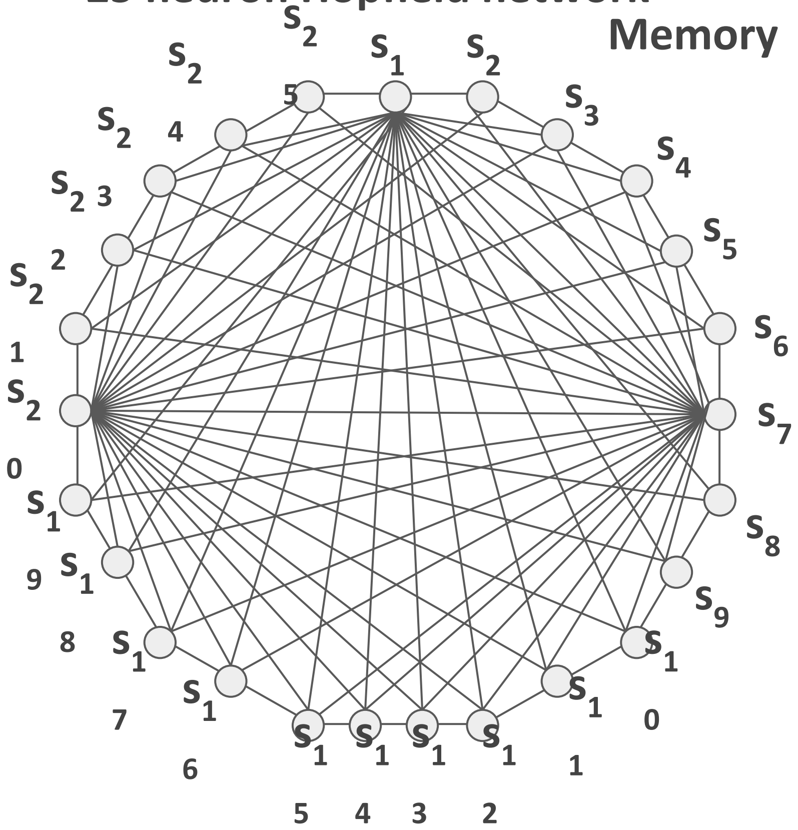


Some restrictions on weights (w) and neurons (s):

- No recursive connections
- No directed connections ($\mathbf{w}_{i,j} = \mathbf{w}_{j,i}$)

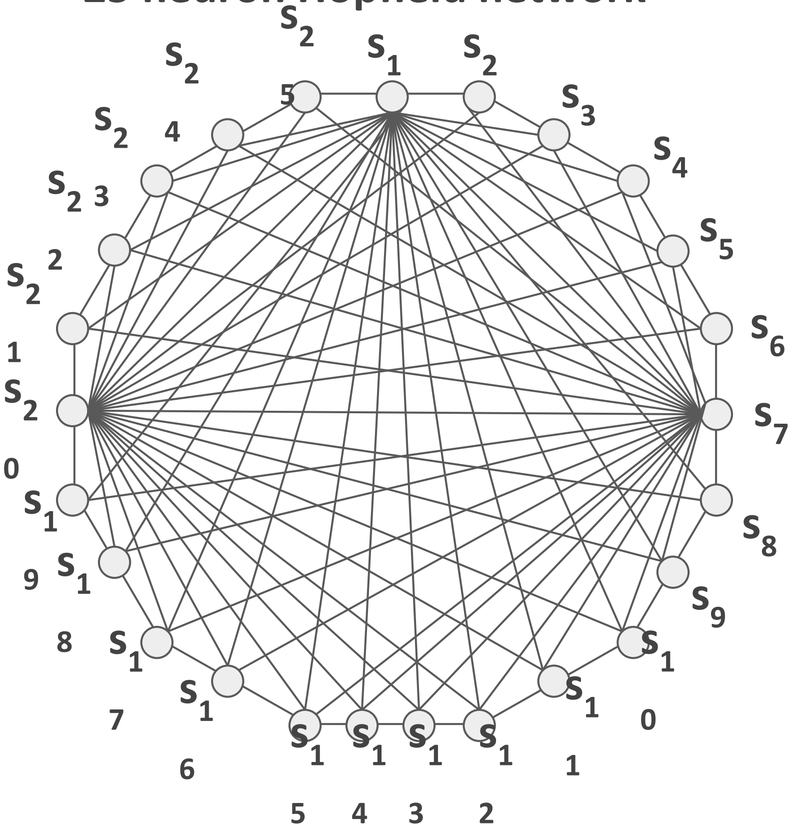
Associative memory (Hopfield) network example

25 neuron Hopfield network



Associative memory (Hopfield) network example

25 neuron Hopfield network

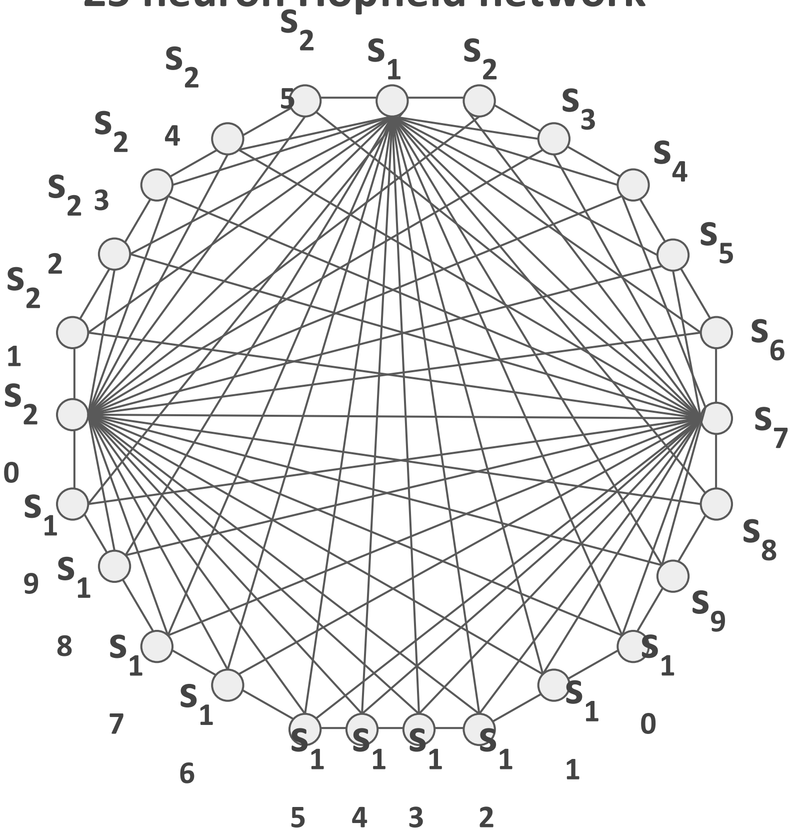


Memory

S_1	S_2	S_3	S_4	S_5
S_6	S_7	S_8	S_9	S_{10}
S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
S_{16}	S_{17}	S_{18}	S_{19}	S_{20}
S_{21}	S_{22}	S_{23}	S_{24}	S_{25}

Associative memory (Hopfield) network example

25 neuron Hopfield network

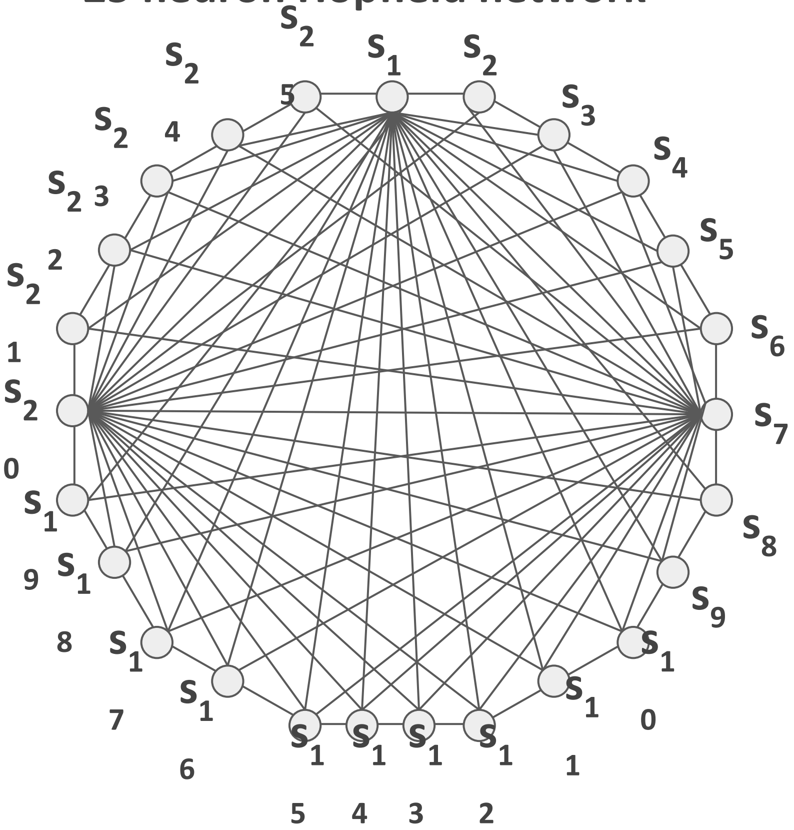


Memory

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

Associative memory (Hopfield) network example

25 neuron Hopfield network



Memory

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

Train weights to include this memory, with **weight update algorithm**.

