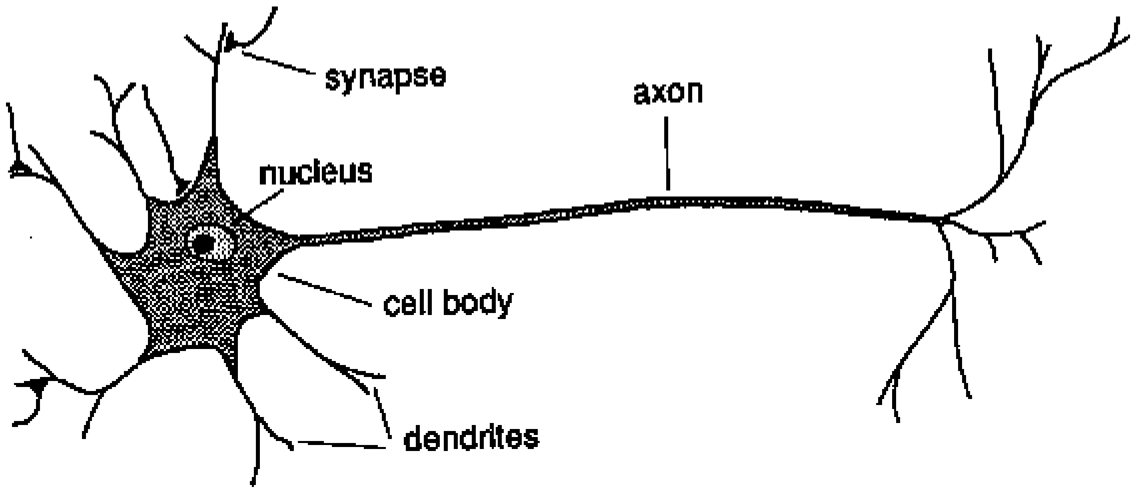
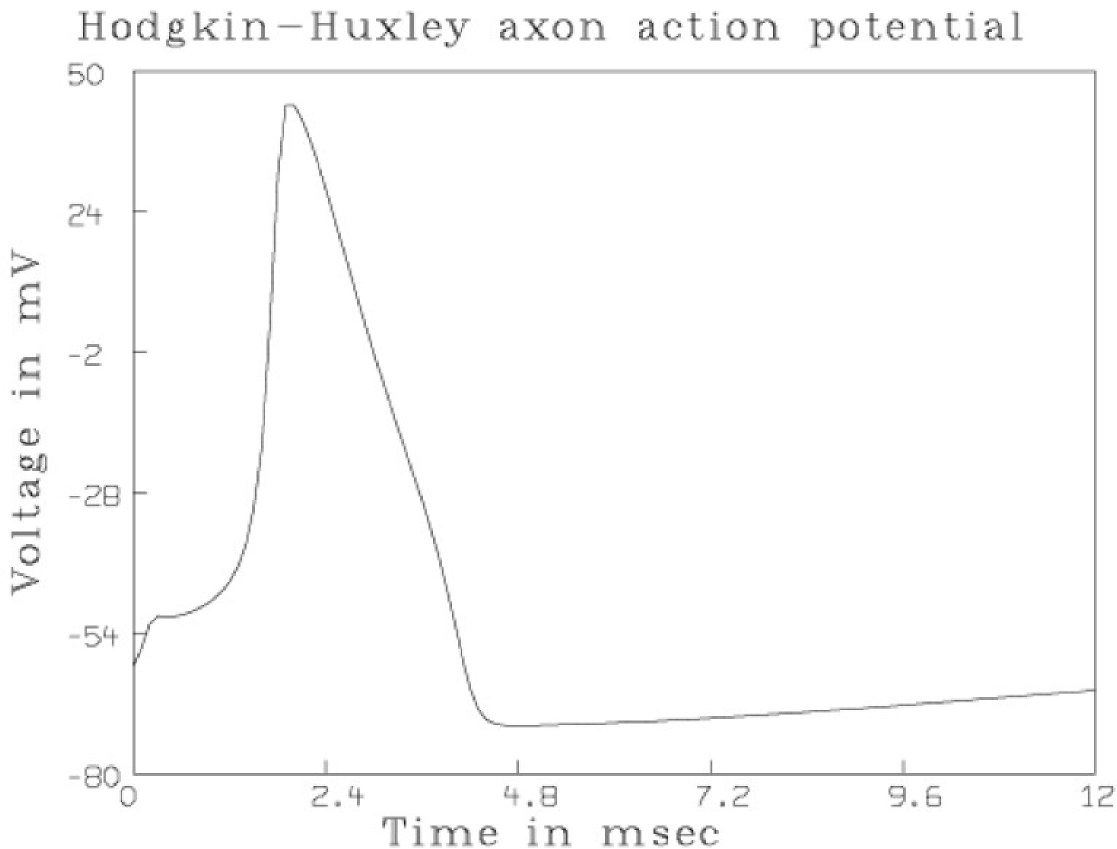
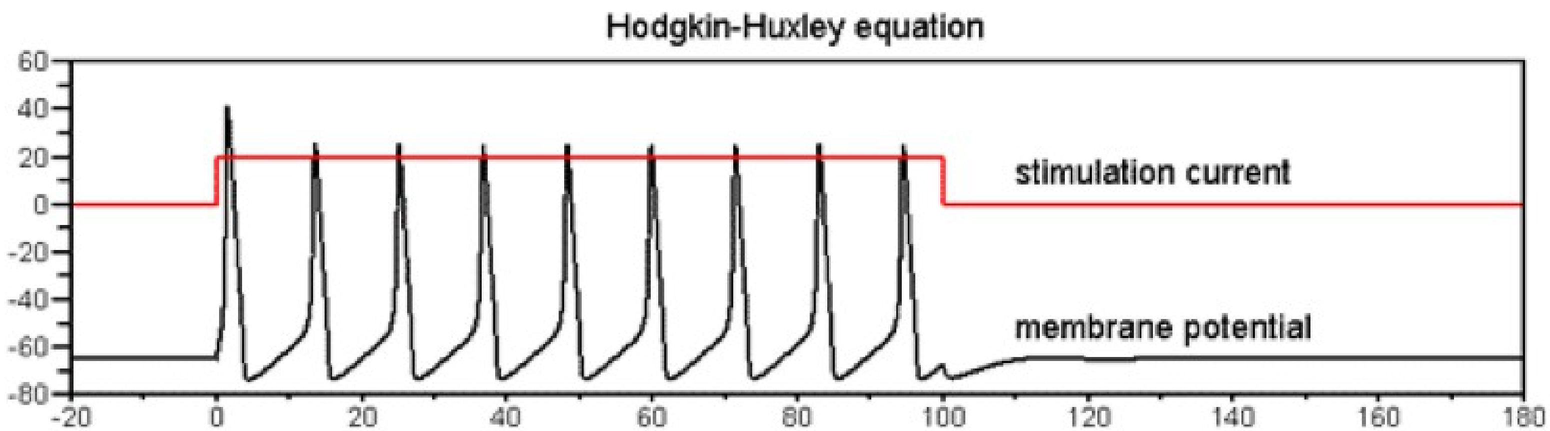
Neurons and Brains



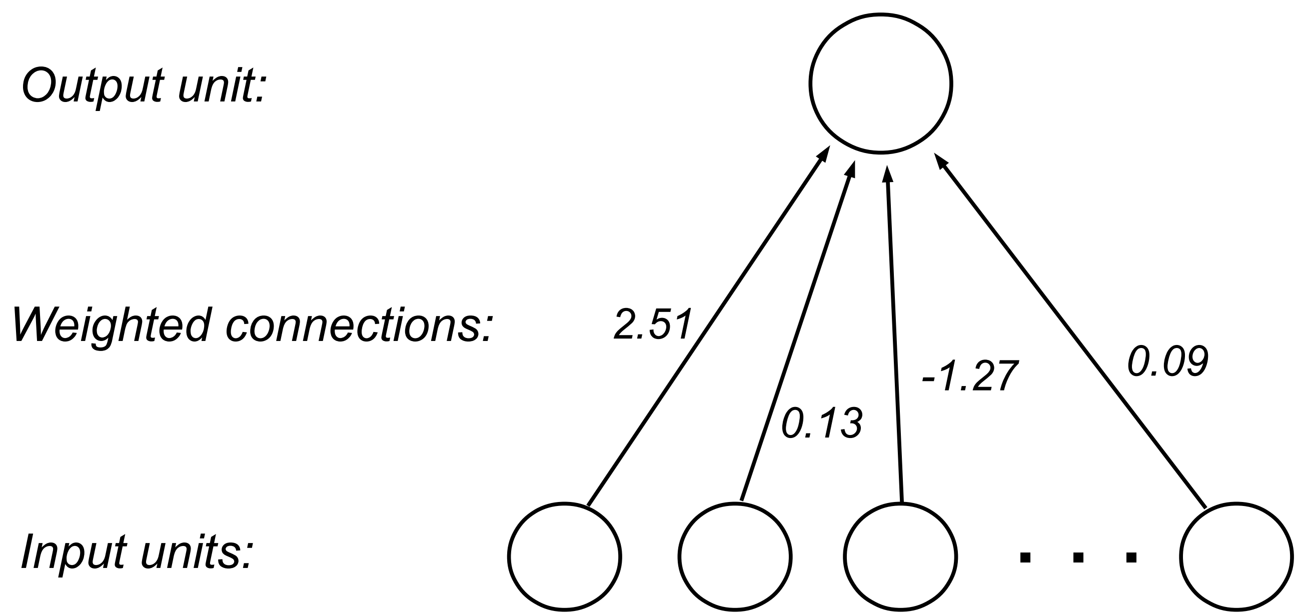
- Your brain has ~ 100 billion neurons
- Each neuron has ~ 10,000 synaptic connections to other neurons
- Hundreds of trillions of connections
- Learning induces changes in the connection strengths between neurons

