Data-driven Intelligent Systems

Lecture 9
Neural Networks - Perceptron

KNOWLEDGE
TECHNOLOGY

http://www.informatik.uni-hamburg.de/WTM/

Overview

- Biological Background
- Perceptron
  - Perceptron Layer
  - Linear Separability
- Multilayer Perceptron (next lecture)

Why Learning? Some Quotes

- “Artificial Intelligence is realised only when a computer can ‘discover’ ***for itself*** new techniques for problem solving”
  Fogel (1966)

- “Intelligent agents must be able to ***change*** through the course of their interactions with the world“
  Luger (2002)

- “A machine or software tool would not be viewed as intelligent if it could not ***adapt to*** changes in its environment“
  Callan (2003)

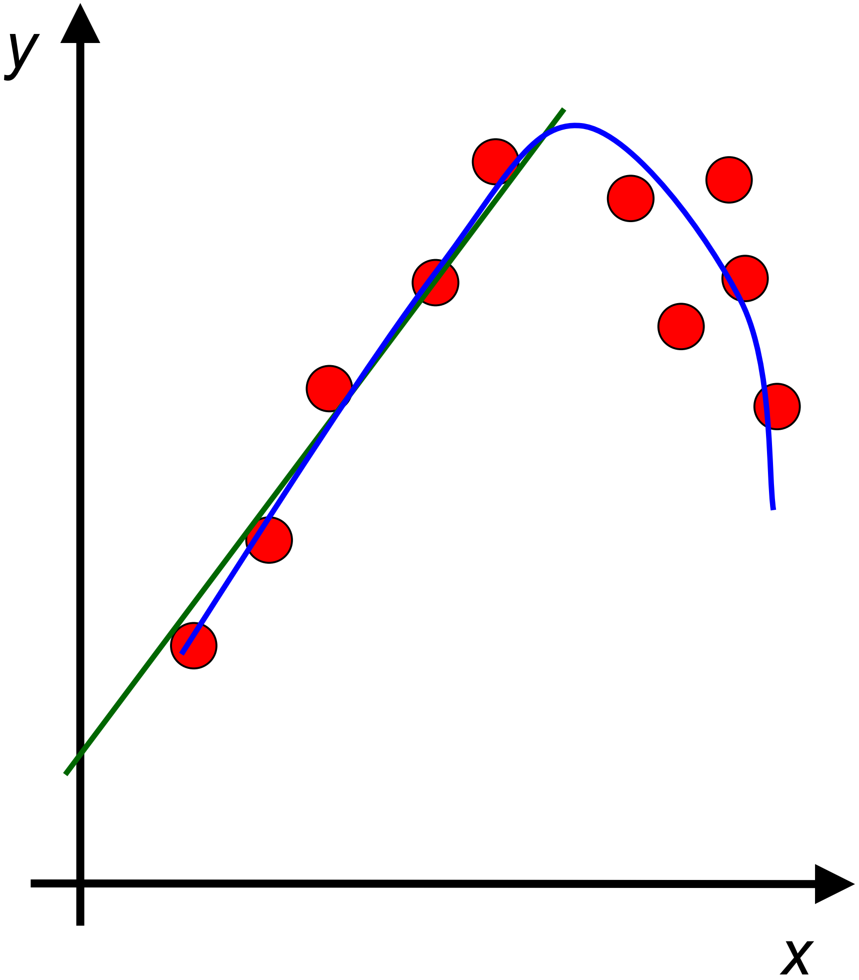
What is Neural Learning?

- Modify and improve behaviour by past experience
- How does the brain learn?
  - Strengths of synaptic connections vary

- Hebb's rule
  - If two neurons connected by a synapse fire simultaneously then the synapse strengthens
  - If two neurons connected by a synapse do not fire simultaneously then the synapse weakens