Associative memory (Hopfield) network example

Partial/incorrect pattern → **Memory**

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

Associative memory (Hopfield) network example

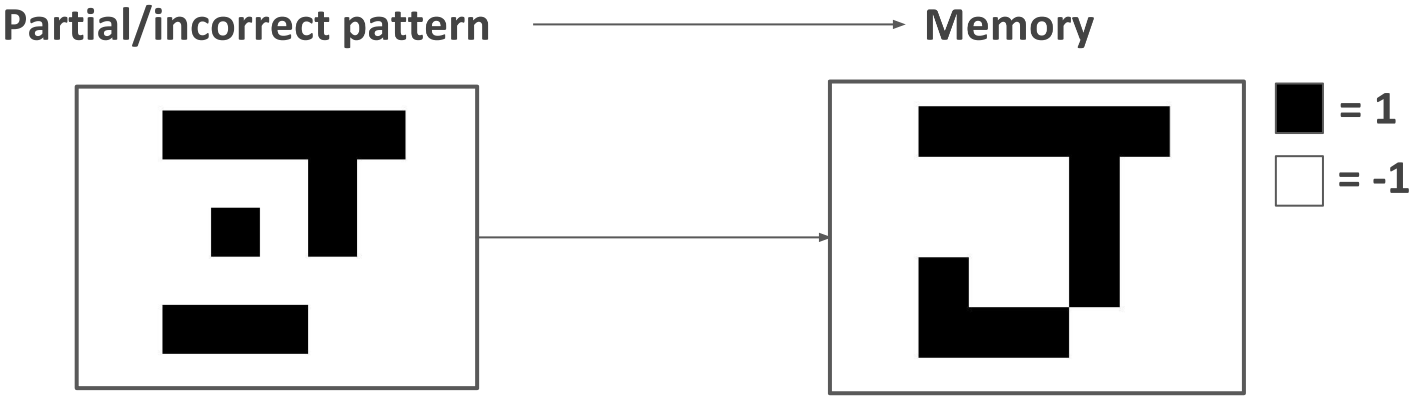
Partial/incorrect pattern → **Memory**

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

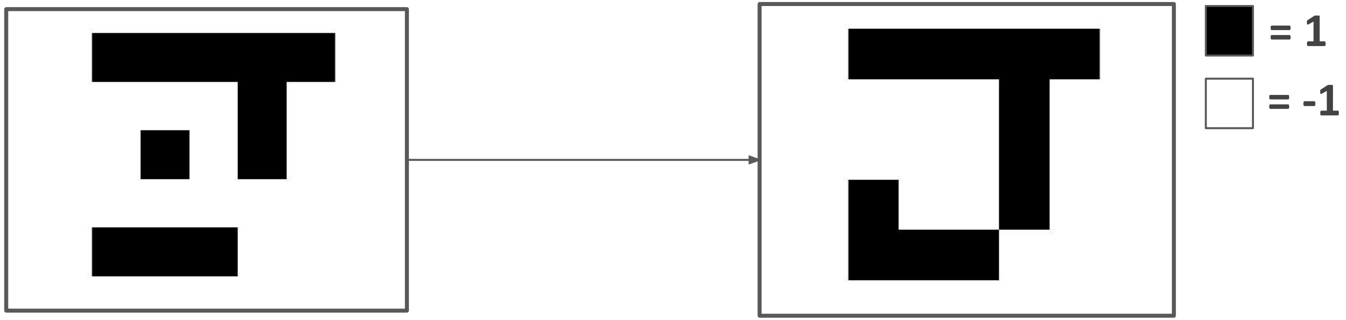
Associative memory (Hopfield) network example

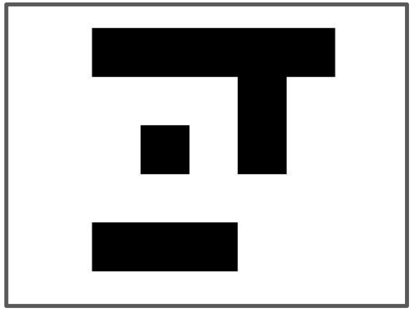


Perform pattern completion using
neuron update algorithm.

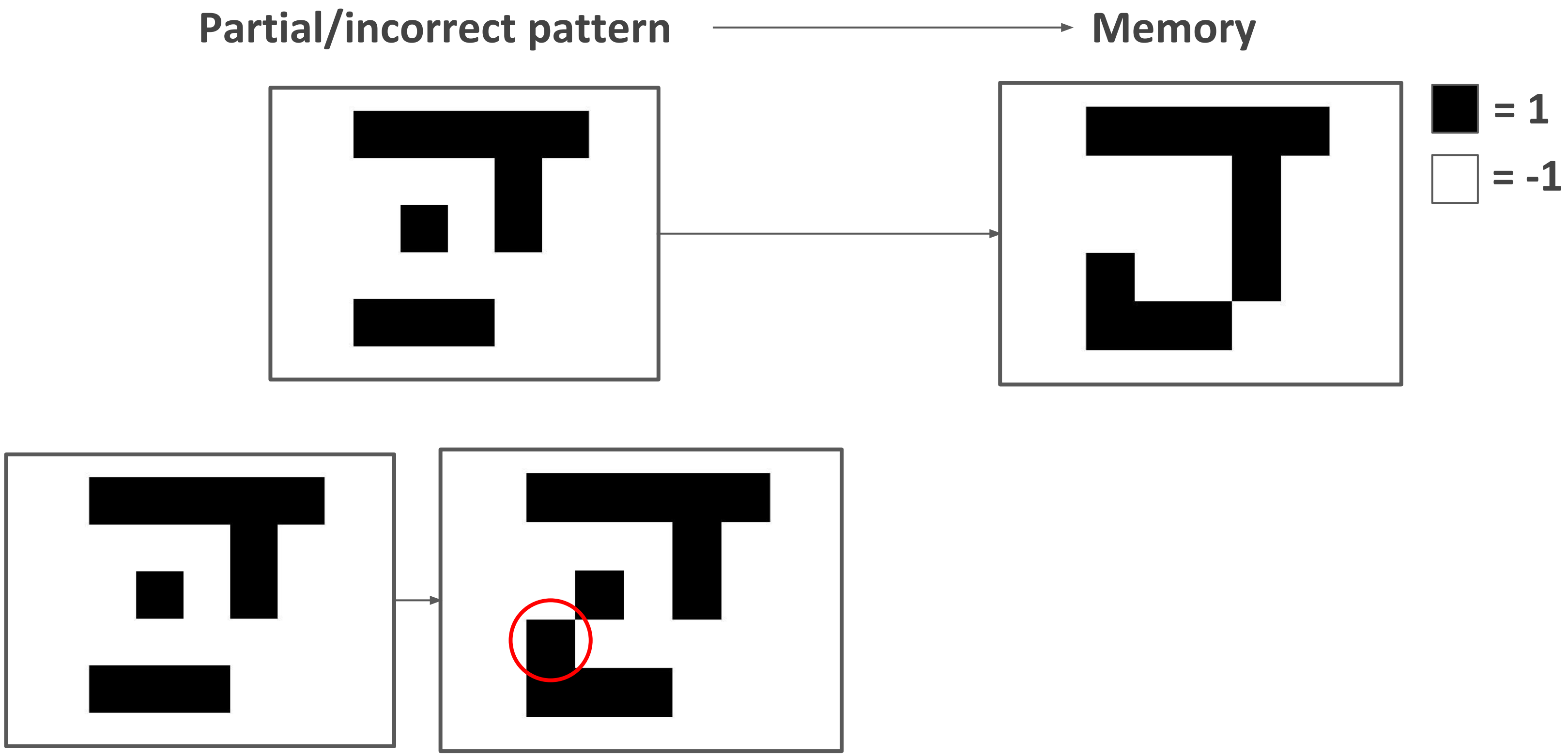
Associative memory (Hopfield) network example

