Binary neurons and networks

Modeling the neural hardware

Grégory Dumont
gregory.dumont@ens.fr
What does the hardware look like?

Ramon y Cajal (Nobel Prize 1906)  →  neuron doctrine
Neurons = basic units of computation

(B) Retinal bipolar cell
Dendrites
Cell body
Axon

(C) Retinal ganglion cell
Dendrites
Cell body
Axon

(D) Retinal amacrine cell
Dendrites
Cell body

(E) Cortical pyramidal cell
Dendrites
Cell body
Axon

(F) Cerebellar Purkinje cells
Dendrites

Dendrites
Soma
Axon
Neurons = basic units of computation

- Dendrites
- Soma
- Axon
The Typical Cortical Neuron

- **Dendrites**: Ø 1 µm
- **Synapses**: ~ 10000
- **Soma**: 20 µm
- **Axon**: 4 cm, Ø 1 µm

To other neurons
The Typical Cortical Neuron

- Dendrites: \( \Phi 1 \mu m \)
- Synapses: \( \sim 10000 \)
- Soma: \( 20 \mu m \)
- Axon: \( 4 \text{ cm, } \Phi 1 \mu m \)

To other neurons

1 synapse \( / 0.5 \mu m \)
The Typical Cortical Neuron

- **Dendrites** (Ø 1 µm)
- **Soma** (20 µm)
- **Axon** (4 cm Ø 1 µm)
- **Synapses** (~ 10000)
- **Membrane potential**
- **Action potentials**

Diagram shows the structure of a typical cortical neuron with dendrites, soma, axon, and synapses. The diagram also illustrates the membrane potential and action potentials.
The Typical Cortical Neuron

- Dendrites (Ø 1 µm)
- Synapses (~ 10000)
- Soma (20 µm)
- Axon (4 cm, Ø 1 µm)

The synapse:
- Presynaptic neuron
- Postsynaptic neuron
- Synaptic cleft
- Vesicles

Action potentials
Membrane potential

To other neurons
The Typical Cortical Neuron

- **Dendrites** (Ø 1 µm)
- **Synapses** (~ 10000)
- **Soma** (20 µm)
- **Axon** (4 cm Ø 1 µm)

**Presynaptic spike**

**Synaptic cleft**

**Vesicles**

**Postsynaptic neuron**

**Action potentials**

**Membrane potential**

**To other neurons**

**The synapse**

**The Typical Cortical Neuron**

**Action potentials**

**Membrane potential**

**To other neurons**

**The synapse**

**Synaptic cleft**

**Vesicles**

**Postsynaptic neuron**

**Tuesday, April 1, 14**
The course

• Description of neural dynamics
• Biophysics of membrane potential dynamics and synaptic transmission
• Dynamics of networks of neurons

**TODAY:** simplest possible description of neurons and networks
The Binary Neuron

\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) \]

McCulloch and Pitts (1943)
What can binary neurons compute?

Inputs: 0 or 1

Outputs: 0 or 1

McCulloch and Pitts (1943)  Neuron = Basic logical unit
Brain = digital machine

Historical perspective:
1930-1950: birth of first computers

Shannon: information theory of digital signals

Turing: universal capabilities of digital machines

Von Neumann: architecture of universal computers

Can we construct an electronic brain?
Birth of Artificial Intelligence
The Binary Neuron

McCulloch and Pitts (1943)
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

Two synaptic inputs

\[ \vec{x} = (x_1, x_2) \quad \vec{w} = (w_1, w_2) \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

Two synaptic inputs

\[ \vec{x} = (x_1, x_2) \quad \vec{w} = (w_1, w_2) \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

Two synaptic inputs

\[ \vec{x} = (x_1, x_2) \quad \vec{w} = (w_1, w_2) \]

\[ \vec{w} \cdot \vec{x} - b = 0 \]
\[
y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b)
\]

Two synaptic inputs

\[
\vec{x} = (x_1, x_2) \quad \vec{w} = (w_1, w_2)
\]

\[
\vec{w} \cdot \vec{x} - b = 0
\]
Two synaptic inputs

\[ \vec{w} \cdot \vec{x} - b = 0 \]

Separates the plane in two regions
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

Three synaptic inputs \( \vec{x} = (x_1, x_2, x_3) \) \( \vec{w} = (w_1, w_2, w_3) \)
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