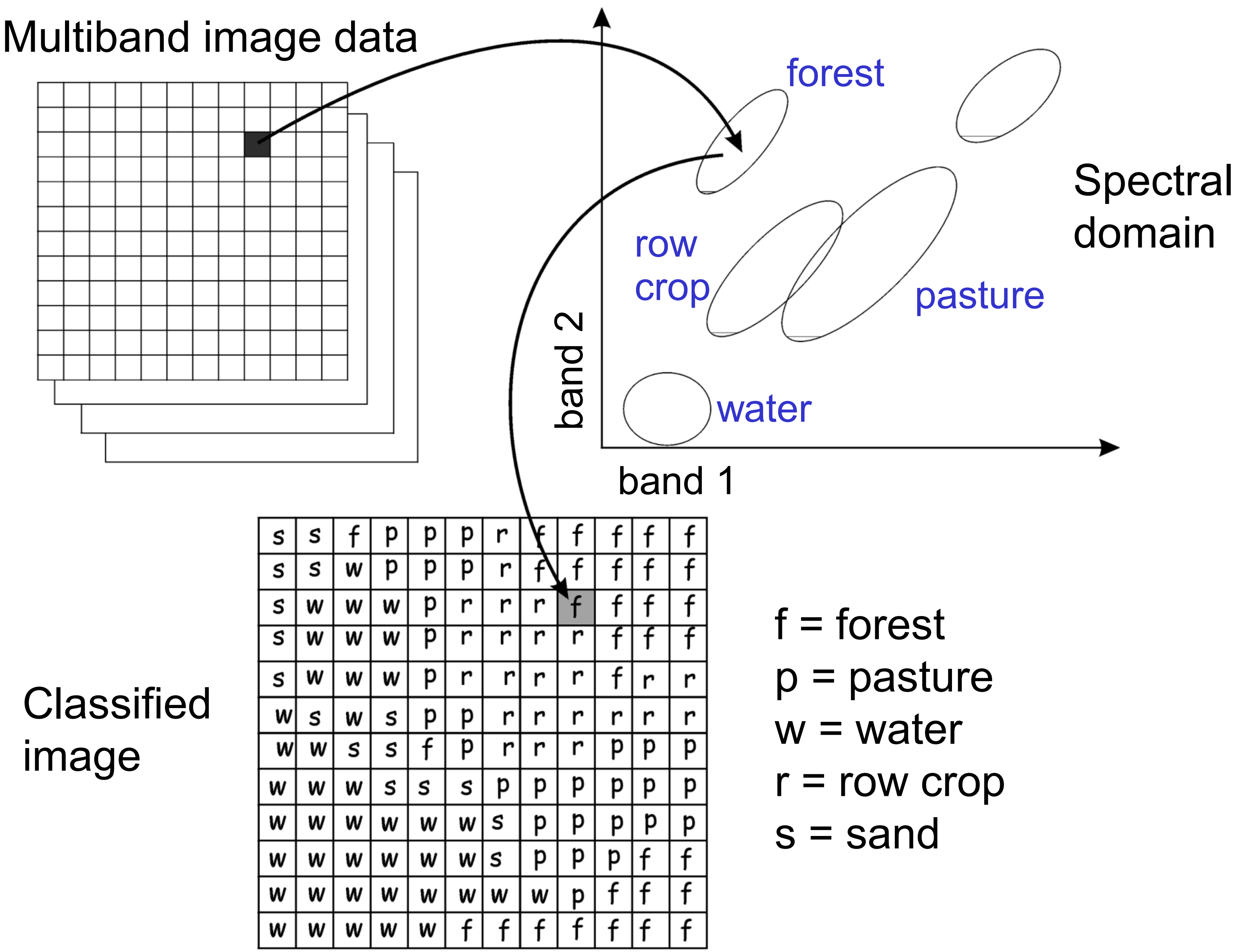
→ "fire together, wire together"

Learning Regression Problems



- Curve Fitting (with ***noise***)
- Function Approximation
- Many other functions could fit the data

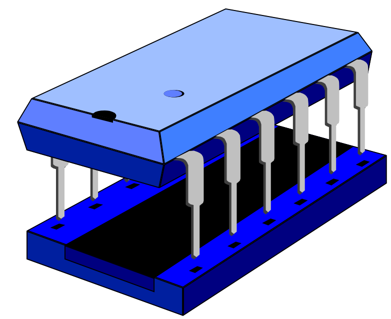
Learning Classification Problems

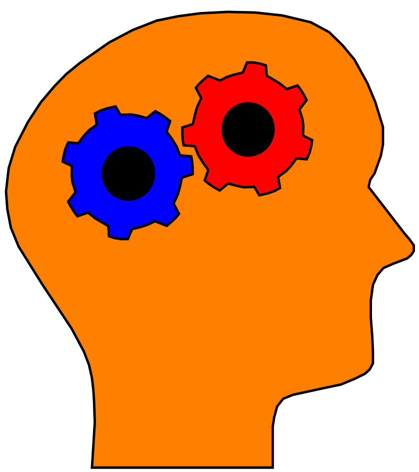


f = forest
p = pasture
w = water
r = row crop
s = sand

Computer versus Brain

- The ***von Neumann architecture*** uses a single processing unit
  - Floating Point Operations Per Second (typical today: 1 Tera FLOPS, 10^{12})
  - Absolute arithmetic precision