Partial/incorrect pattern → **Memory**

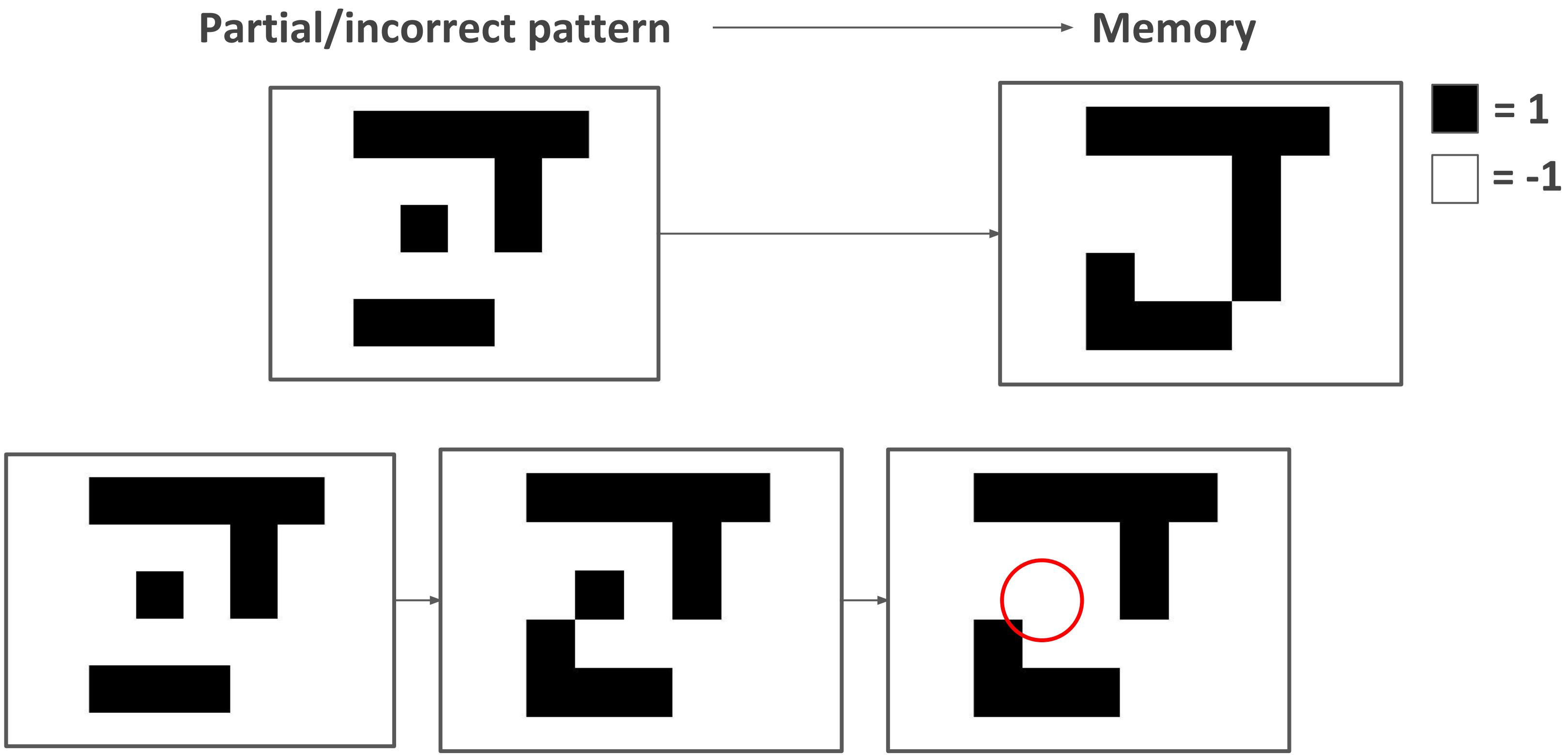




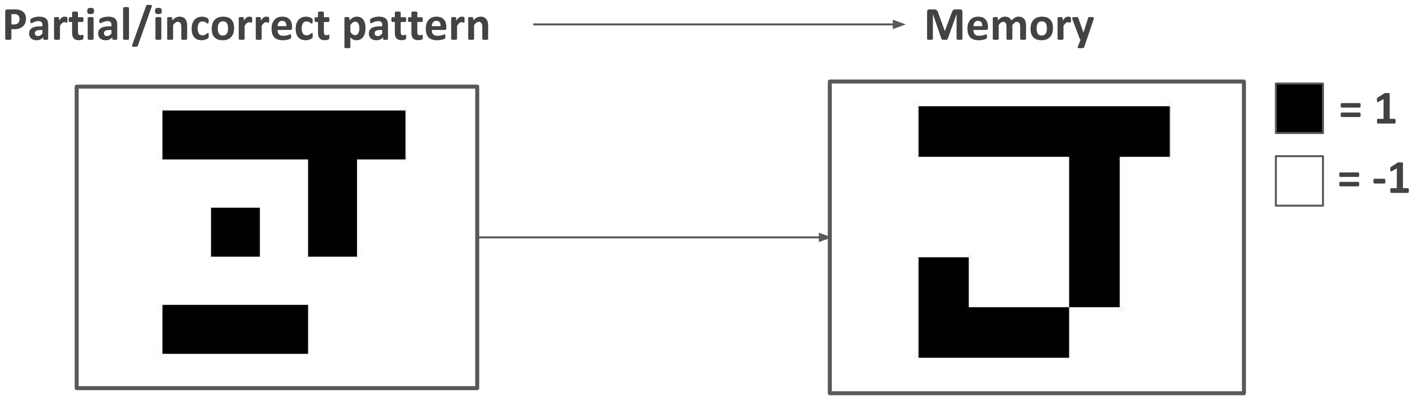
Associative memory (Hopfield) network example

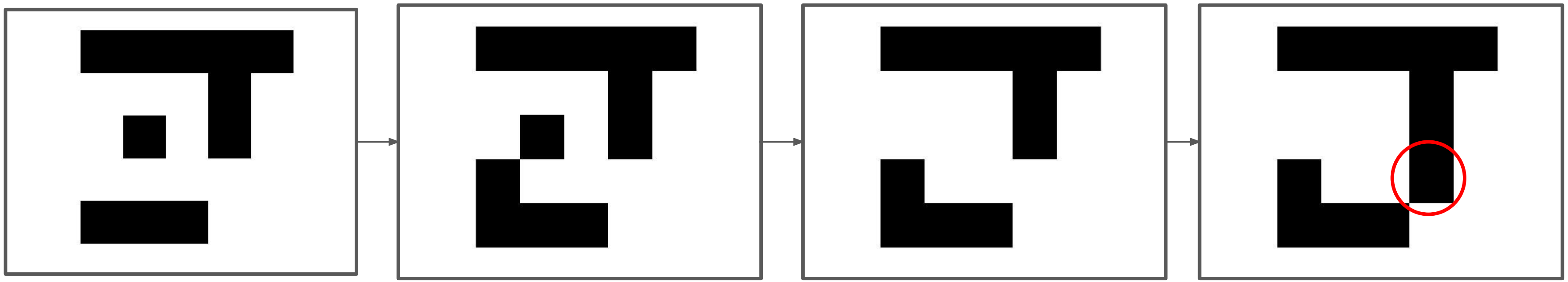


Associative memory (Hopfield) network example



Associative memory (Hopfield) network example





Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training

Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training

Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Store memory in network

(weight update algorithm)

Weight training

Large Hopfield network

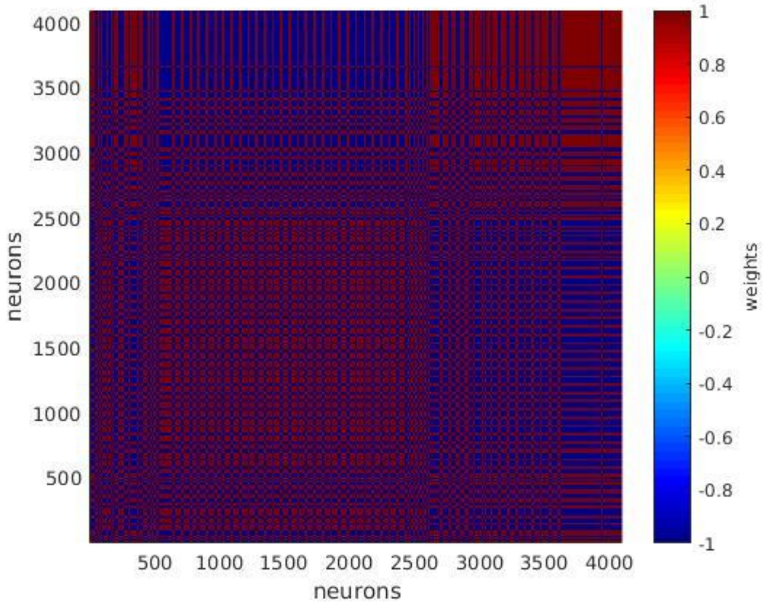
4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Store memory in network

(weight update algorithm)