Hodgkin-Huxley Neuron Model



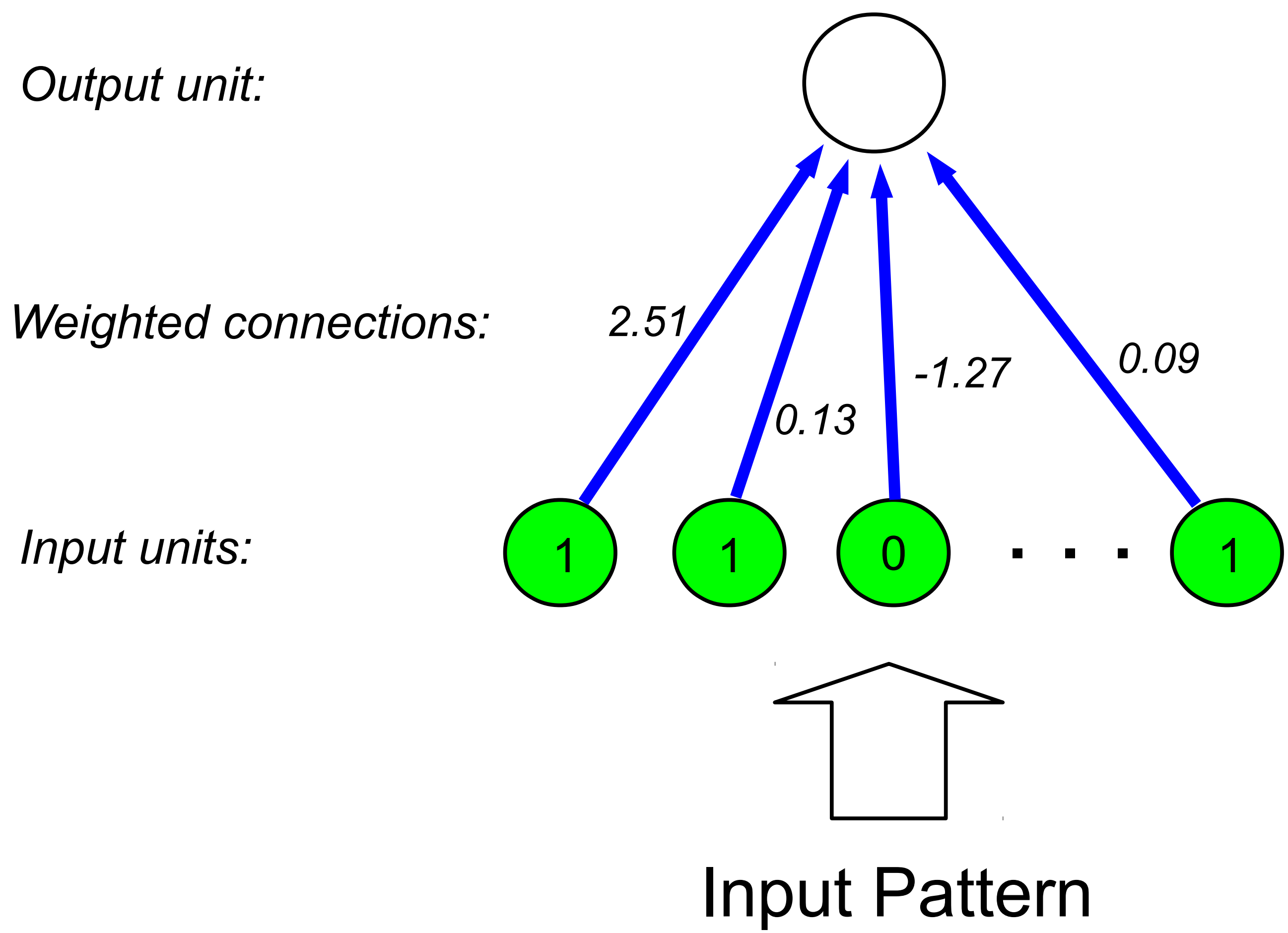


Artificial Neurons: Binary Version



Artificial Neurons: Binary Version

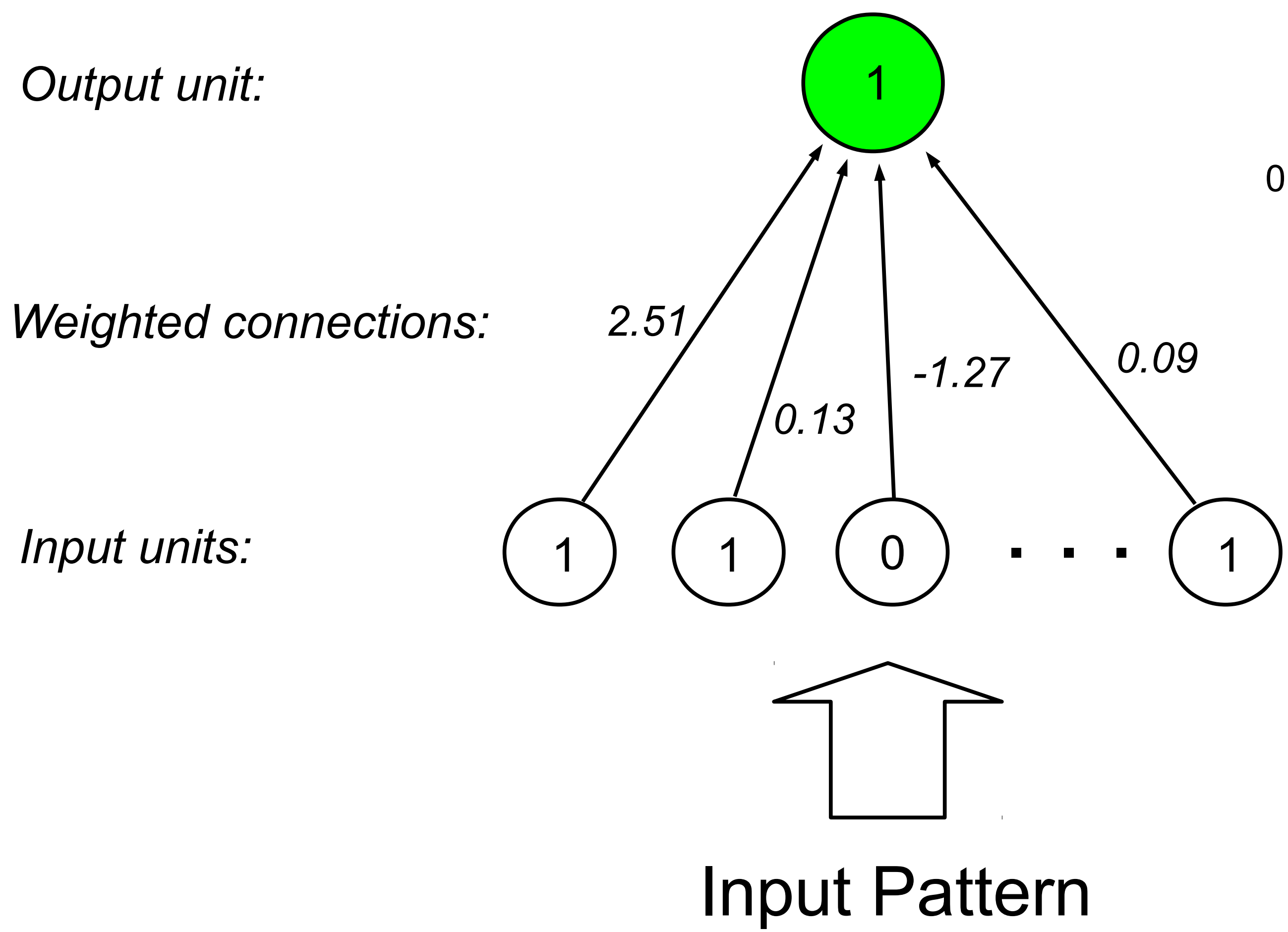
$1 \times 2.51 + 1 \times 0.13 + 0 \times -1.27 + \ldots + 1 \times 0.09 = 2.73$