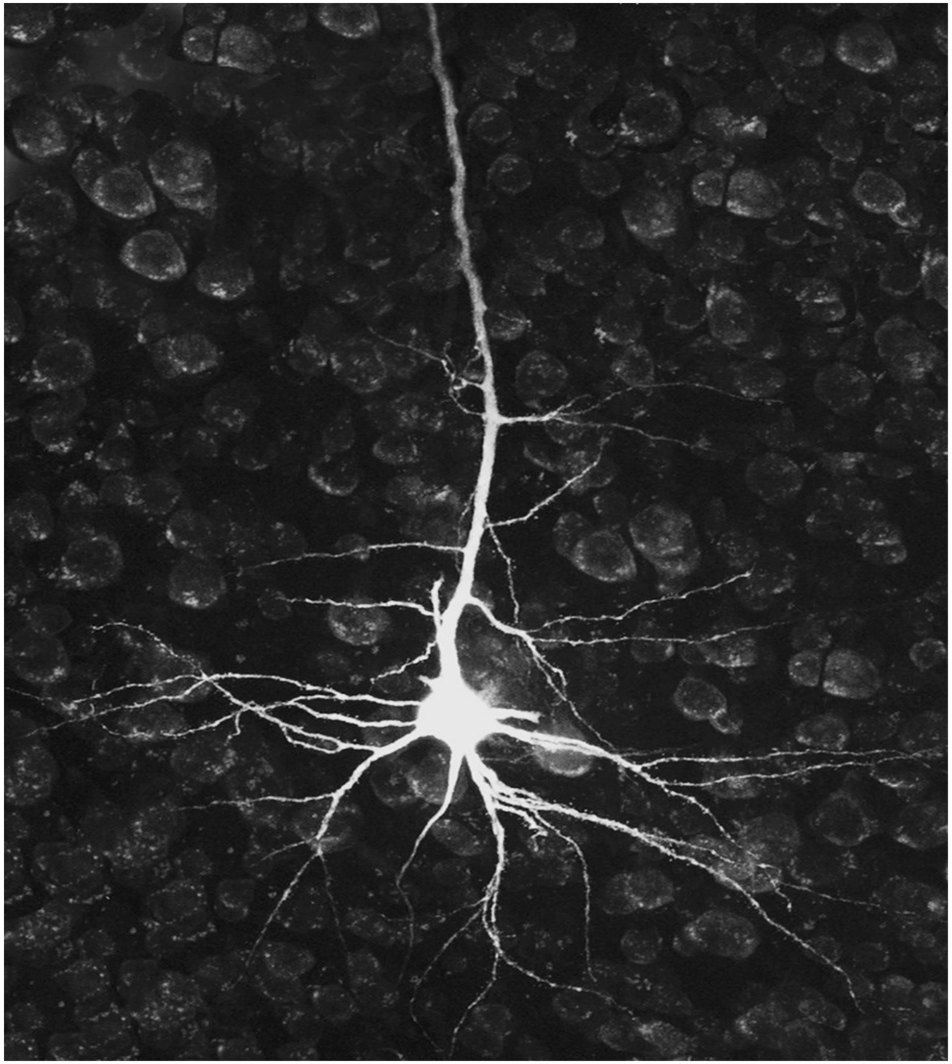
The ***brain***

- Uses many but slow, unreliable processors acting in parallel but they produce robust learned behaviour