Weight training

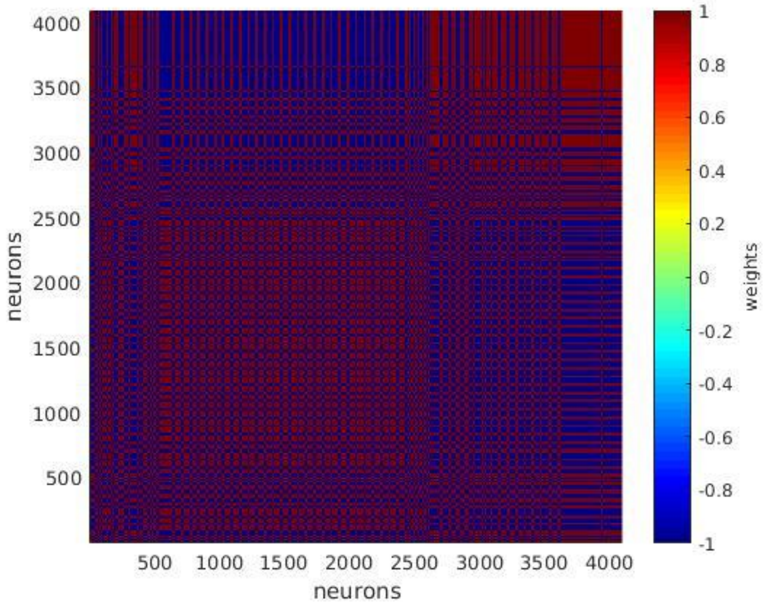


Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training

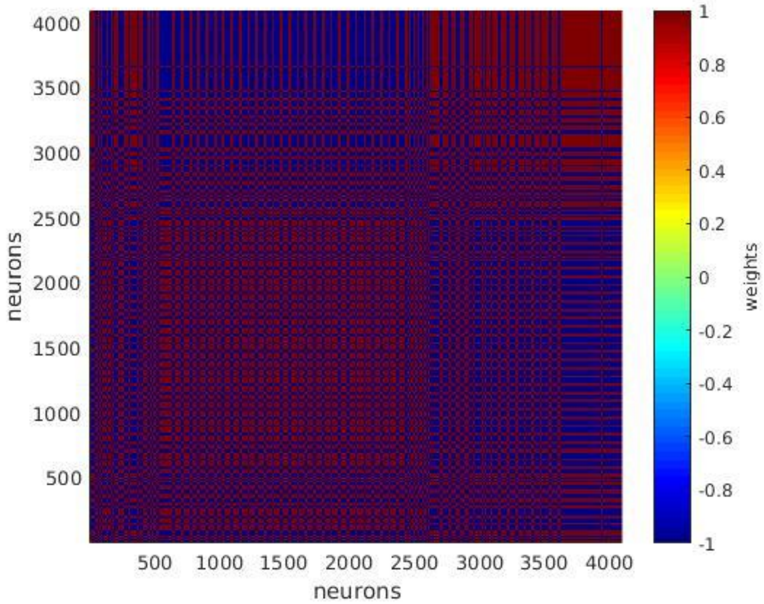


Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training



Large Hopfield network

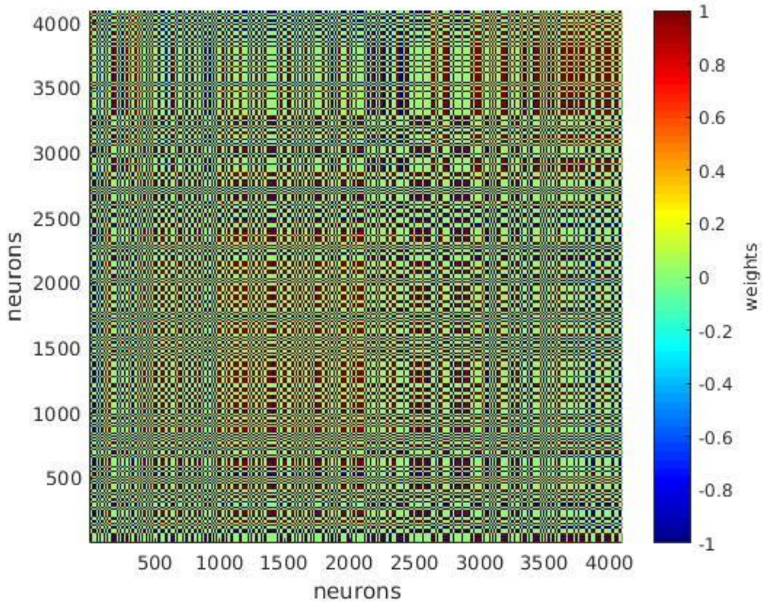
4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training

Store memory in network

(weight update algorithm)

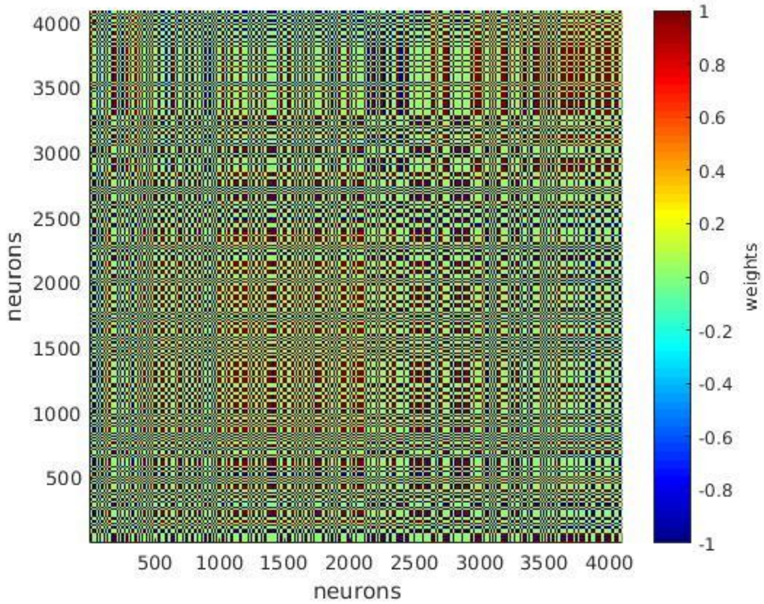


Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training

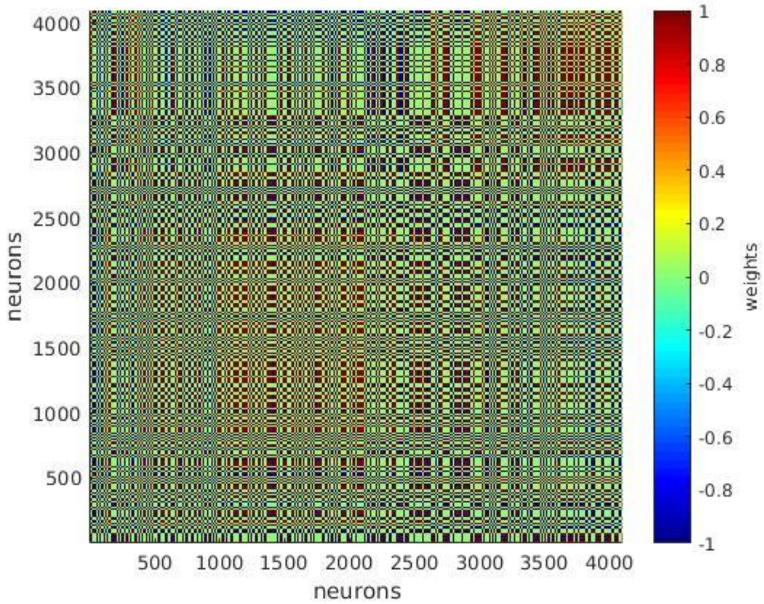


Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Weight training



Large Hopfield network

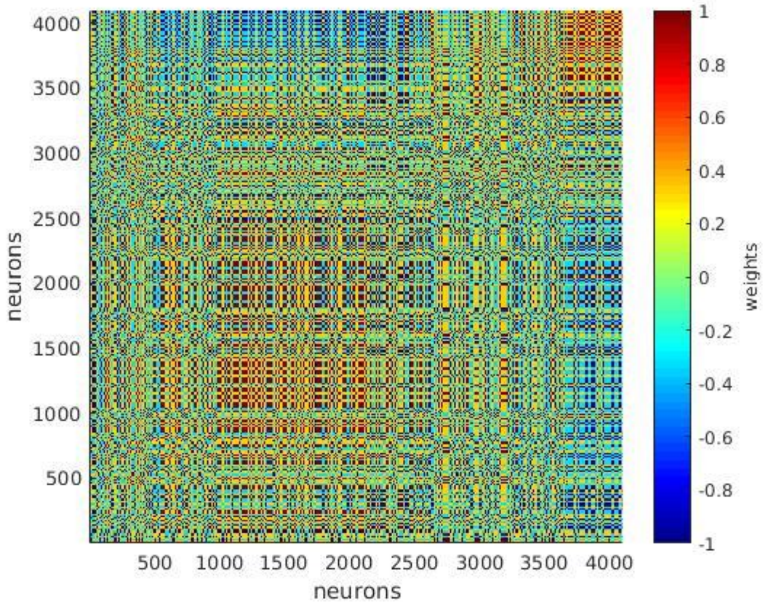
4,096 neuron Hopfield network (8.3 million weights)

Memory encoding in the neurons

Store memory in network

(weight update algorithm)

Weight training



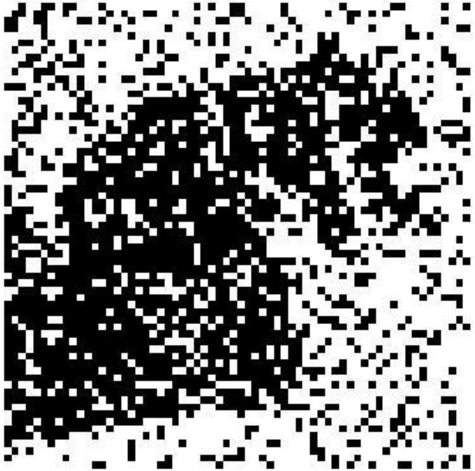
Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory

Add noise
(20% bit flip)

Incomplete Memory



Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory

Add noise
(20% bit flip)

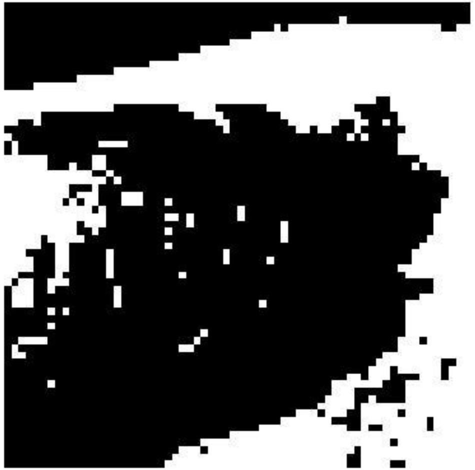
(neuron update algorithm)

Pattern completion

Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory



Add noise
(50% bit flip)

Incomplete Memory

Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory

Add noise
(50% bit flip)

→

(neuron update
algorithm)

Pattern completion

Large Hopfield network

4,096 neuron Hopfield network (8.3 million weights)

Memory

Add noise

Unstable energy configurations can lead to “spurious” memories.

algorithm)

Pattern completion

Five neuron example

- Let's consider the case where we have a five node Hopfield net made up of 0’s and 1’s.

Five neuron example

- Let's consider the case where we have a five node Hopfield net made up of 0’s and 1’s.

- We will use the model system described in the paper.
  - Each node has two states: 0 when “not firing” and 1 when “firing”.

$$\begin{array}{l} V_i \rightarrow 1 \\ V_i \rightarrow 0 \end{array} \quad \text{if} \quad \sum_{j \neq i} T_{ij} V_j \quad \begin{array}{l} > U_i \\ < U_i \end{array} \quad . \qquad [1]$$

- Suppose we want to store two patterns V^1 and V^2.