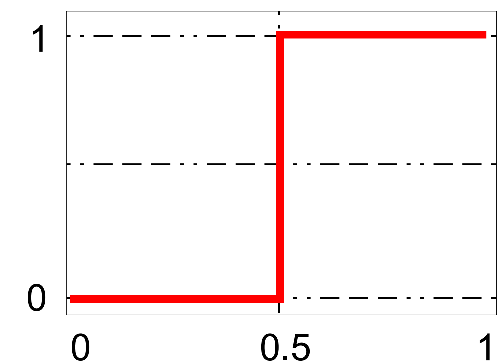


Artificial Neurons: Binary Version

$$1 \times 2.51 + 1 \times 0.13 + 0 \times -1.27 + \ldots + 1 \times 0.09 = 2.73$$

$$2.73 \geq 0.5$$



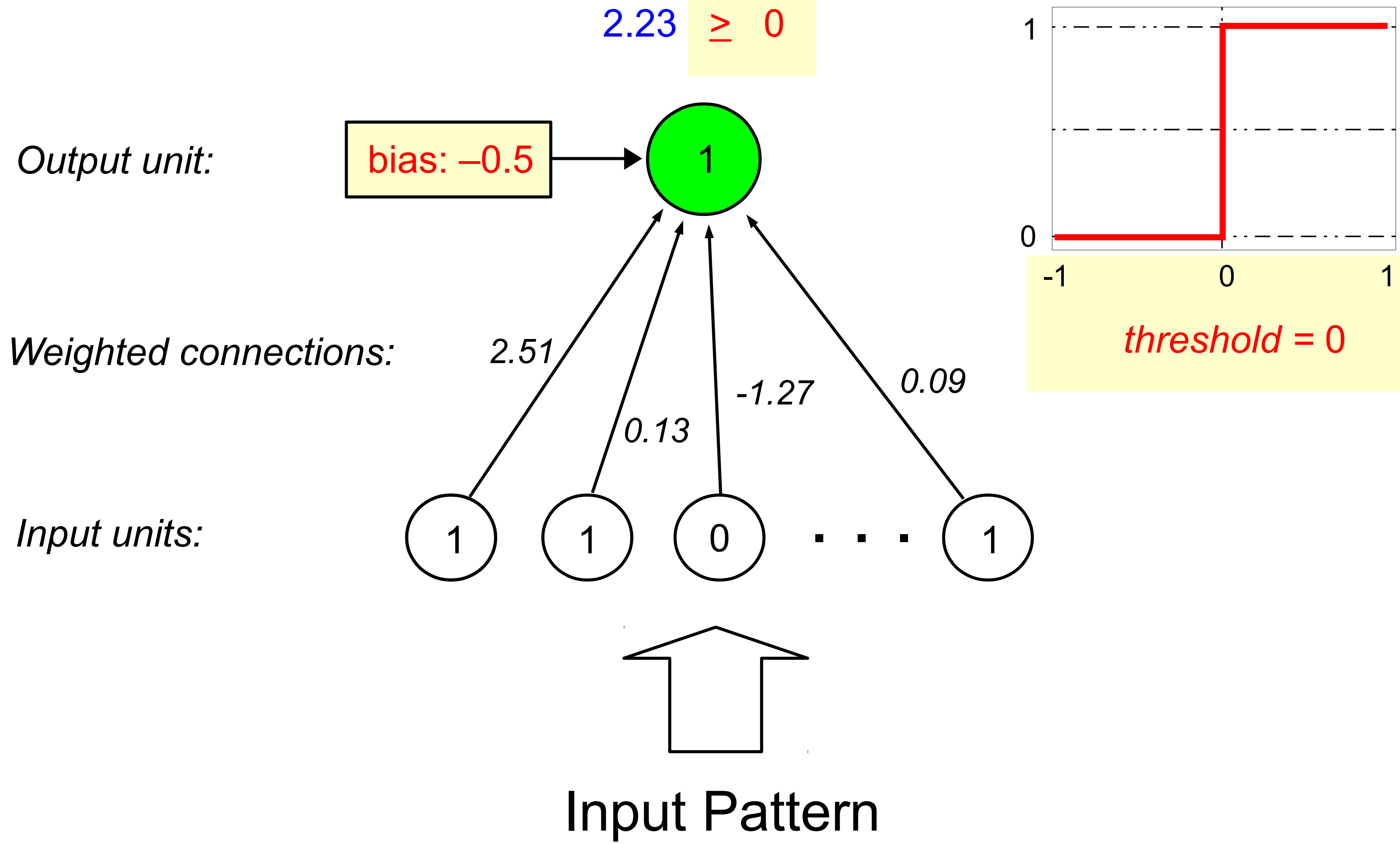


threshold = 0.5

Bias vs. Threshold

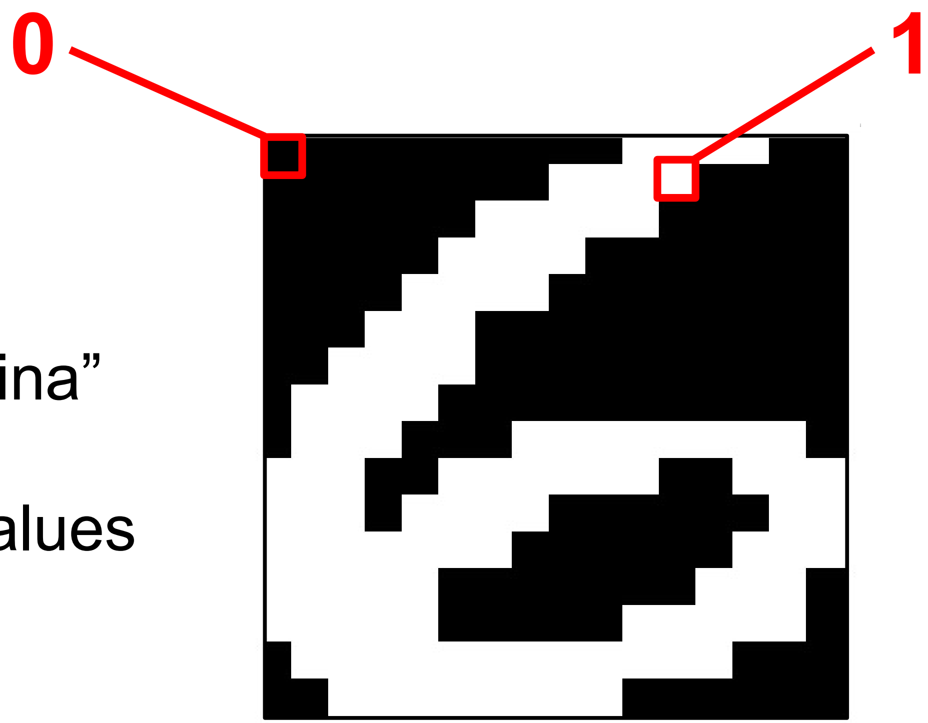
$$1 \times 2.51 + 1 \times 0.13 + 0 \times -1.27 + \ldots + 1 \times 0.09 - 0.5 = 2.23$$

$$2.23 \geq 0$$



Input Patterns

0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 . . .



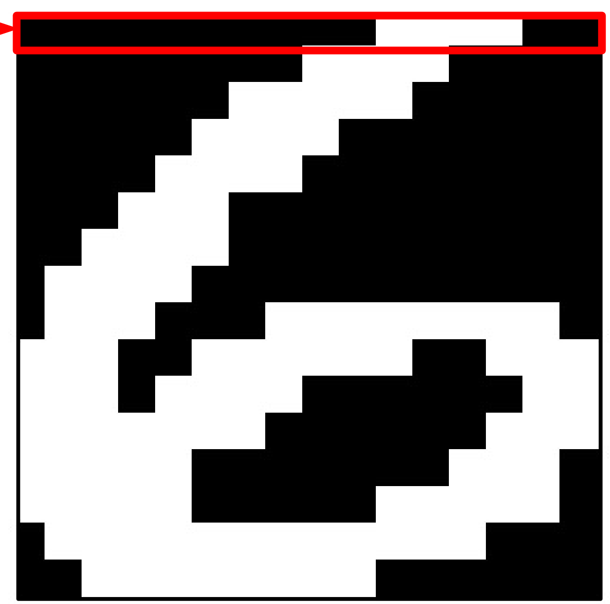
16 × 16 “retina”

256 binary values

Input Patterns

First row

0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 . . .