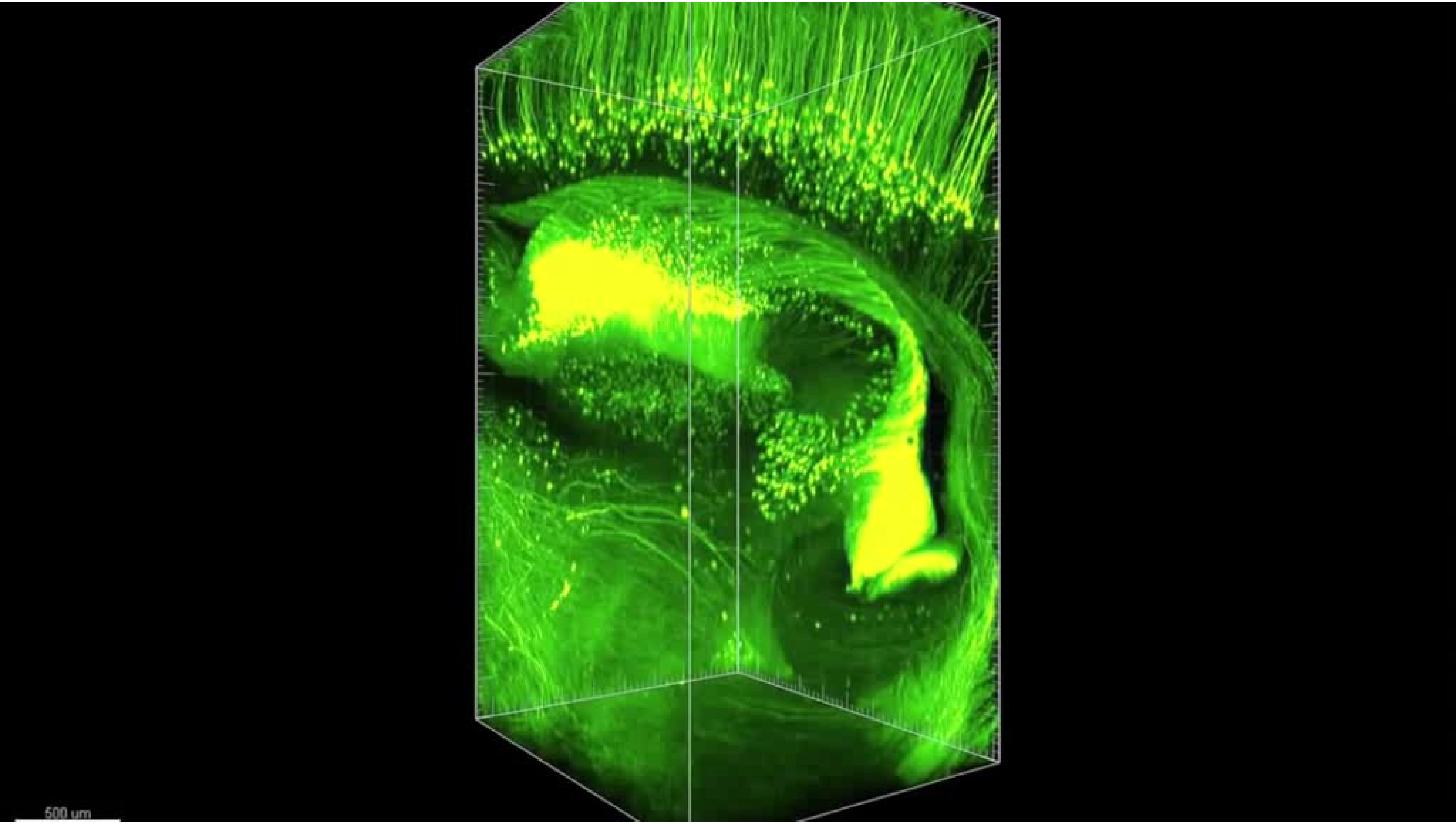
Overview

- Biological Background
- Perceptron
  - Perceptron Layer
  - Linear Separability