  $\mathbf{V^1 = (0\ 1\ 1\ 0\ 1)}$

  $\mathbf{V^2 = (1\ 0\ 1\ 0\ 1)}$

- Suppose we want to store two patterns V^1 and V^2.

  V^1 = (0 1 1 0 1)

  V^2 = (1 0 1 0 1)

- We will use the prescribed information storage algorithm from Hopfield's paper.
  - Note: in this example we use 0's and 1's as did Hopfield in his paper. If we were to use -1's and 1's, the information algorithm simplifies.

$$T_{ij} = \sum_s (2V_i^s - 1)(2V_j^s - 1) \qquad [2]$$

Weight Matrix

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

- Begin by calculating the upper diagonal for the first pattern: $\mathbf{V^1 = (0\ 1\ 1\ 0\ 1)}$

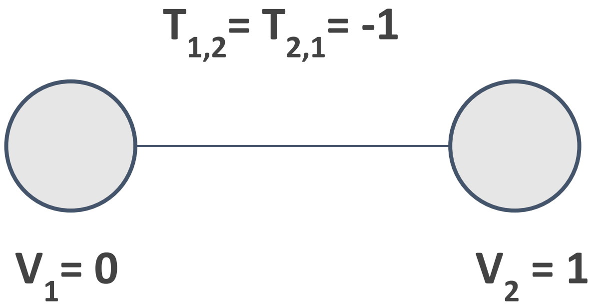
- Let's look at $\mathbf{T_{12}}$.

$$T_{12} = (2V_1 - 1)(2V_2 - 1)$$

$$T_{12} = (2*0 - 1)(2*1 - 1)$$

$$T_{12} = (-1)(1)$$

$$T_{12} = -1$$



Weight Matrix

$$
\begin{matrix}
0 & \mathbf{-1} & T_{13} & T_{14} & T_{15} \\
\mathbf{-1} & 0 & T_{23} & T_{24} & T_{25} \\
T_{31} & T_{32} & 0 & T_{34} & T_{35} \\
T_{41} & T_{42} & T_{43} & 0 & T_{45} \\
T_{51} & T_{52} & T_{53} & T_{54} & 0
\end{matrix}
$$

$\mathbf{T_{13}} = (2V_1 - 1)(2V_3 - 1) = (0 - 1)(2 - 1) = (-1)(1) = -1$

$\mathbf{T_{14}} = (2V_1 - 1)(2V_4 - 1) = (0 - 1)(0 - 1) = (-1)(-1) = 1$

$\mathbf{T_{15}} = (2V_1 - 1)(2V_5 - 1) = (0 - 1)(2 - 1) = (-1)(1) = -1$

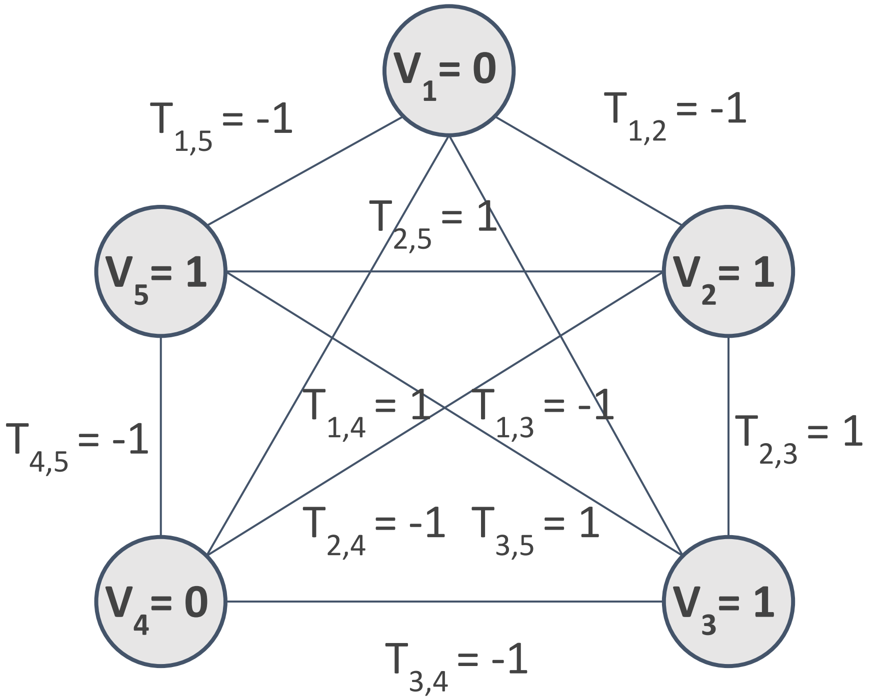
$\mathbf{T_{23}} = (2V_2 - 1)(2V_3 - 1) = (2 - 1)(2 - 1) = (1)(1) = 1$

$\mathbf{T_{24}} = (2V_2 - 1)(2V_4 - 1) = (2 - 1)(0 - 1) = (1)(-1) = -1$

$\mathbf{T_{25}} = (2V_2 - 1)(2V_5 - 1) = (2 - 1)(2 - 1) = (1)(1) = 1$

$\mathbf{T_{34}} = (2V_3 - 1)(2V_4 - 1) = (2 - 1)(0 - 1) = (1)(-1) = -1$

$\mathbf{V^1} = (0, 1, 1, 0, 1)$



Weight Matrix

$$\begin{array}{ccccc}
0 & -1 & -1 & 1 & -1 \\
T_{21} & 0 & 1 & -1 & 1 \\
T_{31} & T_{32} & 0 & -1 & 1 \\
T_{41} & T_{42} & T_{43} & 0 & -1 \\
T_{51} & T_{52} & T_{53} & T_{54} & 0
\end{array}$$

- We know by the formula below, our weight matrix is symmetrical:

$$T_{ij} = (2V_i - 1)(2V_j - 1) = (2V_j - 1)(2V_i - 1) = T_{ji}$$

- We know by the formula below, our weight matrix is symmetrical:

$$\mathbf{T_{ij} = (2V_i - 1)(2V_j - 1) = (2V_j - 1)(2V_i - 1) = T_{ji}}$$

- Thus, we can reflect and fill in the rest of the matrix.

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

Now lets calculate the matrix for the second pattern:

$V^2 = (1\ 0\ 1\ 0\ 1)$

$T_{12} = (2V_1 - 1)(2V_2 - 1) = (2 - 1)(0 - 1) = (1)(-1) = -1$

$T_{13} = (2V_1 - 1)(2V_3 - 1) = (2 - 1)(2 - 1) = (1)(1) = 1$

$T_{14} = (2V_1 - 1)(2V_4 - 1) = (2 - 1)(0 - 1) = (1)(-1) = -1$

$T_{15} = (2V_1 - 1)(2V_5 - 1) = (2 - 1)(2 - 1) = (1)(1) = 1$

$T_{23} = (2V_2 - 1)(2V_3 - 1) = (0 - 1)(2 - 1) = (-1)(1) = -1$

$T_{24} = (2V_2 - 1)(2V_4 - 1) = (0 - 1)(0 - 1) = (-1)(-1) = 1$

$T_{25} = (2V_2 - 1)(2V_5 - 1) = (0 - 1)(2 - 1) = (-1)(1) = -1$

$T_{34} = (2V_3 - 1)(2V_4 - 1) = (2 - 1)(0 - 1) = (1)(-1) = -1$

$T_{35} = (2V_3 - 1)(2V_5 - 1) = (2 - 1)(2 - 1) = (1)(1) = 1$

$T_{45} = (2V_4 - 1)(2V_5 - 1) = (0 - 1)(2 - 1) = (-1)(1) = -1$

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

Now, as in [2], we add the matrices together to get a final weight matrix.

$$T_{ij} = \sum_s (2V_i^s - 1)(2V_j^s - 1) \qquad [2]$$

Now, as in [2], we add the matrices together to get a final weight matrix.

$$T_{ij} = \sum_s (2V_i^s - 1)(2V_j^s - 1) \qquad [2]$$

$$\begin{matrix} 0 & -1 & -1 & 1 & -1 \\ -1 & 0 & 1 & -1 & 1 \\ -1 & 1 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & -1 \\ -1 & 1 & 1 & -1 & 0 \end{matrix} \quad + \quad \begin{matrix} 0 & -1 & 1 & -1 & 1 \\ -1 & 0 & -1 & 1 & -1 \\ 1 & -1 & 0 & -1 & 1 \\ -1 & 1 & -1 & 0 & -1 \\ 1 & -1 & 1 & -1 & 0 \end{matrix}$$

Final Weight Matrix

0	-2	0	0	0
-2	0	0	0	0
0	0	0	-2	2
0	0	-2	0	-2
0	0	2	-2	0

- So now we have a weight matrix for a five neuron Hopfield network that's meant to recognize the memories/patterns **(0 1 1 0 1)** and **(1 0 1 0 1)**.

- So now we have a weight matrix for a five neuron Hopfield network that's meant to recognize the memories/patterns **(0 1 1 0 1)** and **(1 0 1 0 1)**.

- The next step is to choose a state, in this case we will select the state **(1 1 1 1 1)** and update the network until we reach a stable state.

- So now we have a weight matrix for a five neuron Hopfield network that's meant to recognize the memories/patterns **(0 1 1 0 1)** and **(1 0 1 0 1)**.

- The next step is to choose a state, in this case we will select the state **(1 1 1 1 1)** and update the network until we reach a stable state.

- The Hamming distance between the two patterns is 2. The smaller this distance, the harder it is to tell these patterns apart.