16 $\times$ 16 “retina”

256 binary values

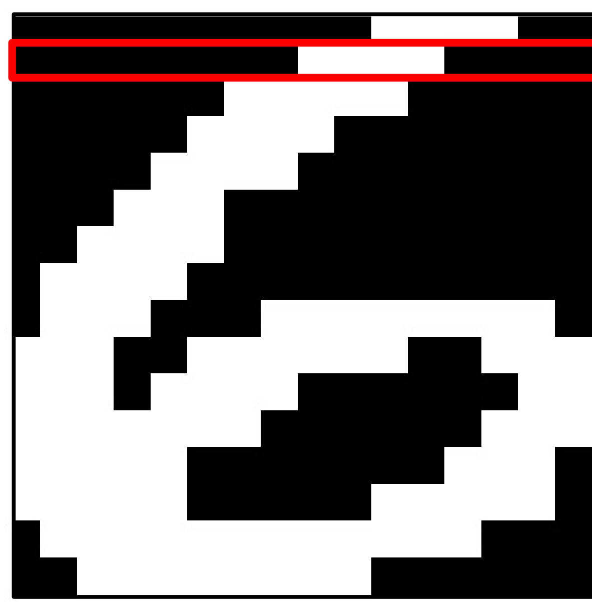
Input Patterns

First row — Second row — etc.

0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 . . .

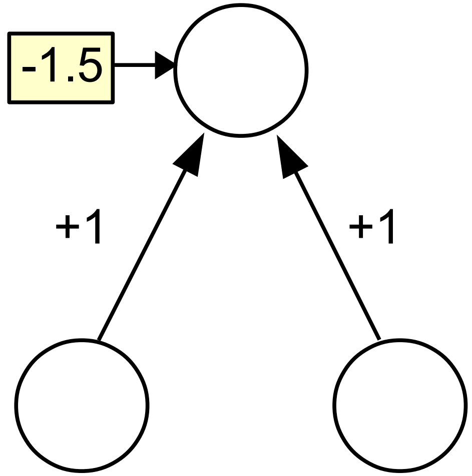
16 × 16 "retina"

256 binary values



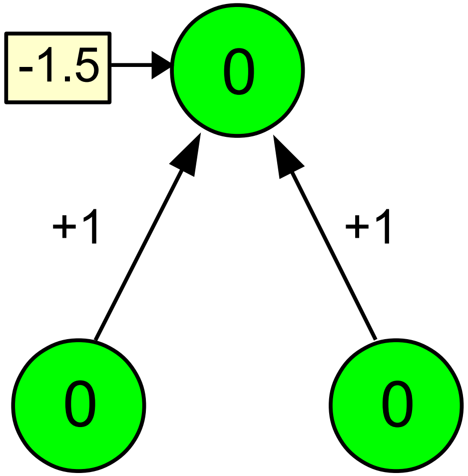
The Logical Function AND

0	0	$\Rightarrow$	0
0	1	$\Rightarrow$	0
1	0	$\Rightarrow$	0
1	1	$\Rightarrow$	1



The Logical Function AND

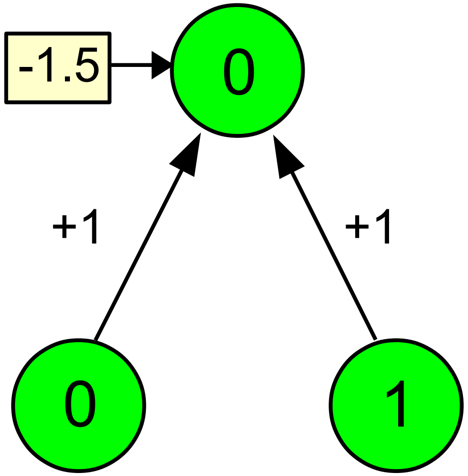
0	0	$\Rightarrow$	0
0	1	$\Rightarrow$	0
1	0	$\Rightarrow$	0
1	1	$\Rightarrow$	1



$$0 \times 1 + 0 \times 1 - 1.5 = \mathbf{-1.5} < 0$$

The Logical Function AND

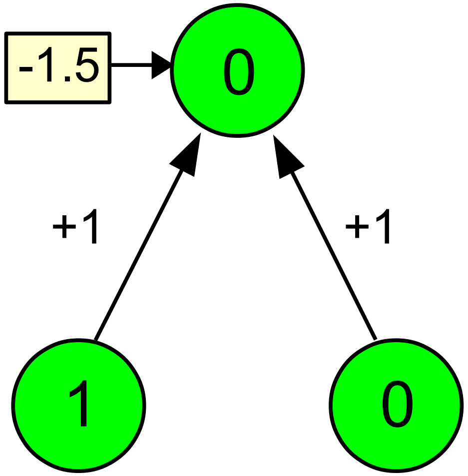
0	0	$\Rightarrow$	0
0	1	$\Rightarrow$	0
1	0	$\Rightarrow$	0
1	1	$\Rightarrow$	1



$$0 \times 1 \;+\; 1 \times 1 \;-\; 1.5 \;=\; \mathbf{-0.5} \;<\; 0$$

The Logical Function AND

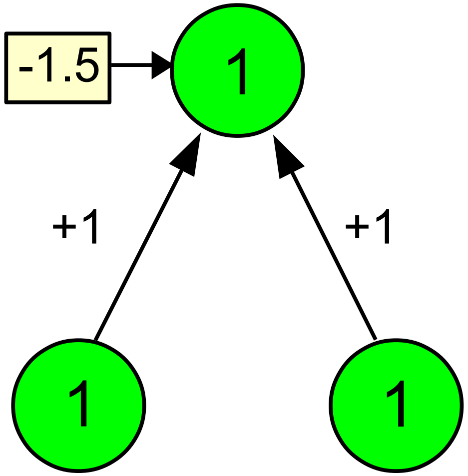
0	0	$\Rightarrow$	0
0	1	$\Rightarrow$	0
1	0	$\Rightarrow$	0
1	1	$\Rightarrow$	1



$$1 \times 1 + 0 \times 1 - 1.5 = \mathbf{-0.5} < 0$$

The Logical Function AND

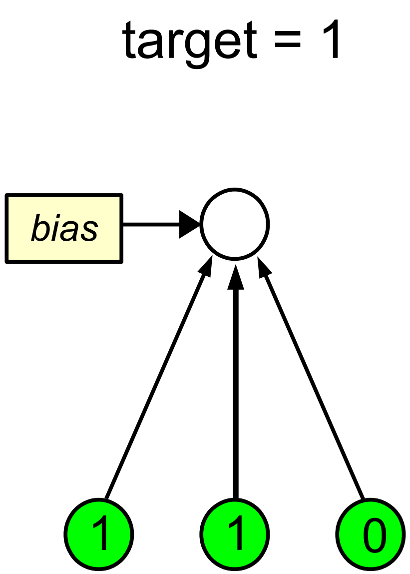
0	0	$\Rightarrow$	0
0	1	$\Rightarrow$	0
1	0	$\Rightarrow$	0
1	1	$\Rightarrow$	1



$$1 \times 1 \;+\; 1 \times 1 \;-\; 1.5 \;=\; \mathbf{+0.5} \;\geq 0$$

Perceptrons

- Binary threshold neurons
- Studied by Frank Rosenblatt of Cornell in early 1960's
- Perceptron training procedure
    1. present an input pattern

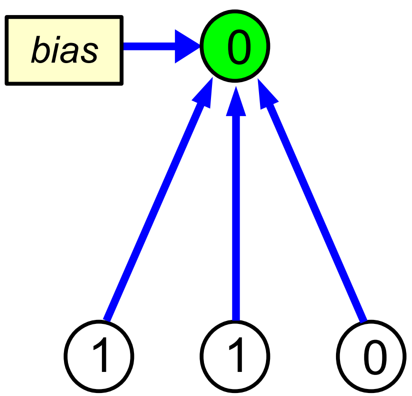


Perceptrons

- Binary threshold neurons
- Studied by Frank Rosenblatt of Cornell in early 1960's
- Perceptron training procedure

  1. present an input pattern
  2. compute output value

target = 1



output = Θ(*sum of: inputs* $\times$ *weights* + *bias*)

"threshold" function:

if sum $\geq$ 0: output = 1
if sum < 0: output = 0

Perceptrons

- Binary threshold neurons
- Studied by Frank Rosenblatt of Cornell in early 1960's
- Perceptron training procedure

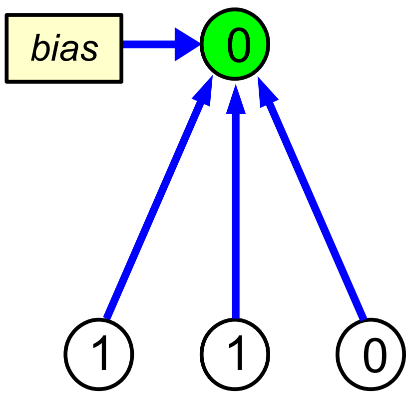
  1. present an input pattern
  2. compute output value

     $output = \Theta(sum\ of{:}\ inputs \times weights\ +\ bias)$
  3. compare output to target value

     $error = target - output$

target = 1

error = 1 – 0 = 1



Perceptrons

- Binary threshold neurons
- Studied by Frank Rosenblatt of Cornell in early 1960's
- Perceptron training procedure