A Real Neuron

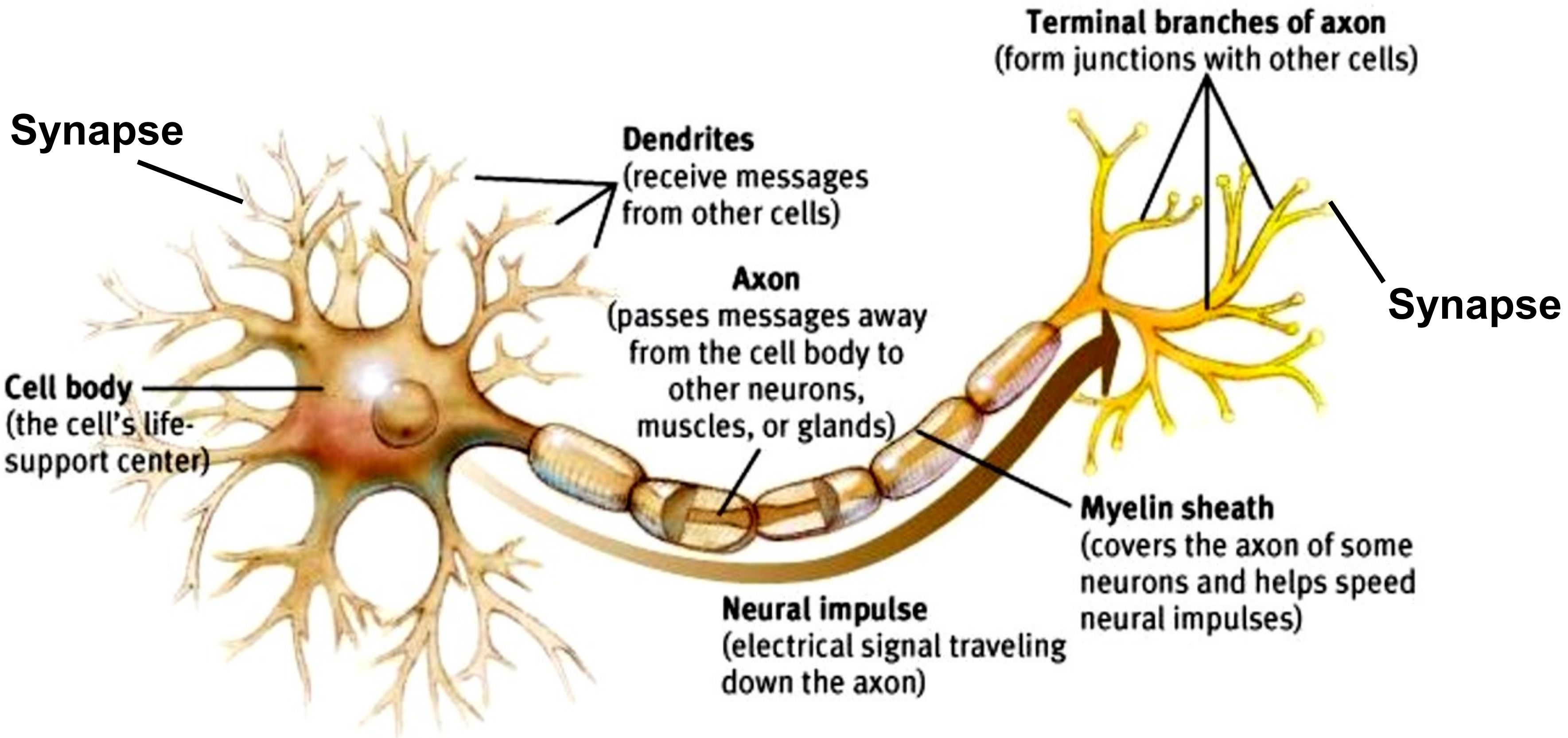


3D-View of Neurons in the Brain



Shen, H. See-through brains clarify connections. Nature, vol. 496, pp. 151, Macmillan Publishers Limited, 11 April 2013. [Video online]

The Neuron

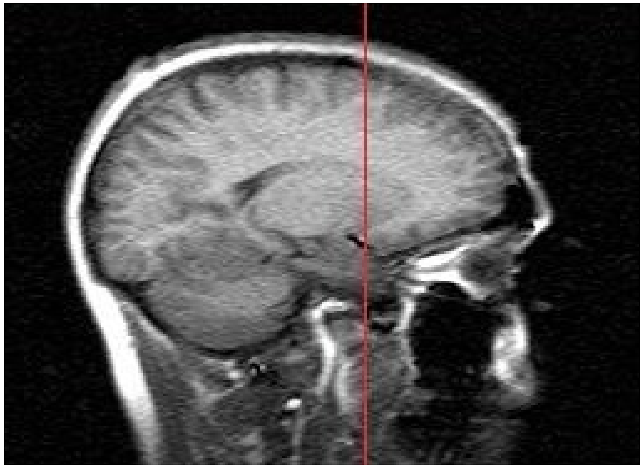


Synapse
Terminal branches of axon
(form junctions with other cells)
Dendrites
(receive messages
from other cells)
Axon
(passes messages away
from the cell body to
other neurons,
muscles, or glands)
Synapse
Cell body
(the cell's life-
support center)
Myelin sheath
(covers the axon of some
neurons and helps speed
neural impulses)
Neural impulse
(electrical signal traveling
down the axon)

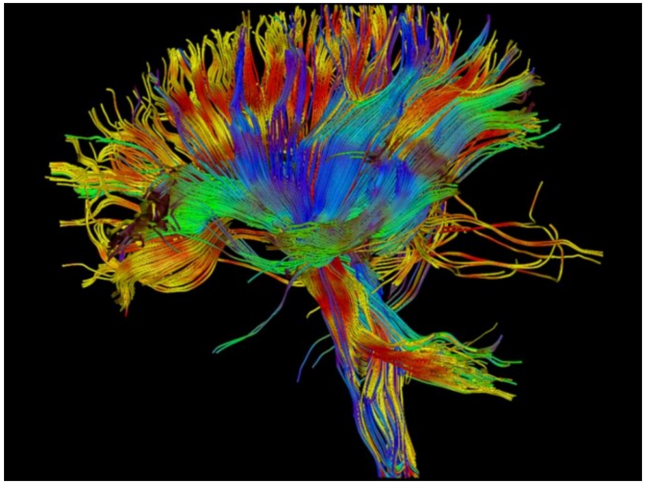
A Neuron`s Firing

Sound: amplified action potentials ("spikes")

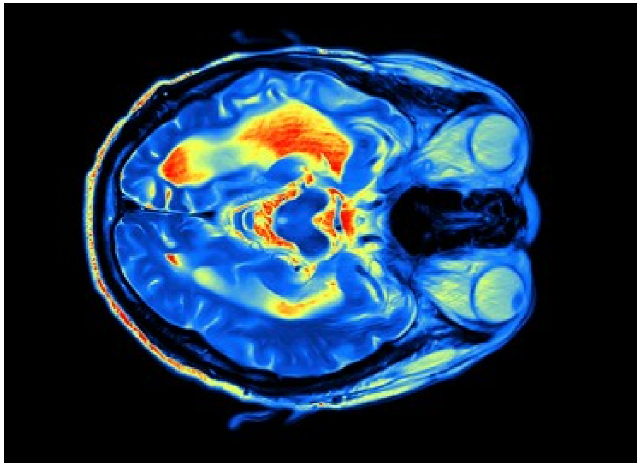
Noninvasive Inspections of the Brain



Computer tomography



Diffusion tensor imaging



Functional magnetic resonance imaging (fMRI)