- So now we have a weight matrix for a five neuron Hopfield network that's meant to recognize the memories/patterns **(0 1 1 0 1)** and **(1 0 1 0 1)**.

- The next step is to choose a state, in this case we will select the state **(1 1 1 1 1)** and update the network until we reach a stable state.

- The Hamming distance between the two patterns is 2. The smaller this distance, the harder it is to tell these patterns apart.

- As an analogy, consider how it might be more difficult to discriminate between an “O” and a “Q” as compared to two letters that are much different (i.e. have a greater hamming distance).

Updating the nodes

- Quite simply, to update the nodes of the Hopfield network you do a weighted sum of the inputs from the other nodes.
  - If the value is less than your threshold (in this case U = 0), you output 0.
  - Otherwise, the output is 1.
  - We will also make the stipulation that if the weighted sum = 0, the output is 1.
  - This is directly from [1] below:

$$\begin{array}{l} V_i \to 1 \\ V_i \to 0 \end{array} \quad \text{if} \quad \sum_{j \neq i} T_{ij} V_j \begin{array}{l} > U_i \\ < U_i \end{array} . \qquad [1]$$

- **There are two ways to update the nodes:**
  - **Asynchronous:** one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating
  -

- **There are two ways to update the nodes:**
  - **Asynchronous:** one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating
  - **Synchronous:** the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum.
-

- **There are two ways to update the nodes:**
  - **Asynchronous:** one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating
  - **Synchronous:** the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum.

- The original network proposed by Hopfield involved asynchronous processing where the nodes were randomly updated.
-

- **There are two ways to update the nodes:**
  - **Asynchronous:** one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating
  - **Synchronous:** the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum.

- The original network proposed by Hopfield involved asynchronous processing where the nodes were randomly updated.
- Asynchronous processing is believed to better model how our neurons fire.
-

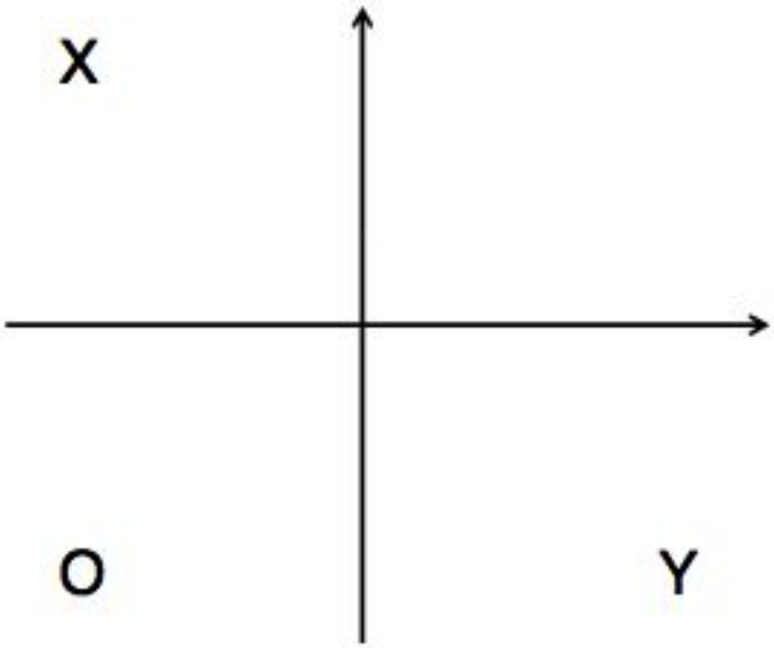
- **There are two ways to update the nodes:**
  - **Asynchronous:** one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating
  - **Synchronous:** the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum.

- The original network proposed by Hopfield involved asynchronous processing where the nodes were randomly updated.
- Asynchronous processing is believed to better model how our neurons fire.
- Additionally, problems involving oscillatory states arise with synchronous updating.

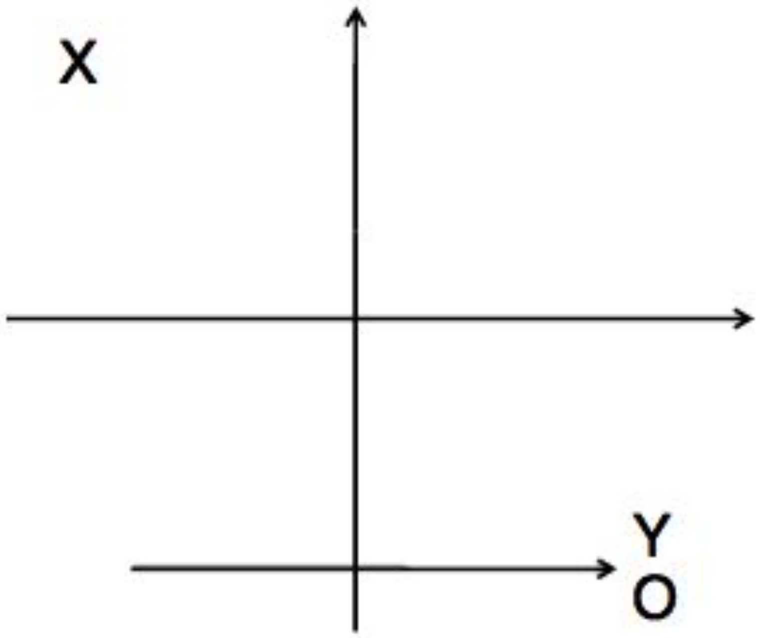
- **There are two ways to update the nodes:**
  - **Asynchronous:** one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating
  - **Synchronous:** the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum.

- The original network proposed by Hopfield involved asynchronous processing where the nodes were randomly updated.
- Asynchronous processing is believed to better model how our neurons fire.
- Additionally, problems involving oscillatory states arise with synchronous updating.
- For the sake of simplicity, we will update the nodes one by one in order a fixed order **1, 5, 2, 4, 3**.