Three synaptic inputs

\[ \vec{x} = (x_1, x_2, x_3) \quad \vec{w} = (w_1, w_2, w_3) \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\mathbf{w} \cdot \mathbf{x} - b) \]

Three synaptic inputs

\[ \mathbf{x} = (x_1, x_2, x_3) \quad \mathbf{w} = (w_1, w_2, w_3) \]

\[ \mathbf{w} \cdot \mathbf{x} - b = 0 \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

Three synaptic inputs

\[ \vec{x} = (x_1, x_2, x_3) \quad \vec{w} = (w_1, w_2, w_3) \]

\[ \vec{w} \cdot \vec{x} - b = 0 \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\bar{w} \cdot \bar{x} - b) \]

Three synaptic inputs

\[ \bar{x} = (x_1, x_2, x_3) \quad \bar{w} = (w_1, w_2, w_3) \]

\[ \bar{w} \cdot \bar{x} - b = 0 \]

\[ y = 1 \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} . \vec{x} - b) \]

Three synaptic inputs

\[ \vec{x} = (x_1, x_2, x_3) \quad \vec{w} = (w_1, w_2, w_3) \]

\[ \vec{w} . \vec{x} - b = 0 \]

\[ y = 1 \]

\[ y = 0 \]

Separates the space in two regions
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

N synaptic inputs \[ \vec{x} = (x_1, \ldots, x_N) \quad \vec{w} = (w_1, \ldots, w_N) \]

\[ \vec{w} \cdot \vec{x} - b = 0 \]
\[ y = H \left( \sum_{k=1}^{N} w_k x_k - b \right) = H (\vec{w} \cdot \vec{x} - b) \]

\( \vec{w} \) has \( N \) synaptic inputs
\( \vec{x} = (x_1, \ldots, x_N) \)
\( \vec{w} = (w_1, \ldots, w_N) \)

\[ \vec{w} \cdot \vec{x} - b = 0 \]
defines a hyperplane

Separates the space in two regions

binary classifier
Binary classification task
Motion discrimination task

0% coherence  50% coherence  100% coherence
Responses of a single neuron

B

Opposite Direction

coherence = 12.8%

Preferred Direction

coherence = 3.2%

coherence = 0.8%

number of trials

firing rate (Hz)
Activity of neurons in MT

Can an upstream neuron read out the motion direction?
Neural readout of motion direction

\[ \text{Output} = \begin{cases} 1 \text{ if right} \\ 0 \text{ if left} \end{cases} \]
Neural readout of motion direction

\[
\text{Output} = \begin{cases} 
1 & \text{if right} \\
0 & \text{if left}
\end{cases}
\]

Correct readout!
Neural readout of motion direction

\[
\text{Output} = \begin{cases} 
1 & \text{if right} \\
0 & \text{if left} 
\end{cases}
\]
Binary neuron = linear classifier

Need to adjust synaptic weights!!

Learning = modification of synaptic weights
Learning in the Binary Neuron

\[ x_1 \, x_2 \, x_3 \ldots x_n \]

Synaptic Inputs

\[ w_1 \, w_2 \, w_3 \, w_n \]

Synaptic weights

\[ \sum \]

Summation

\[ \text{Threshold} \]

Output

\[ y \]
Learning in the Binary Neuron

given

\[
x_1, x_2, x_3, \ldots, x_n
\]

Synaptic Inputs

\[
w_1, w_2, w_3, \ldots, w_n
\]

Synaptic weights

\[
\sum
\]

Summation

\[
\text{Threshold}
\]

Output

\[
y
\]
Learning in the Binary Neuron

Synaptic Inputs
Synaptic weights
Summation
Threshold
Output

given

\[
x_1 \quad x_2 \quad x_3 \ldots x_n
\]

given

\[
w_1 \quad w_2 \quad w_3 \quad w_n
\]

\[
y
\]
Learning in the Binary Neuron

Given: $x_1, x_2, x_3, \ldots, x_n$ (Synaptic Inputs)

Synaptic weights: $w_1, w_2, w_3, w_n$

Summation: $\sum$

Threshold: $\leq$

Output: $y$

Synaptic plasticity underlies learning
Exemple: classify red points as 1 and blue points as 0
Exemple: classify red points as 1 and blue points as 0
Exemple: classify red points as 1 and blue points as 0
Exemple: classify red points as 1 and blue points as 0
Exemple: classify red points as 1 and blue points as 0
Example: classify red points as 1 and blue points as 0.
Exemple: classify red points as 1 and blue points as 0
Exemple: classify red points as 1 and blue points as 0.
The perceptron learning rule