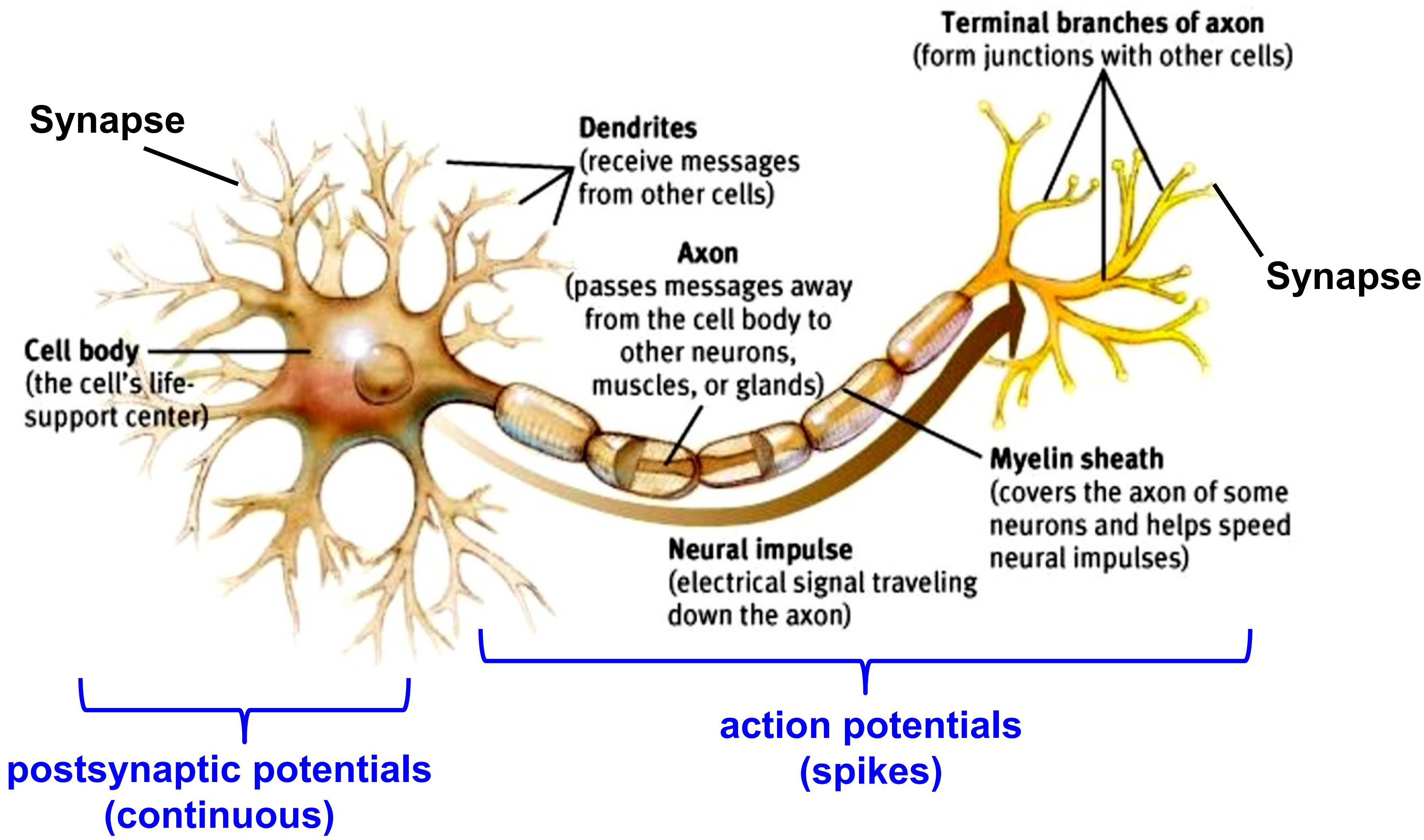
Parallel Processing in the Brain

The human brain:

- Weight on average 1.4kg
- Contains around ***10^{11} neurons***
  - In computational terms, 10^{11} simple processors
    - Each takes a few milliseconds to do a computation
    - But the whole brain is very fast
  - Many *different types*
- Has about ***10^{14} synapses***
  - ***Highly connected***
  - Things done massively in parallel
  - Robust to faults