1. present an input pattern
2. compute output value

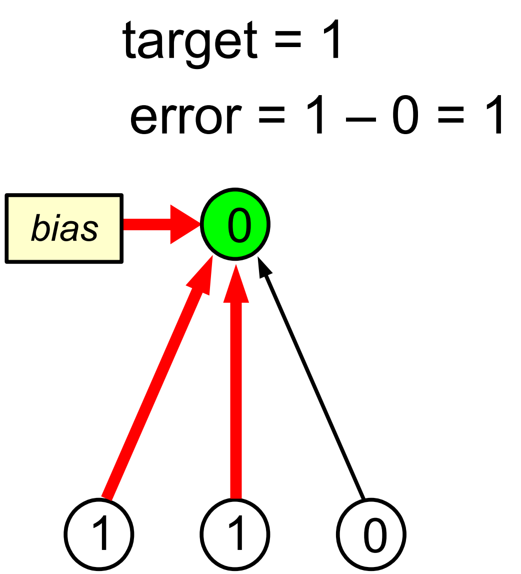
   output = Θ(*sum of: inputs* $\times$ *weights* + *bias*)
3. compare output to target value

   error = *target* – *output*
4. if incorrect, adjust weights and bias

   weight_adjustment = $\varepsilon \times$ *input* $\times$ *error*

   bias_adjustment = $\varepsilon \times$ *error*

"learning rate" $(0 < \varepsilon < 1)$



Perceptrons

- Binary threshold neurons
- Studied by Frank Rosenblatt of Cornell in early 1960's
- Perceptron training procedure
  1. present an input pattern
  2. compute output value

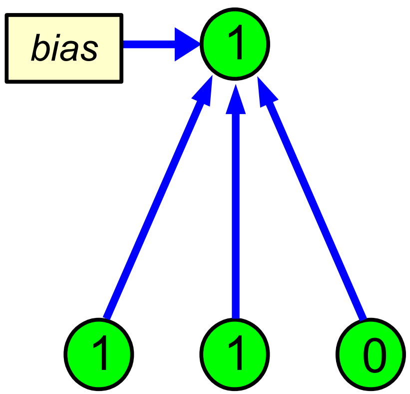
     output = Θ(*sum of: inputs* $\times$ *weights* + *bias*)
  3. compare output to target value

     error = *target* – *output*
  4. if incorrect, adjust weights and bias

     weight_adjustment = $\varepsilon \times$ *input* $\times$ *error*
  5. repeat until all input patterns give the correct output value

target = 1

error = 0



Perceptrons

- **Perceptron learning theorem**

> *The perceptron training procedure is **guaranteed** to find weight values that correctly solve the problem, within a finite number of steps, provided such weight values exist.*

- Not all problems can be solved by **single-layer** perceptrons
- Classic example: **XOR**

$$0\ 0 \Rightarrow 0 \qquad 0\ 1 \Rightarrow 1$$
$$1\ 1 \Rightarrow 0 \qquad 1\ 0 \Rightarrow 1$$

- Perceptrons with **multiple layers** of weights can solve XOR
- But **no training procedure** or **learning theorem** for multi-layer networks was known in the 1960s

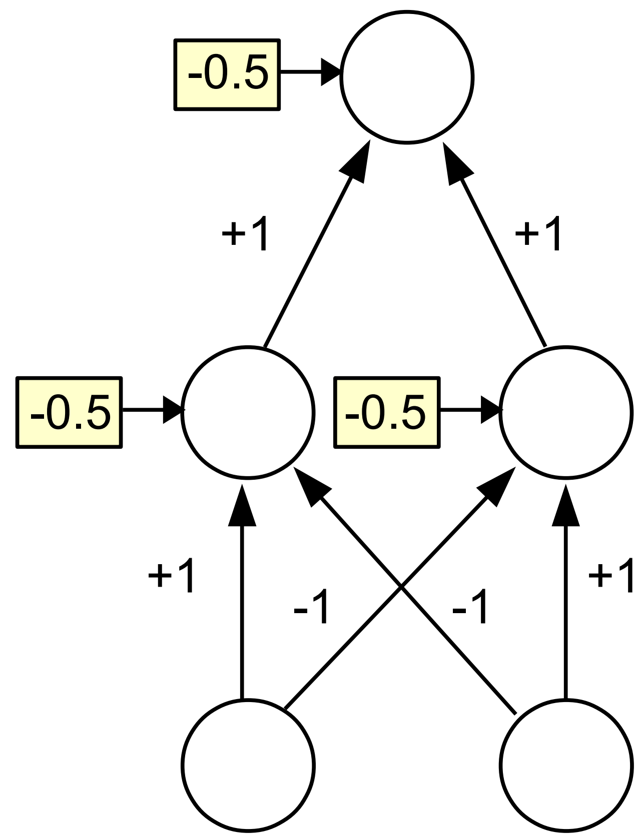
A Neural Network for XOR

0 0 $\Rightarrow$ 0

0 1 $\Rightarrow$ 1

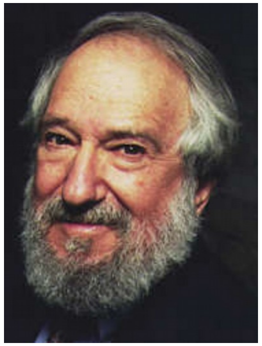
1 0 $\Rightarrow$ 1

1 1 $\Rightarrow$ 0

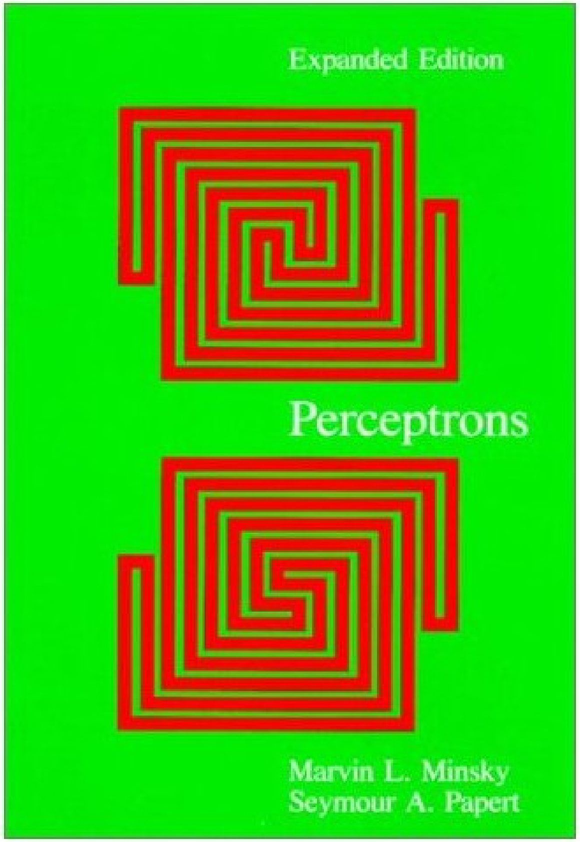


Perceptrons

- Marvin Minsky and Seymour Papert of MIT published *Perceptrons* in 1969



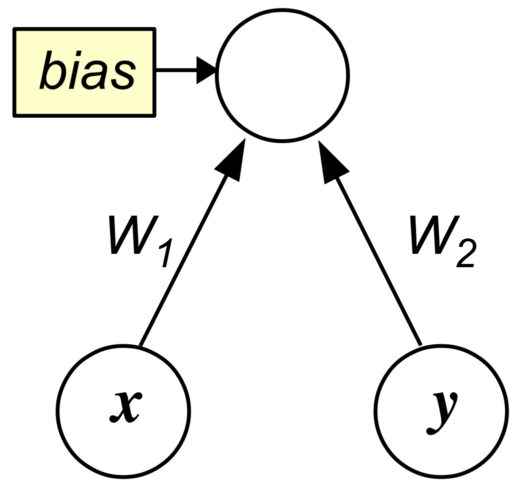
- They rigorously analyzed the limitations of perceptrons, and doubted that a training procedure existed for networks with multiple layers of weights



- This caused many people to seriously question the potential of neural networks

- As a result, interest in neural network research (and funding) largely dried up for more than a decade

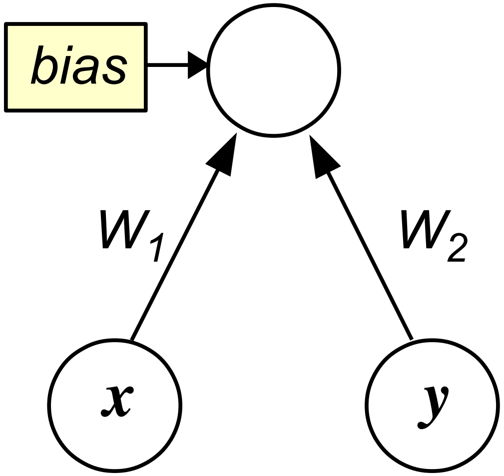
A perceptron is an "adjustable line"



$$W_1\, x \;+\; W_2\, y \;+\; bias \;=\; sum$$

- When *sum* > 0, the input x, y is classified one way (1)
- When *sum* < 0, the input x, y is classified the other way (0)
- When *sum* = 0, the input x, y is right on the "border"

A perceptron is an “adjustable line”