Rosenblatt (1958)

Training set of p patterns: \[ \{(x^{(0)}, d_0), (x^{(1)}, d_1) \ldots (x^{(p)}, d_p)\} \]

where \( x^{(k)} \) is an input vector

\[ d_k = 0 \text{ or } 1 \] is a desired output
The perceptron learning rule

Rosenblatt (1958)

Training set of p patterns: \( \{(x^{(0)}, d_0), (x^{(1)}, d_1), \ldots (x^{(p)}, d_p)\} \)

where \( x^{(k)} \) is an input vector
\( d_k = 0 \) or \( 1 \) is a desired output

On every step:
for each pattern \( k \)

1. compute the output
\( y_k = H(\sum_{i=1}^{N} w_i x^{(k)}_i) \)

2. if \( y_k \neq d_k \) update the weights:
\( w_i(t + 1) = w_i(t) + (d_k - y_k)x^{(k)}_i \)

Converges in a finite number of steps if a solution exists
The perceptron learning rule

Aim: classify red points as 1 and blue points as 0
The perceptron learning rule

Start with a random set of synaptic weights
The perceptron learning rule

Choose a misclassified pattern
The perceptron learning rule

Choose a misclassified pattern
The perceptron learning rule

Update weight vector

\[ \vec{w} = \vec{w} + \vec{x}^{(1)} \]
The perceptron learning rule

Update weight vector

$$\vec{w} = \vec{w} + \vec{x}^{(1)}$$
The perceptron learning rule

Update weight vector

\[ \vec{w} = \vec{w} + \vec{x}^{(1)} \]
The perceptron learning rule

Update weight vector
The perceptron learning rule

Choose next misclassified pattern
The perceptron learning rule

Choose next misclassified pattern

\[ \vec{x}^{(2)} \]
The perceptron learning rule

Update weight vector

$$\vec{w} = \vec{w} + \vec{x}^{(2)}$$
The perceptron learning rule

Correct classification: learning terminates
What can a perceptron do?

Rosenblatt: « The perceptron may eventually be able to learn, make decisions, and translate languages »
What can a perceptron do?

Rosenblatt: « The perceptron may eventually be able to learn, make decisions, and translate languages »

Exemple: train neurons to recognize hand-written digits
What can a perceptron do?

Rosenblatt: « The perceptron may eventually be able to learn, make decisions, and translate languages »

Exemple: train neurons to recognize hand-written digits

Train ten binary neurons

Inputs: vector of pixel values corresponding to digits

Neuron 3: output = 1 if input is the digit 3, output = 0 otherwise
The perceptron

Frank Rosenblatt
Can the perceptron perform universal binary computations?

Binary inputs \[ x_i = 0 \text{ or } 1 \]

Can any function \( f : \{0, 1\}^n \to \{0, 1\} \) be implemented by a perceptron?
Implementing binary functions with a perceptron

N=2, only 4 possible input patterns

Example: \( y = \text{AND}(x_1, x_2) \)
Implementing binary functions with a perceptron

n=2, only 4 possible patterns

Example: \( y = \text{AND}(x_1, x_2) \)
Implementing binary functions with a perceptron

$n=2$, only 4 possible patterns

Example: $y = \text{OR}(x_1, x_2)$
Implementing binary functions with a perceptron

n=2, only 4 possible patterns

Example: y = OR(x1, x2)
Implementing binary functions with a perceptron

n=2, only 4 possible patterns

Example: $y = \text{XOR}(x1, x2)$
Implementing binary functions with a perceptron

n=2, only 4 possible patterns

Example: \( y = \text{XOR}(x_1, x_2) \)

IMPOSSIBLE!!!
A binary neuron can only implement linearly separable functions

Two sets are linearly separable if there exists a hyperplane separating them

A binary neuron can only implement linearly separable functions

Marvin Minsky and Seymour Papert
*Perceptrons* (1969)

→ AI winter: halt in research and funding during 10 years
What can binary neurons compute?

Single binary neurons can compute only linearly separable functions
What can feedforward networks compute?
What can feedforward networks compute?

Single layer networks can compute only linearly separable functions
What can multilayer feedforward networks compute?
Multilayer networks can compute an XOR
Multilayer networks can compute an XOR

Diagram: 

- OR
- AND
- NON-AND
Multilayer networks can compute an XOR
Multilayer networks can compute any binary function!