Neuron Activity

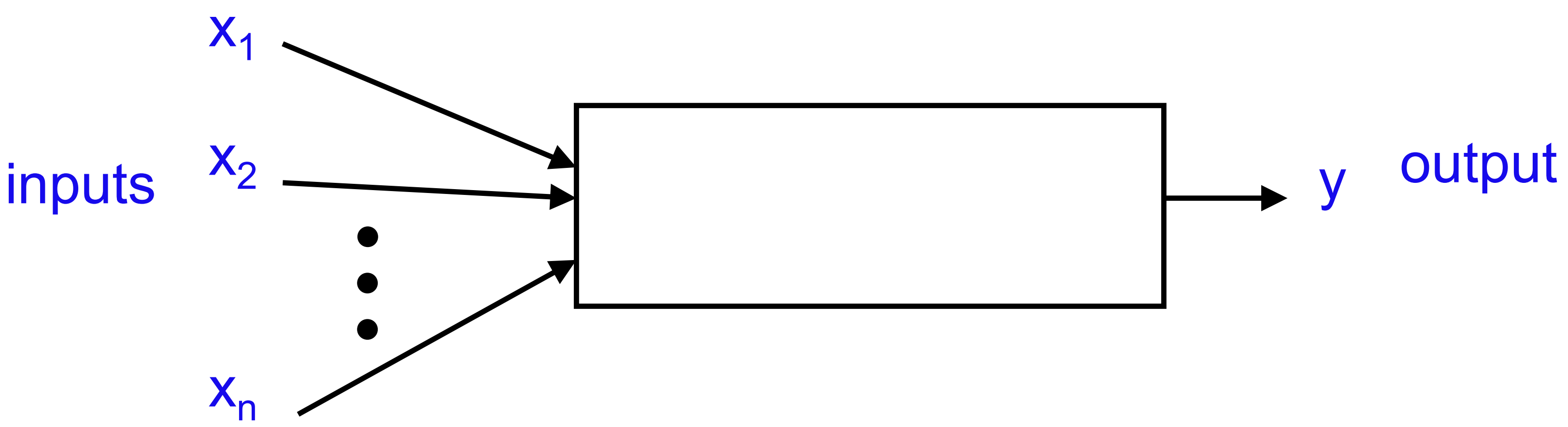
The Neuron


Terminal branches of axon
(form junctions with other cells)
Synapse
Dendrites
(receive messages
from other cells)
Axon
(passes messages away
from the cell body to
other neurons,
muscles, or glands)
Synapse
Cell body
(the cell's life-
support center)
Myelin sheath
(covers the axon of some
neurons and helps speed
neural impulses)
Neural impulse
(electrical signal traveling
down the axon)
postsynaptic potentials
(continuous)
action potentials
(spikes)

Overview

- Biological Background
- Perceptron
  - Perceptron Layer
  - Linear Separability

Perceptron Neuron

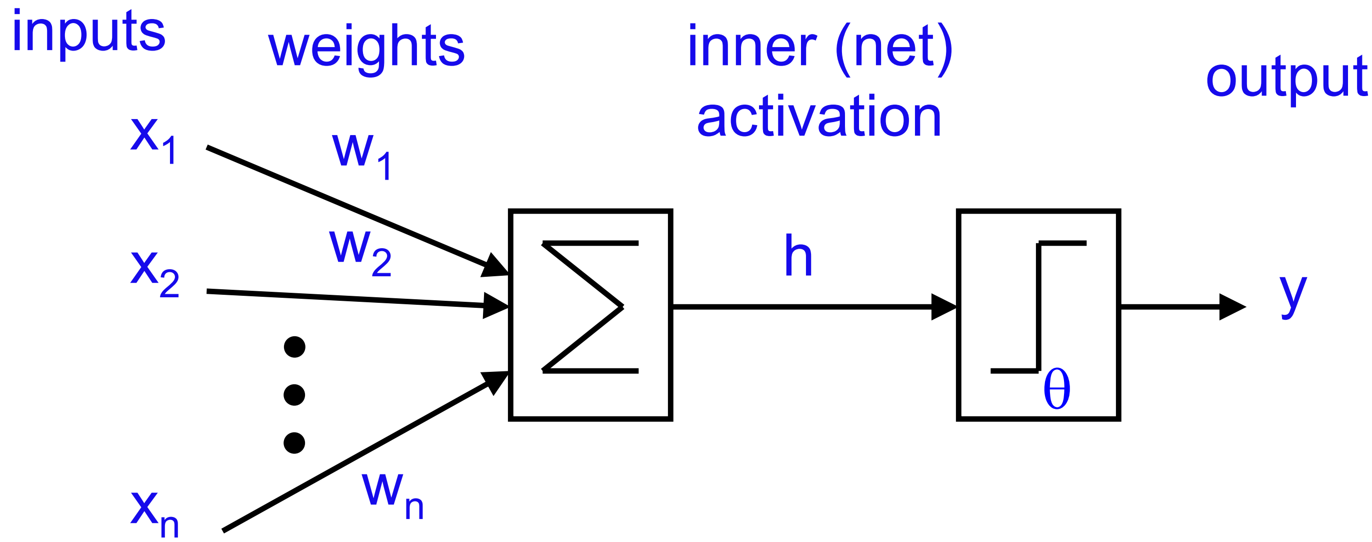


One neuron transforms (multiple) inputs $\{x_j\}$ to one output y

- Typically, a data point **x** is presented as input vector/tuple
- **x** consists of the values $\{x_1, x_2, \ldots, x_n\}$ of the data attributes

- Outputs of multiple neurons can be combined into a vector and presented as input to other neurons → neural network