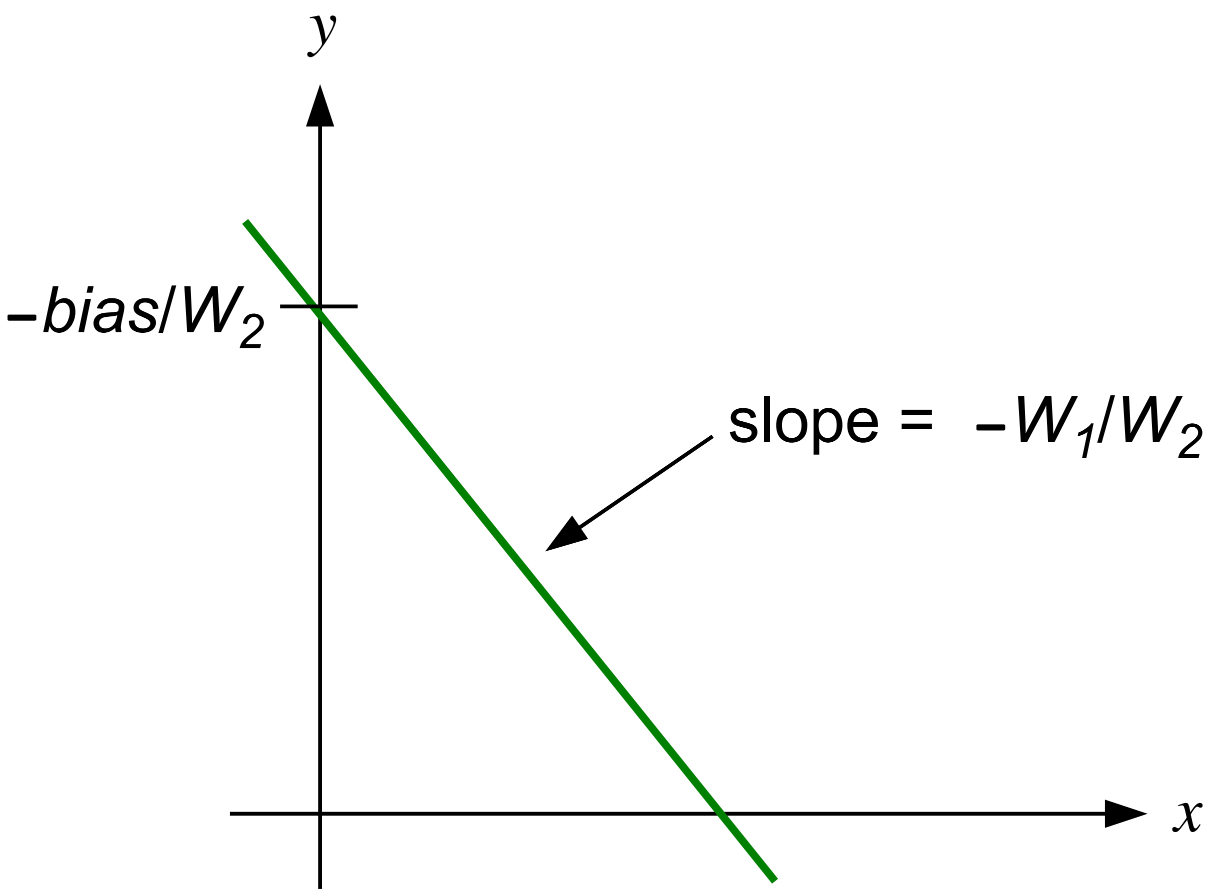


$$W_1\,x \;+\; W_2\,y \;+\; bias \;=\; 0$$

This is the equation of a line, which we can rewrite in standard slope-intercept form as $y = mx + b$:

$$y \;=\; \underbrace{-W_1/W_2}_{\text{Slope of line}}\; x \;+\; \underbrace{-bias/W_2}_{\text{Intercept of line with } y\text{-axis}}$$

A perceptron is an "adjustable line"



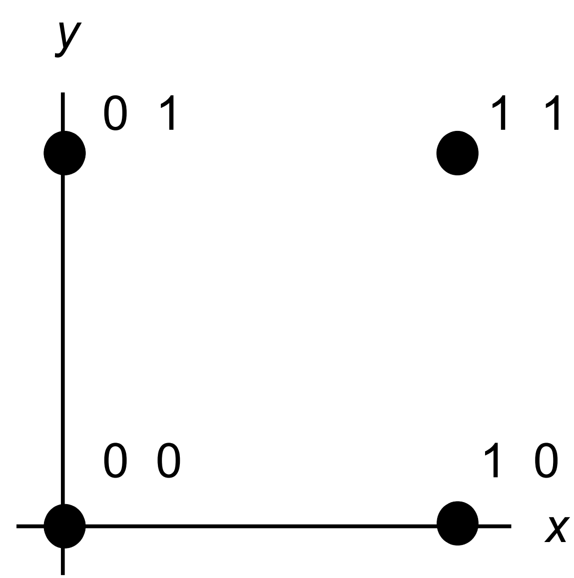
By adjusting the values of $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, and ***bias***, we can change the orientation of the line in any way we like

Linear Separability

- Input patterns correspond to **points** in the **input space**

input patterns

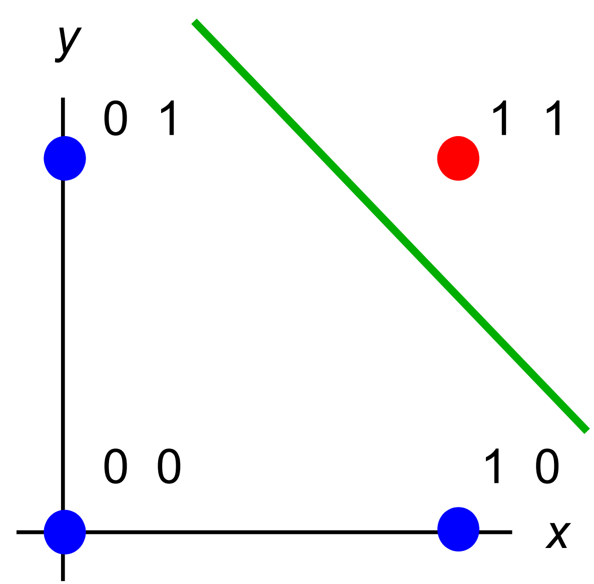
x	*y*	*x* **AND** *y*
0	0	0
0	1	0
1	0	0
1	1	1



Linear Separability

- Input patterns correspond to **points** in the **input space**
- A perceptron that correctly classifies input patterns as belonging to category A or category B corresponds to a **straight line** dividing the input space into two halves
- The two categories of input patterns are **linearly separable**

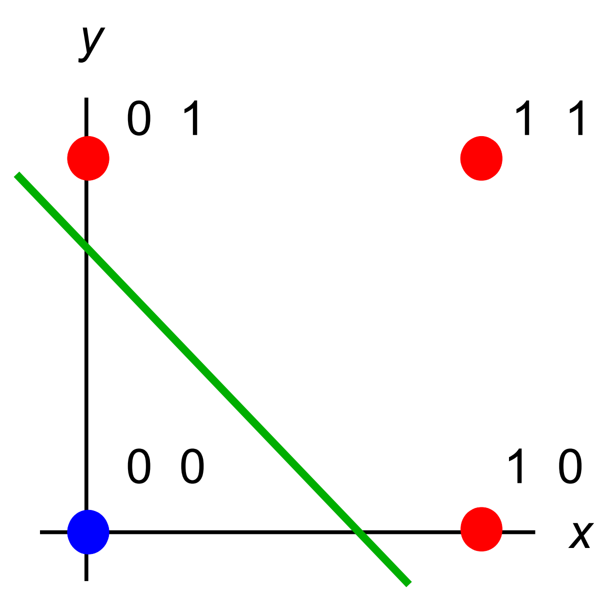
	x	*y*	*x* **AND** *y*
	0	0	0
A	0	1	0
	1	0	0
B	1	1	1



Linear Separability

- Input patterns correspond to **points** in the **input space**
- A perceptron that correctly classifies input patterns as belonging to category A or category B corresponds to a **straight line** dividing the input space into two halves
- The two categories of input patterns are **linearly separable**

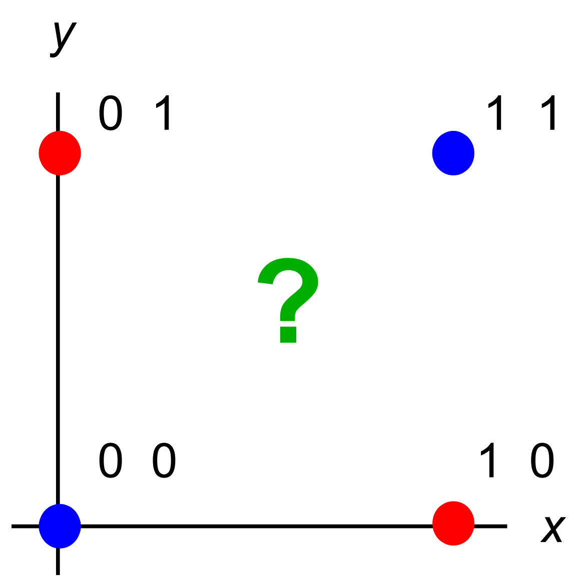
	x	*y*	*x* **OR** *y*
A	0	0	0
	0	1	1
B	1	0	1
	1	1	1