Any binary function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \)

can be represented using only ANDs and ORs

[disjunctive normal form and conjunctive normal form]

→ Multilayer networks have universal computational properties
Multilayer networks can compute any binary function!

Any binary function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented using only ANDs and ORs

[disjunctive normal form and conjunctive normal form]

→ Multilayer networks have universal computational properties

... but how to train them?
Training multilayer networks

Set of $p$ training patterns $\{(x^{(0)}, d_0), (x^{(1)}, d_1), \ldots (x^{(p)}, d_p)\}$

Aim: minimize cost function $E = \sum_{k=1}^{p} ||y_k - d_k||^2$

by changing the synaptic weights in the network

→ Backpropagation algorithm (Rumelhart, Hinton and Williams 1986)

David Rumelhart

Geoff Hinton
Backpropagation

Renaissance of Artificial Neural Networks since the 80’s

but

requires retrograde propagation along axons and synapses

→ Not compatible with biology
Hebb’s postulate

When an axon of cell A is near enough to excite cell B and 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 efficiency, as one of the cells firing B, is increased. (1949)

If A and B are active at the same time, \( w_{A \rightarrow B} \) increases.
The perceptron learning rule

Rosenblatt (1958)

Training set of $p$ patterns: \[ \{(x^{(0)}, d_0), (x^{(1)}, d_1) \ldots (x^{(p)}, d_p)\} \]

where $x^{(k)}$ is an input vector
$d_k = 0 \text{ or } 1$ is a desired output

On every step:
for each pattern $k$

1. compute the output $y_k = H(\sum_{i=1}^{N} w_i x^{(k)}_i)$

2. if $y_k \neq d_k$ update the weights:

\[ w_i(t + 1) = w_i(t) + (d_k - y_k)x^{(k)}_i \]

--- Input x Output

--- hebbian learning

Tuesday, April 1, 14
So far: feedforward networks
More general: recurrent connections

Input

Hidden units

Output

$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \ldots \quad \mathbf{x}_n$

$\mathbf{y}_1 \quad \mathbf{y}_2$
More general: recurrent connections
More general: recurrent connections
Recurrent networks: need to look at the dynamics!

\[ y_1 \quad y_2 \]

Input

Hidden units

Output

\[ x_1 \quad x_2 \quad x_3 \ldots \quad x_n \]
Recurrent networks: need to look at the dynamics!

\[ x_1, x_2, x_3 \ldots, x_N \]

\[ y_1, y_2, y_3 \ldots, y_N \]
Network dynamics in discrete time

Network of N units

\[ y_i(t + 1) = H \left[ \sum_{j=1}^{N} w_{ij} y_j(t) + x_i(t) \right] \]
Network dynamics in discrete time

Network of $N$ units

$$y_i(t + 1) = H \left[ \sum_{j=1}^{N} w_{ij} y_j(t) + x_i(t) \right]$$

Activity of neuron $i$ at next timestep
Network dynamics in discrete time

Network of N units

\[ y_i(t + 1) = H\left[ \sum_{j=1}^{N} w_{ij} y_j(t) + x_i(t) \right] \]

- Activity of neuron \( i \) at next timestep
- Total input from network at present timestep
- External input
Network dynamics in discrete time

Network of N units

\[ y_i(t + 1) = H\left[ \sum_{j=1}^{N} w_{ij} y_j(t) + x_i(t) \right] \]
Network dynamics in discrete time

Network of N units

\[ y_i(t + 1) = H \left[ \sum_{j=1}^{N} w_{ij} y_j(t) + x_i(t) \right] \]

Activity of neuron I at next timestep

Total input from network at present timestep

External input

Change of notations:

\[ y_i = \begin{cases} 1 & \text{active} \\ -1 & \text{inactive} \end{cases} \]

\[ H[x] = \text{sgn}[x] \]
Example

Network of $N = 3$ units

activity: 

$$\vec{y}(t) = (y_1(t), y_2(t), y_3(t))$$

synaptic matrix: 