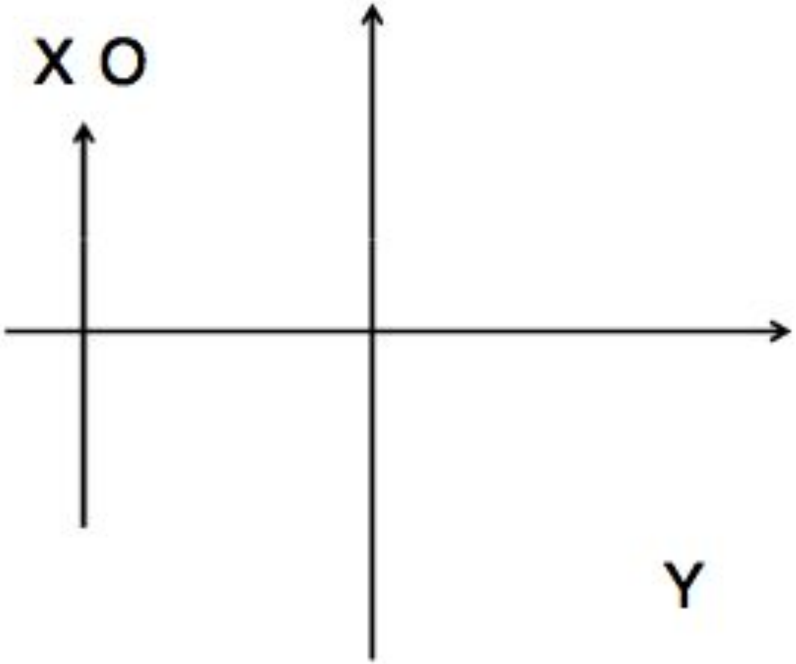
Asynchronous



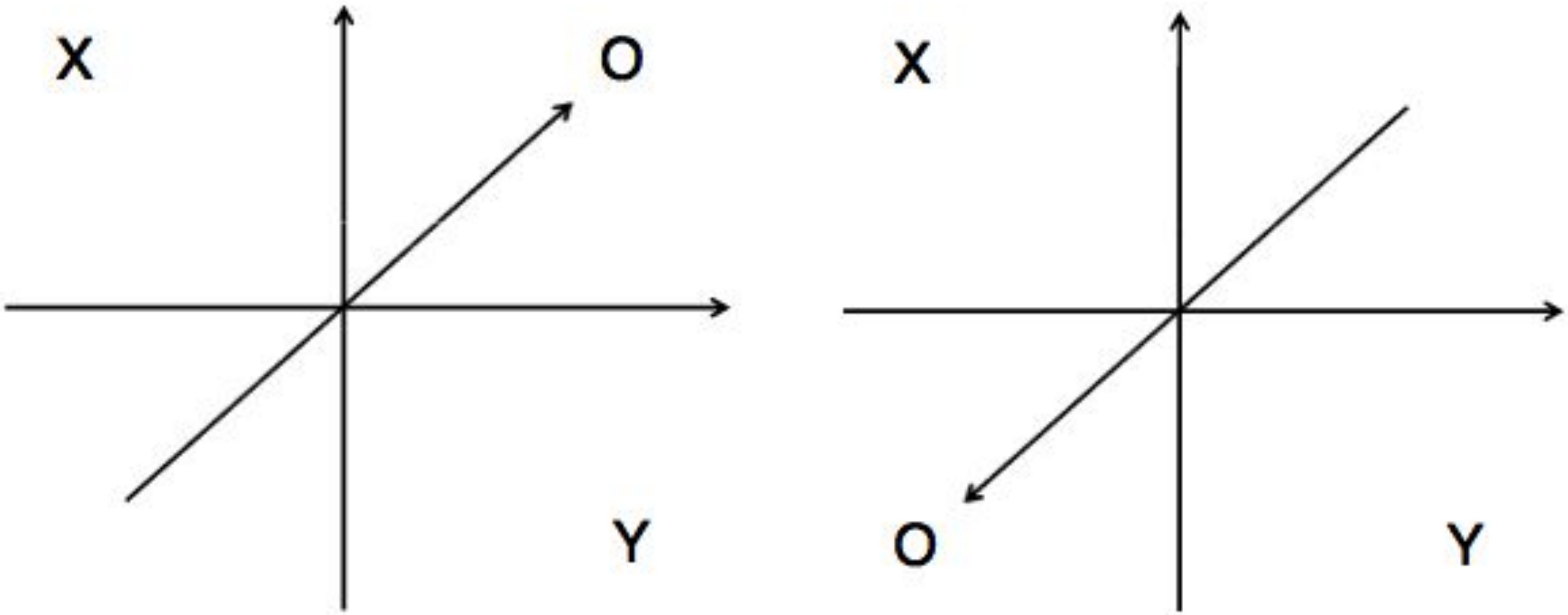
Asynchronous



Asynchronous



Synchronous



$$\mathbf{V_1 in = \sum W_{j1} V_j}$$

$$= T_{21}V_1 + T_{31}V_2 + T_{41}V_4 + T_{51}V_5$$

$$= (-2*1) + (0*1) + (0*1) + (0*1)$$

$$= -2$$

-2 is less than threshold so **V1 = 0** and our current state = **(0 1 1 1 1)**

State = (0 1 1 1 1)

V_5in = (0 0 2 -2 0) * (0 1 1 1 1) = 0

since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)

V_5in = (0 0 2 -2 0) * (0 1 1 1 1) = 0

since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)

V_2in = (-2 0 0 0 0) * (0 1 1 1 1) = 0

since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_5 in}$ = (0 0 2 -2 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_2 in}$ = (-2 0 0 0 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_4 in}$ = (0 0 -2 0 -2) * (0 1 1 1 1) = -4
since -4 < 0, V_4 = 0 *(it changed)*

State = (0 1 1 1 1)
V_5in = (0 0 2 -2 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
V_2in = (-2 0 0 0 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
V_4in = (0 0 -2 0 -2) * (0 1 1 1 1) = -4
since -4 < 0, V_4 = 0 *(it changed)*

State = (0 1 1 0 1)
V_3in = (0 0 0 -2 2) * (0 1 1 0 1) = 2
since 2 >= 0, V_3 = 1 *(it didn't change)*

State = (0 1 1 1 1)
V_5in = (0 0 2 -2 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
V_2in = (-2 0 0 0 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
V_4in = (0 0 -2 0 -2) * (0 1 1 1 1) = -4
since -4 < 0, V_4 = 0 *(it changed)*

State = (0 1 1 0 1)
V_3in = (0 0 0 -2 2) * (0 1 1 0 1) = 2
since 2 >= 0, V_3 = 1 *(it didn't change)*

State = (0 1 1 0 1)
V_1in = (0 -2 0 0 0) * (0 1 1 0 1) = -2
since -2 < 0, V_1 = 0 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_5 in}$ = (0 0 2 -2 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_2 in}$ = (-2 0 0 0 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_4 in}$ = (0 0 -2 0 -2) * (0 1 1 1 1) = -4
since -4 < 0, V_4 = 0 *(it changed)*

State = (0 1 1 0 1)
$\mathbf{V_3 in}$ = (0 0 0 -2 2) * (0 1 1 0 1) = 2
since 2 >= 0, V_3 = 1 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_1 in}$ = (0 -2 0 0 0) * (0 1 1 0 1) = -2
since -2 < 0, V_1 = 0 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_5 in}$ = (0 0 2 -2 0) * (0 1 1 0 1) = 2
since 2 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_5 in}$ = (0 0 2 -2 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_2 in}$ = (-2 0 0 0 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_4 in}$ = (0 0 -2 0 -2) * (0 1 1 1 1) = -4
since -4 < 0, V_4 = 0 *(it changed)*

State = (0 1 1 0 1)
$\mathbf{V_3 in}$ = (0 0 0 -2 2) * (0 1 1 0 1) = 2
since 2 >= 0, V_3 = 1 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_1 in}$ = (0 -2 0 0 0) * (0 1 1 0 1) = -2
since -2 < 0, V_1 = 0 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_5 in}$ = (0 0 2 -2 0) * (0 1 1 0 1) = 2
since 2 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_2 in}$ = (-2 0 0 0 0) * (0 1 1 0 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_5in}$ = (0 0 2 -2 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_2in}$ = (-2 0 0 0 0) * (0 1 1 1 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 1 1)
$\mathbf{V_4in}$ = (0 0 -2 0 -2) * (0 1 1 1 1) = -4
since -4 < 0, V_4 = 0 *(it changed)*

State = (0 1 1 0 1)
$\mathbf{V_3in}$ = (0 0 0 -2 2) * (0 1 1 0 1) = 2
since 2 >= 0, V_3 = 1 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_1in}$ = (0 -2 0 0 0) * (0 1 1 0 1) = -2
since -2 < 0, V_1 = 0 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_5in}$ = (0 0 2 -2 0) * (0 1 1 0 1) = 2
since 2 >= 0, V_5 = 1 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_2in}$ = (-2 0 0 0 0) * (0 1 1 0 1) = 0
since 0 >= 0, V_2 = 1 *(it didn't change)*

State = (0 1 1 0 1)
$\mathbf{V_4in}$ = (0 0 -2 0 -2) * (0 1 1 0 1) = -4
since -4 < 0, V_4 = 0 *(it didn't change)*

- We can stop when we cycle through all the nodes once and none of them change (thus the reason a fixed order was selected).

- The network ends up at the pattern **(0 1 1 0 1)**.

- In saying this, in this case the network will end up at a pre-learned pattern using a different fixed order of updating or if the nodes are updated randomly.

Energy Function

- Energy function [7] is derived from two-state Ising model:

$$E = -\frac{1}{2}\sum_{i \neq j}\sum T_{ij} V_i V_j \quad . \qquad [7]$$

- [8] can be derived from [7] by simple subtraction:

$$\Delta E = -\Delta V_i \sum_{j \neq i'} T_{ij} V_j \quad . \qquad [8]$$

$$\Delta E = -\Delta V_i \sum_{i \neq j} T_{ij} V_j$$

- A simple proof shows that delta E in [8] is **never > 0**.
- Consider the weighted sum describing the input to unit i:

$$\sum_{i \neq j} T_{ij} V_j$$

- **CASE 1:** The weighted sum is negative
  - Because the weighted sum is negative we know from [1] that Vi must be 0.

$$\begin{matrix} V_i \to 1 \\ V_i \to 0 \end{matrix} \quad \text{if} \quad \sum_{j \neq i} T_{ij} V_j \quad \begin{matrix} > U_i \\ < U_i \end{matrix} \quad . \qquad [1]$$

$$\Delta E = -\Delta V_i \sum_{i \neq j} T_{ij} V_j$$

- Either it changed from a one to a zero and thus Vi = -1, or it was zero and remained so and thus delta E = 0.
- In the former case, the weighted sum is negative, Vi is negative, and there is a negative term in [8] and thus delta E must be negative.

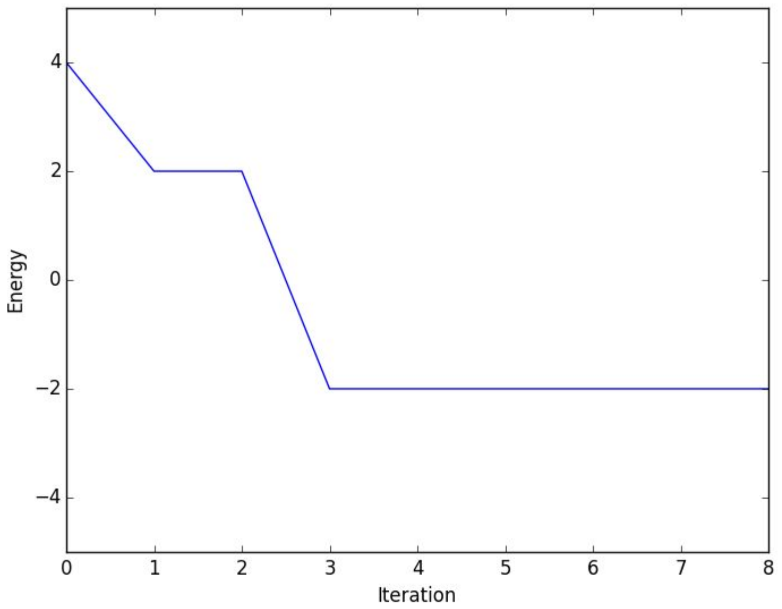
- **CASE 2:** The weighted sum is positive
  - Because the weighted sum is positive, via [1] Vi must be +1
  - Vi was either a one and remained so, and thus delta E =0, or it was 0 and changed to +1. In the latter case, delta Vi is positive, the weighted sum is positive, and we have a negative term in [8] and thus, delta E must be negative.

- Going back to our example, let's calculate the energy following each update.