Linear Separability

- Input patterns correspond to **points** in the **input space**
- A perceptron that correctly classifies input patterns as belonging to category A or category B corresponds to a **straight line** dividing the input space into two halves
- The two categories of input patterns are **linearly separable**

	x	*y*	*x* **XOR** *y*
A	0	0	0
B	0	1	1
	1	0	1
A	1	1	0

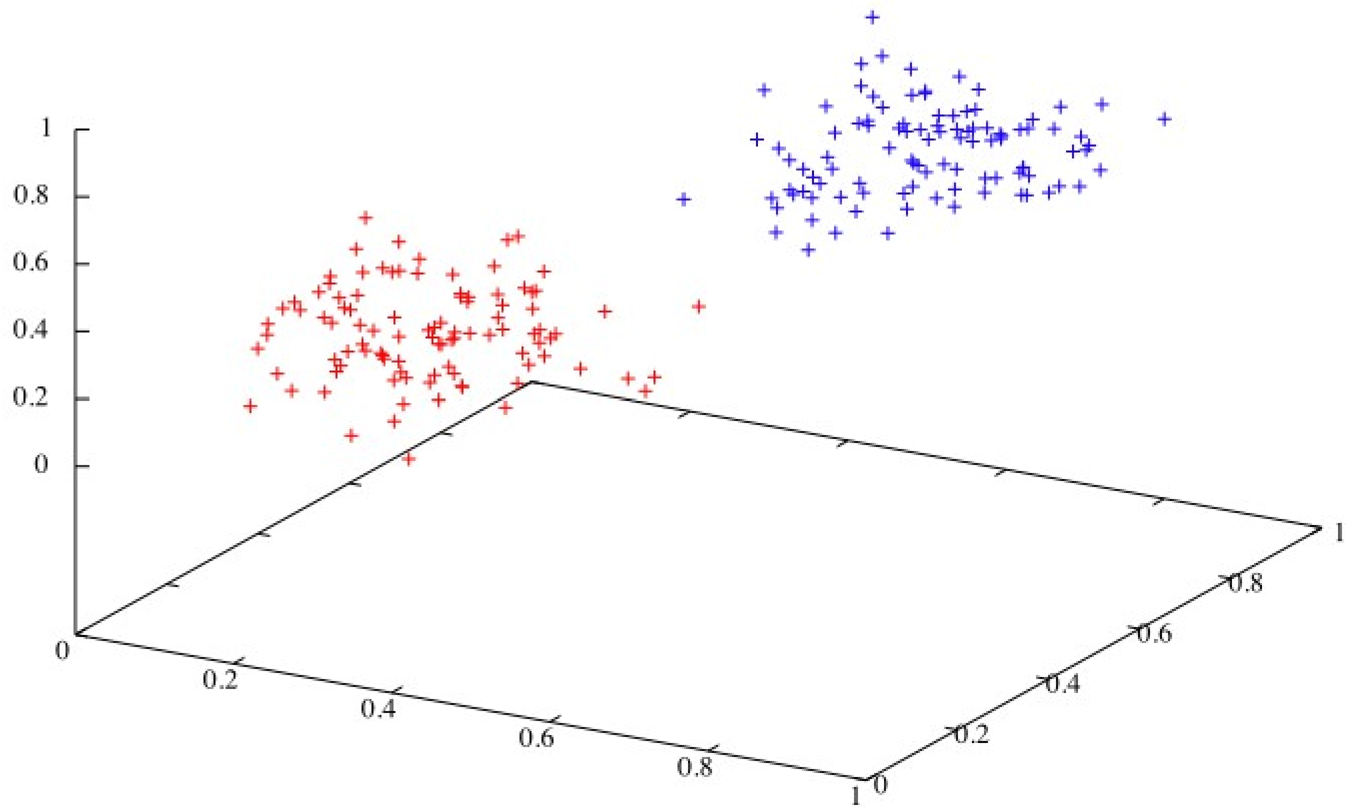


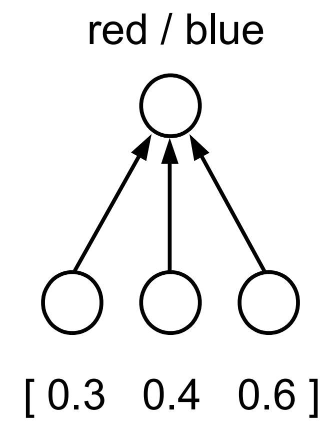
Linear Separability

- Minsky and Papert proved that many interesting problems are not linearly separable, and thus no perceptron can learn them

Linear Separability

- This idea applies to input spaces of any dimensionality
- Example: 3-dimensional input patterns

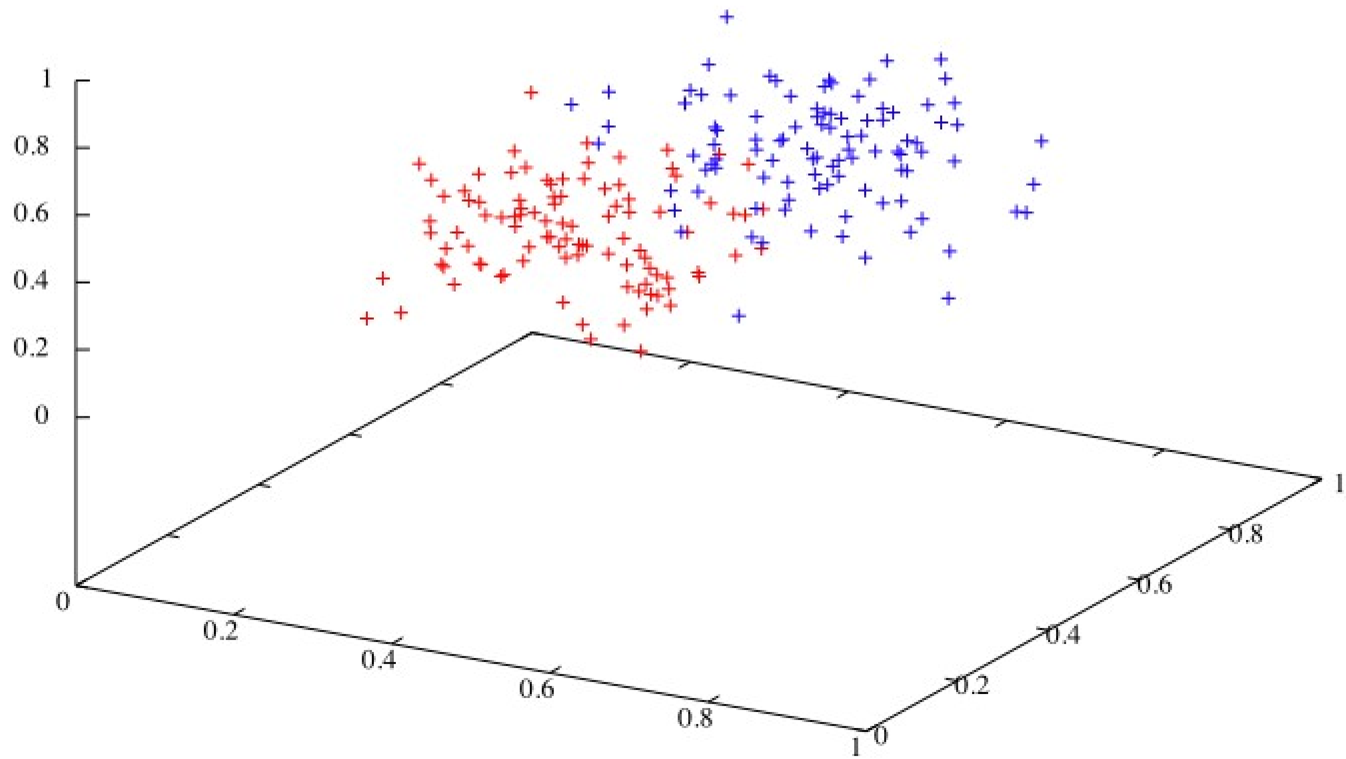


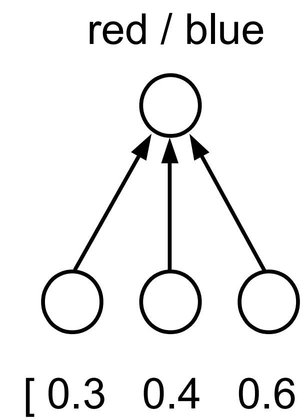


linearly separable

Linear Separability

- This idea applies to input spaces of any dimensionality
- Example: 3-dimensional input patterns