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

dynamics: 

$$\vec{y}(t + 1) = sgn \left[ W . \vec{y}(t) \right]$$
Example

Network of $N = 3$ units

initial condition: $\vec{y}(t = 1) = (1, 1, 1)$

synaptic matrix: $W = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$

dynamics: $\vec{y}(t + 1) = sgn \left[ W.\vec{y}(t) \right]$
Dynamics - first time step

\[ \bar{y}(t + 1) = sgn [W.\bar{y}(t)] \]
Dynamics - first time step

\[ \bar{y}(t + 1) = sgn \left[ W.\bar{y}(t) \right] \]

\[
W.\bar{y}(1) = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1 \\
\end{pmatrix}
\begin{pmatrix}
1 \\
1 \\
1 \\
\end{pmatrix} = \begin{pmatrix}
1 \\
-1 \\
1 \\
\end{pmatrix}
\]
Dynamics - first time step

\[ \tilde{y}(t + 1) = sgn \left[ W.\tilde{y}(t) \right] \]

\[
W.\tilde{y}(1) = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1
\end{pmatrix}
\begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
= \begin{pmatrix}
1 \\
-1 \\
1
\end{pmatrix}
\]

\[ \tilde{y}(2) = sgn \left( \begin{pmatrix}
1 \\
-1 \\
1
\end{pmatrix} \right) = \begin{pmatrix}
1 \\
-1 \\
1
\end{pmatrix} \]
Dynamics - second time step

\[ \tilde{y}(t + 1) = sgn [W.\tilde{y}(t)] \]

\[ W.\tilde{y}(2) = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \]
Dynamics - second time step

\[ \bar{y}(t + 1) = sgn \left[ W.\bar{y}(t) \right] \]

\[
W.\bar{y}(2) = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1 \\
1 & -1 & 1
\end{pmatrix}
\begin{pmatrix}
1 \\
-1 \\
1 \\
1
\end{pmatrix} = \begin{pmatrix}
1 \\
-1 \\
1 \\
1
\end{pmatrix}
\]

\[
\bar{y}(3) = sgn \begin{pmatrix}
1 \\
-1 \\
1 \\
1
\end{pmatrix} = \begin{pmatrix}
1 \\
-1 \\
1 \\
1
\end{pmatrix}
\]
Dynamics - second time step

\[ \bar{y}(t + 1) = \text{sgn} \left[ W.\bar{y}(t) \right] \]

\[
W.\bar{y}(2) = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}
\]

\[
\bar{y}(3) = \text{sgn} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \bar{y}(2)!
\]

The activity does not evolve anymore = fixed point
Fixed points of network dynamics

External input only on first step = set initial conditions

Dynamics stop when

\[ y_i(t + 1) = y_i(t) \]
Fixed points of network dynamics

External input only on first step = set initial conditions

Dynamics stop when

$$y_i(t + 1) = y_i(t)$$

$$y_i = \text{sgn} \left( \sum_{j=1}^{N} w_{ij} y_j \right)$$

fixed point = output of the network
Start from different initial condition

Network of $N = 3$ units

initial condition:
$$\vec{y}(t = 1) = (1, -1, -1)$$

synaptic matrix:
$$W = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1
\end{pmatrix}$$

dynamics:
$$\vec{y}(t + 1) = sgn \left[ W.\vec{y}(t) \right]$$
Dynamics - first time step

\[ \vec{y}(t + 1) = sgn [W.\vec{y}(t)] \]

\[
W.\vec{y}(1) = \begin{pmatrix}
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1 \\
\end{pmatrix}
\begin{pmatrix}
1 \\
-1 \\
-1 \\
\end{pmatrix}
= \begin{pmatrix}
1 \\
-1 \\
1 \\
\end{pmatrix}
\]

\[ \vec{y}(2) = sgn \begin{pmatrix}
1 \\
-1 \\
1 \\
\end{pmatrix}
= \begin{pmatrix}
1 \\
-1 \\
1 \\
\end{pmatrix}
\]

same fixed point!
Attractors

Given an input (=initial condition),
the network dynamics will evolve to the closest fixed point.
Attractors

Given an input (=initial condition),
the network dynamics will evolve to the closest fixed point.

Different initial conditions can lead to the same fixed point.
Attractors

Given an input (=initial condition),
the network dynamics will evolve to the closest fixed point.

Different initial conditions can lead to the same fixed point.

→ fixed points = attractors for the dynamics
Attractors

Given an input (=initial condition),
the network dynamics will evolve to the closest fixed point.

Different initial conditions can lead to the same fixed point.

$\rightarrow$ **fixed points = attractors for the dynamics**

**attractor neural networks:**
**store patterns as fixed points = memories**
Attractors

Given an input (=initial condition),
the network dynamics will evolve to the closest fixed point.