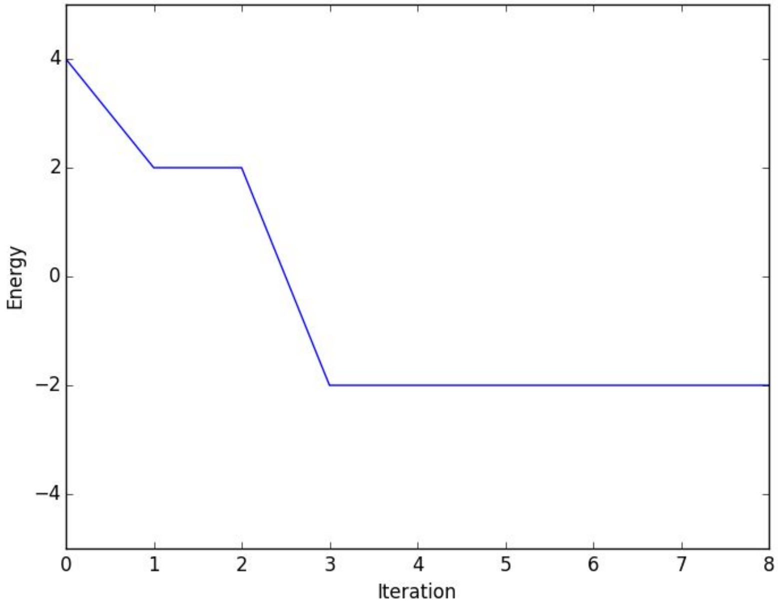
- $$(-1/2)*\left[\begin{matrix} 0 & -2 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & -2 & 0 & -2 \\ 0 & 0 & 2 & -2 & 0 \end{matrix} \quad * \quad \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{matrix}\right] =$$

$$(-1/2)[(-2*1)+(-2*1)+(-2*1)+(2*1)+(-2*1)+(-2*1)+(2*1)+(-2*1)] = 4$$

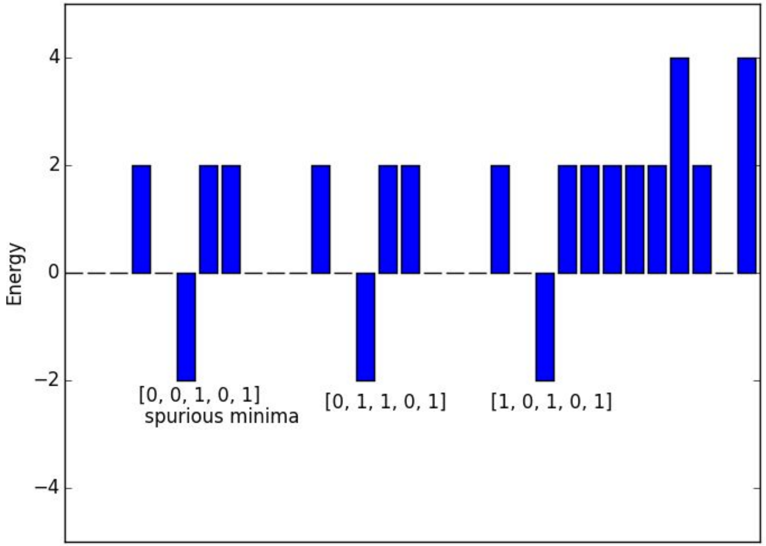
- Following this step, the state was **(0 1 1 1 1)**. For the sake of brevity: **E = 2.0**
- The final state was **(0 1 1 0 1)**. **E = -2.0 <- Minimum**



- Following this step, the state was **(0 1 1 1 1)**. For the sake of brevity: **E = 2.0**
- The final state was **(0 1 1 0 1)**. **E = -2.0 <- Minimum**

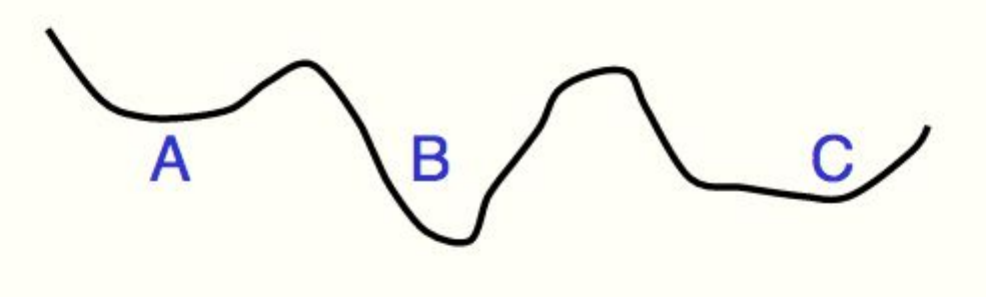


- Because our example was only five neurons (and thus 2^5 possible combinations), it is possible to represent the energy of all states.
- In doing so, you can see that the trained patterns are lowest in energy.
- Additionally, we have a spurious minima. It is worth noting this spurious minima occurs when **[0, 1, 1, 0, 1]** changes from **[0, 0, 1, 0, 1]** and thus, delta E in this case is 0.



Local Minima

- **Because a Hopfield net always makes decisions that lead to lower and lower energy, it makes it impossible to escape true local minima.**
- **In the figure below, B and C are learned patterns where A is not.**
- **If the state starts near state A and falls into A, it will be impossible with asynchronous updating to escape A and reach a trained pattern.**

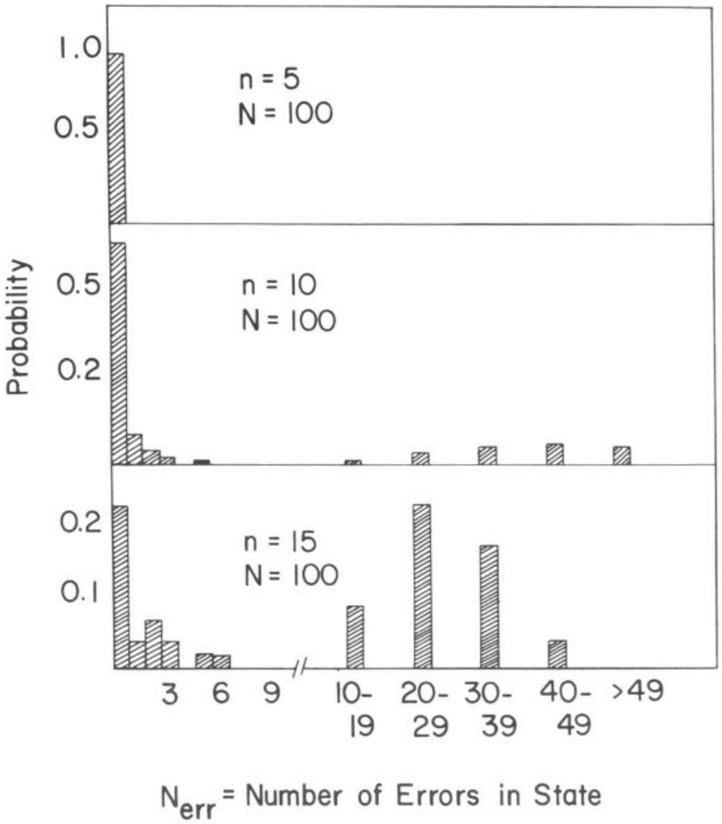


The memory capacity

- How many memories can a Hopfield network hold?

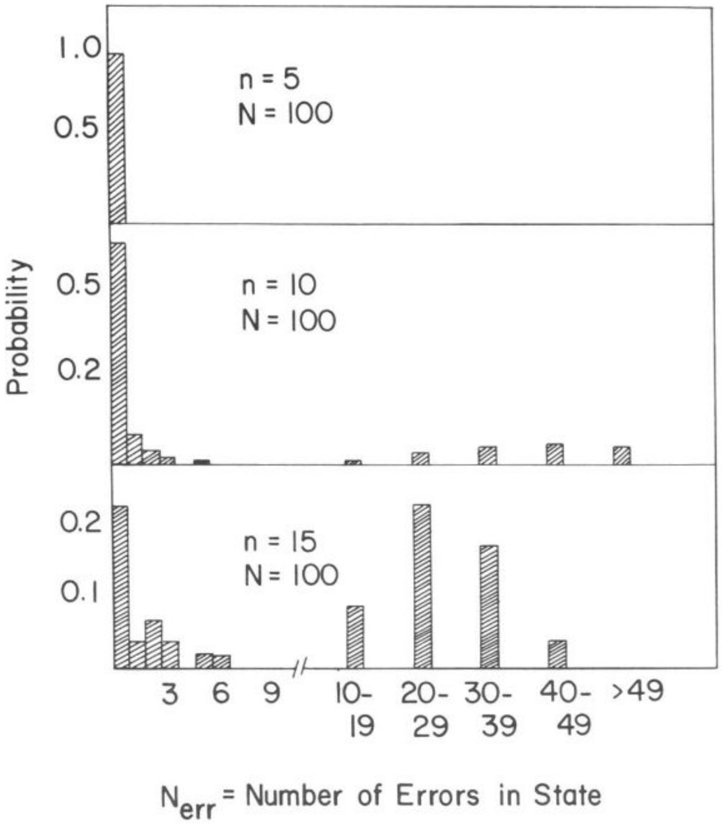
The memory capacity

- How many memories can a Hopfield network hold?
- Empirical evidence suggests 0.15 * # of neurons.



The memory capacity

- How many memories can a Hopfield network hold?
- Empirical evidence suggests 0.15 * # of neurons.
- The addition of new memories beyond the capacity overloads the network and makes all retrieval impossible.
- Clipping training weights to “forget” distant memories can prevent this from occurring.



Preventing “spurious” memories

- You can prevent two energy minima/memories from combining by “unlearning” spurious memories.
- Francis Crick proposed that this “unlearning” procedure could be what REM sleep is for; preventing spurious memory formation using random inputs from the thalamus.

[JJ Hopfield. ‘Unlearning’ has a stabilizing effect in collective memories. Nature. 1983.]

[F Crick. The function of dream sleep. Nature. 1983.]