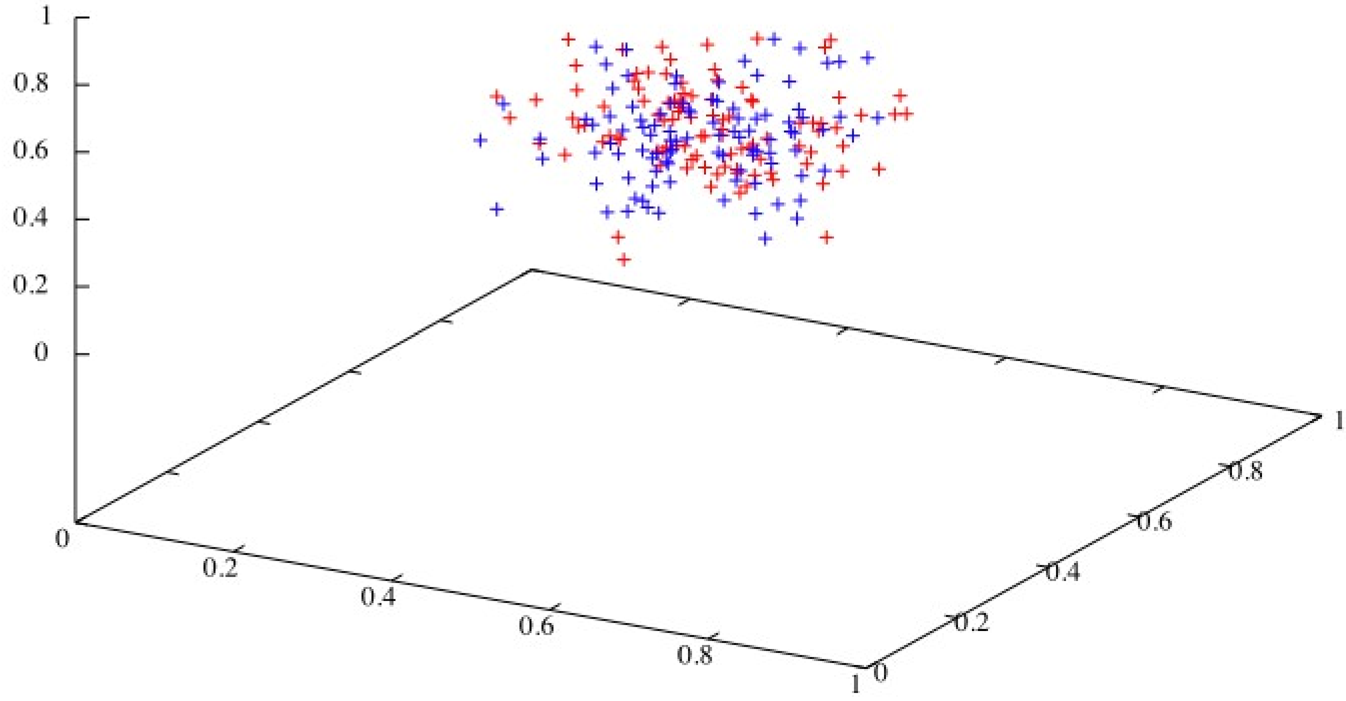


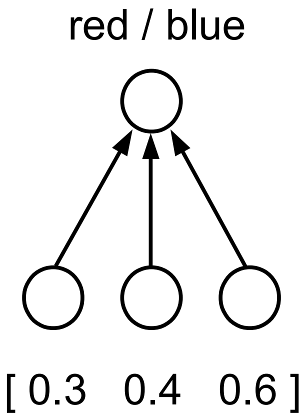


partially linearly separable

Linear Separability

- This idea applies to input spaces of any dimensionality
- Example: 3-dimensional input patterns

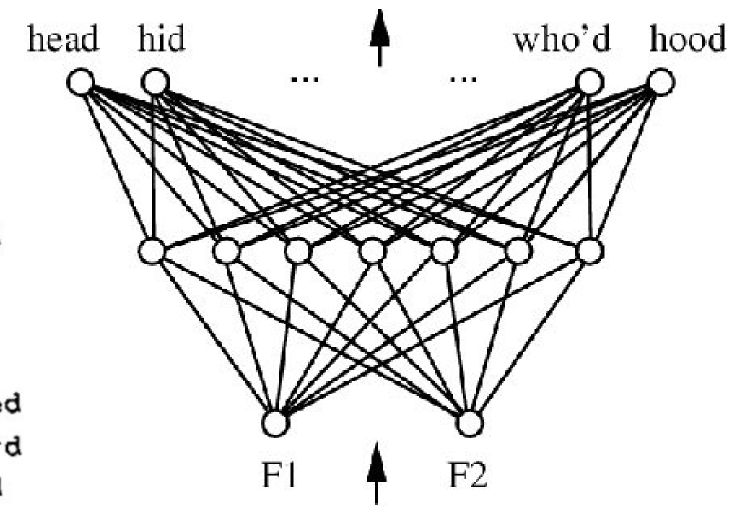


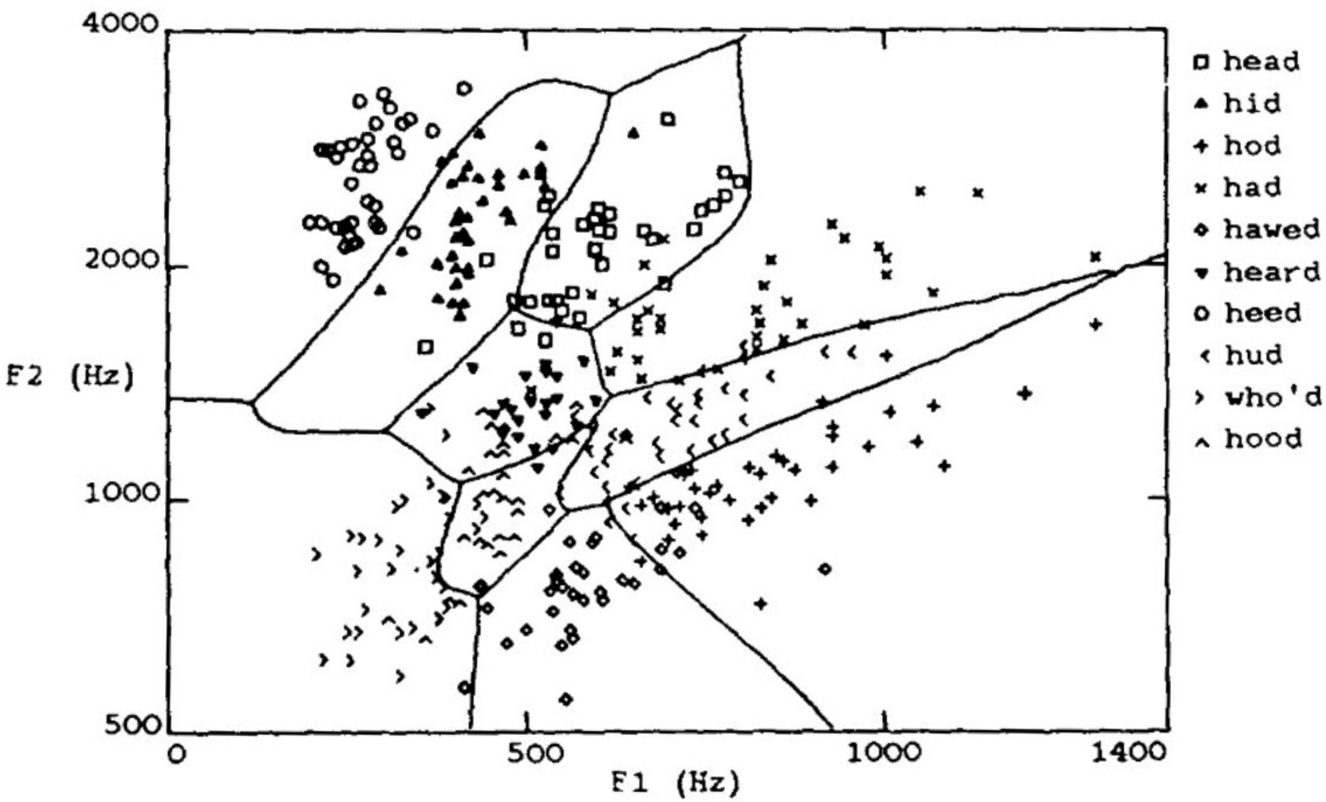


not linearly separable

Linear Separability

- Multi-layer networks can learn to classify input patterns that are not linearly separable
- Example: recognizing vowels





Parallel Distributed Processing (PDP)

- In the 1980s, a way to train multi-layer networks was discovered, called the **backpropagation** learning algorithm
- David Rumelhart, Geoffrey Hinton, James McClelland, and others revived interest in neural networks with the publication of the "PDP books"
- Showed that Minsky and Papert's analysis was too pessimistic
- Backpropagation is one of the key components of modern-day research in **deep learning**