Different initial conditions can lead to the same fixed point.

fixed points = attractors for the dynamics

attractor neural networks:
store patterns as fixed points = memories

Fixed points depend on synaptic weights.

how to set weights to encode desired patterns?
Hopfield learning rule for recurrent networks

Set of \( p \) desired outcomes:

\[ \{ \xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(p)} \} \]

Set weights to (Hopfield 1982):

\[ w_{ij} = \frac{1}{N} \sum_{k=1}^{p} \xi_i^{(k)} \xi_j^{(k)} \]

Pseudo-hebbian rule
Symmetric connections: the network possesses an energy function
Network dynamics minimize the energy function

\[ \text{Stored patterns are located at the minima of the energy function} \]
Exemple: network of n=100 neurons

Every box represents a neuron
All neurons are interconnected

\[ y_i = \begin{cases} 1 & \text{Black} \\ -1 & \text{White} \end{cases} \]
Exemple of learning in a recurrent network

Desired output: $\xi^{(1)}$:

Synaptic weights:

$$w_{ij} = \frac{1}{N} \xi_{i}^{(1)} \xi_{j}^{(1)}$$
Exemple of learning in a recurrent network

Desired output: $\xi^{(1)}$

Synaptic weights:

$$w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)}$$
Exemple of learning in a recurrent network

Desired output: $\xi^{(1)}$

Synaptic weights:

$$ w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)} $$
Exemple of learning in a recurrent network

Desired output

\[ \xi^{(1)} : \]

Synaptic weights:

\[ w_{ij} = \frac{1}{N} \xi^{(1)}_i \xi^{(1)}_j \]

\[ w = -1 \]
\( \xi^{(1)} \) is a fixed point of the dynamics

Fixed point equation:

\[
y_i = \text{sgn}\left[\sum_{j=1}^{n} w_{ij} y_j(t)\right]
\]

with:

\[
w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)}
\]
$\xi^{(1)}$ is a fixed point of the dynamics

Fixed point equation:

$$y_i = \text{sgn}\left[\sum_{j=1}^{n} w_{ij} y_j(t)\right]$$

with:

$$w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)}$$
\( \xi^{(1)} \) is a fixed point of the dynamics

Fixed point equation:

\[
y_i = \text{sgn}\left[\sum_{j=1}^{n} w_{ij} y_j(t)\right]
\]

with:

\[
w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)}
\]

\[
y_i = \xi_i = \text{sgn}\left(\sum_{j=1}^{N} w_{ij} \xi_j\right)
= \text{sgn}\left(\sum_{j=1}^{N} \frac{\xi_i \xi_j}{N} \xi_j\right)
\]
\( \xi^{(1)} \) is a fixed point of the dynamics

Fixed point equation:

\[
y_i = \text{sgn} \left[ \sum_{j=1}^{n} w_{ij} y_j(t) \right]
\]

with:

\[
 w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)}
\]

\[
y_i = \xi_i
\]

\[
= \text{sgn} \left( \sum_{j=1}^{N} w_{ij} \xi_j \right)
\]

\[
= \text{sgn} \left( \sum_{j=1}^{N} \frac{\xi_i \xi_j}{N} \xi_j \right)
\]

\[
= \text{sgn} \left( \frac{\xi_i}{N} \sum_{j=1}^{N} \xi_j \xi_j \right)
\]
\( \xi^{(1)} \) is a fixed point of the dynamics

Fixed point equation:

\[
y_i = \text{sgn}\left[ \sum_{j=1}^{n} w_{ij} y_j(t) \right] = y_i = \xi_i
\]

with:

\[
wx = \frac{1}{N} \xi^{(1)}_i \xi^{(1)}_j
\]
\( \dot{\xi}^{(1)} \) is a fixed point of the dynamics

Fixed point equation:

\[
y_i = \text{sgn}\left[ \sum_{j=1}^{n} w_{ij} y_j(t) \right]
\]

with:

\[
w_{ij} = \frac{1}{N} \xi_i^{(1)} \xi_j^{(1)}
\]

\[
y_i = \xi_i
\]

\[
= \text{sgn}\left( \sum_{j=1}^{N} w_{ij} \xi_j \right)
\]

\[
= \text{sgn}\left( \sum_{j=1}^{N} \frac{\xi_i \xi_j}{N} \xi_j \right)
\]

\[
= \text{sgn}\left( \frac{\xi_i}{N} \sum_{j=1}^{N} \xi_j \xi_j \right)
\]

\[
= \text{sgn}\left( \frac{\xi_i}{N} N \right)
\]

\[
= \text{sgn}(\xi_i)
\]

\[
= \xi_i
\]
Exemple of learning in a recurrent network

Pattern is stored in the network and recovered from suitable initial conditions
Content-addressable, associative memory
Content-addressable, associative memory
Inverse pattern is stored too!
Storing two patterns

\[ w_{ij} = \frac{\xi^{(1)}_i \xi^{(1)}_j + \eta^{(1)}_i \eta^{(1)}_j}{N} \]
Storing two patterns

\[
\text{sgn} \left( \sum_{j=1}^{N} w_{ij} \xi_j \right)
\]

\[
= \text{sgn} \left( \sum_{j=1}^{N} \frac{\xi_i \xi_j + \eta_i \eta_j}{N} \xi_j \right)
\]
Storing two patterns

\[
\text{sgn} \left( \sum_{j=1}^{N} w_{ij} \xi_j \right) \\
= \text{sgn} \left( \sum_{j=1}^{N} \frac{\xi_i \xi_j + \eta_i \eta_j}{N} \xi_j \right) \\
= \text{sgn} \left( \frac{\xi_i}{N} \sum_{j=1}^{N} \xi_j \xi_j + \frac{\eta_i}{N} \sum_{j=1}^{N} \eta_j \xi_j \right)
\]
Storing two patterns

\[
\text{sgn}\left(\sum_{j=1}^{N} w_{ij} \xi_j\right)
\]

\[
= \text{sgn}\left(\sum_{j=1}^{N} \frac{\xi_i \xi_j + \eta_i \eta_j}{N} \xi_j\right)
\]

\[
= \text{sgn}\left(\frac{\xi_i}{N} \sum_{j=1}^{N} \xi_j \xi_j + \frac{\eta_i}{N} \sum_{j=1}^{N} \eta_j \xi_j\right)
\]

"Cross-talk" term

\[
C = \frac{1}{N} \sum_{j=1}^{N} \eta_j \xi_j
\]

\[-1 \leq C \leq 1\]

\[\rightarrow\text{Overlap between patterns}\]
Storing two patterns

\[
\text{sgn} \left( \sum_{j=1}^{N} w_{ij} \xi_j \right) \\
= \text{sgn} \left( \sum_{j=1}^{N} \frac{\xi_i \xi_j + \eta_i \eta_j}{N} \xi_j \right) \\
= \text{sgn} \left( \frac{\xi_i}{N} \sum_{j=1}^{N} \xi_j \xi_j + \frac{\eta_i}{N} \sum_{j=1}^{N} \eta_j \xi_j \right) \\
= \text{sgn} \left( \xi_i + \frac{\eta_i}{N} \sum_{j=1}^{N} \eta_j \xi_j \right) \\
= \text{sgn} \left( \xi_i + \eta_i C \right) \\
= \xi_i
\]

\(-1 \leq C \leq 1\)
Storing three patterns

\[ \xi_i = \text{sgn} \left( \sum_{j=1}^{N} w_{ij} \xi_j \right) \]

\[ = \text{sgn} \left( \xi_i + \eta_i C_1 + \gamma_i C_2 \right) \]

\[ \rightarrow \text{More “cross-talk” !!} \]
Limitations of Hopfield networks

- inverse patterns are always stored
- if three or more patterns are stored, spurious memories can appear
- the number of random patterns that can be stored is $p=0.138N$
- catastrophe: if the capacity is reached, all patterns are erased
- biologically unrealistic (symmetry between -1 and +1, no separation between excitation and inhibition, infinite number of synaptic states)
Binary neurons and networks: summary

- Binary neurons act as binary linear classifiers
- Learning rules can be used to train the neurons to produce the desired output
- Single layer feedforward networks can compute linearly separable operations [PERCEPTRON – HEBBIAN learning rule]
- Multilayer feedforward networks can compute any binary function
- Recurrent networks can memorize and recall patterns [ATTRACTOR networks]
Outlook

BINARY NEURONS AND NETWORKS

ARTIFICIAL NEURAL NETWORKS
aim: solve machine-learning problems

NEUROSCIENCE
aim: understand how the brain works

→ inspired by biology

→ constrained by biology