Perceptron Neuron



Greatly simplified biological neurons

- Sum the inputs x_j each being weighted with weight w_j
- The total sum is h
- If h is more than some threshold θ
  - then neuron fires: $y = 1$,
  - else not: $y = 0$ (sometimes used: $y = -1$)

Perceptron Neuron

n input neurons

$$h = \sum_{j=1}^{n} w_j x_j$$

$$y = \begin{cases} 1 & h \geq \theta \\ 0 & h < \theta \end{cases}$$

for some threshold θ

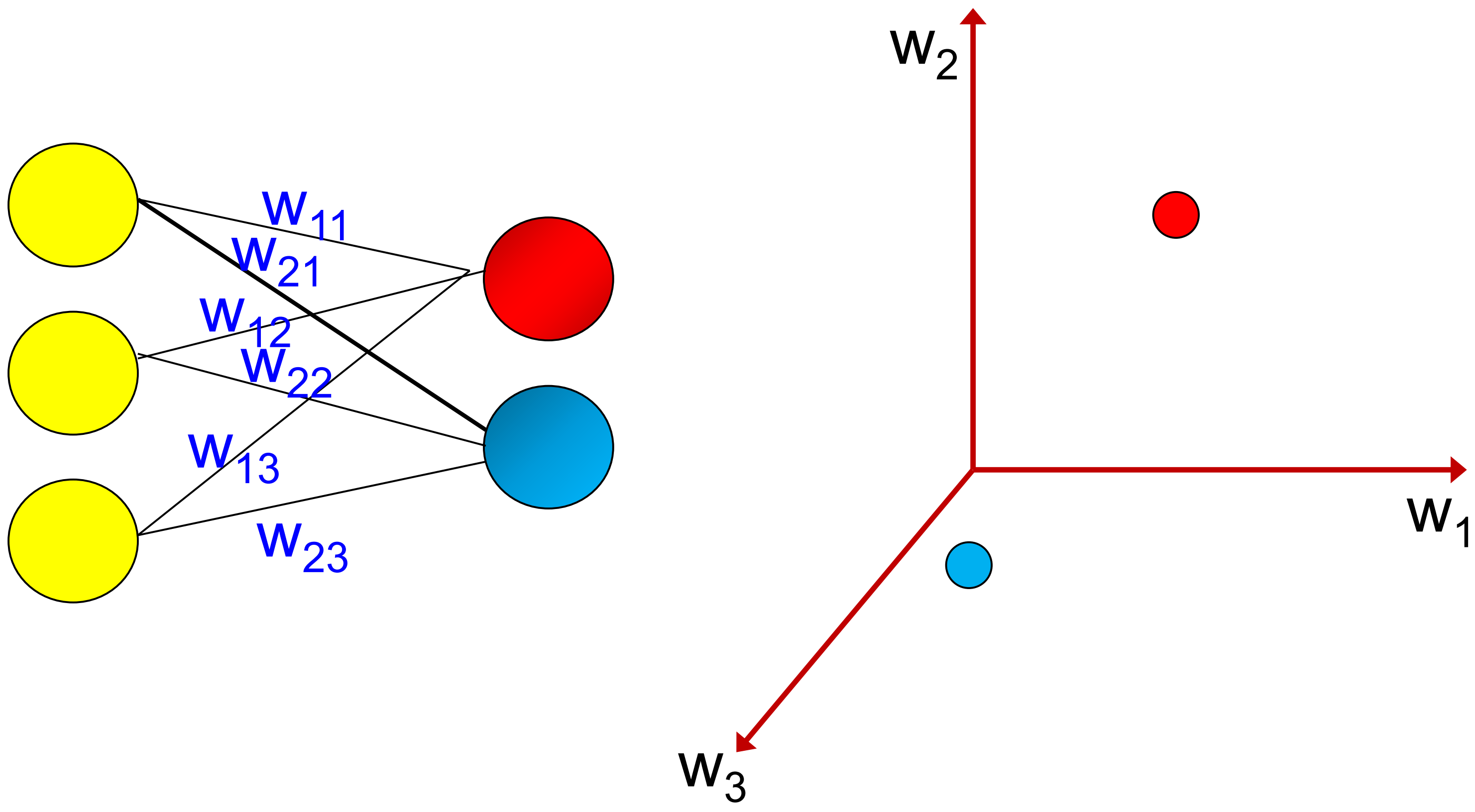
How biologically (un)realistic?

- The weight w_j can be positive or negative
- A unit can become inhibitory or excitatory, or both
- Use only a linear sum of inputs
- Use a continuous output instead of pulses (spike train)
- No refractory period

Some Terminology

Term	*Typical Symbol*	*Alternate Term(s)*
Input vector	$\boldsymbol{x}$	*input activation*
Weights	w_{ij}	*synaptic weights, parameters*
Inner activation	h	*net activation*
Activation function	g	*transfer function; threshold function*
Output	y	*(outer) activation; prediction*
Target	t	*teacher value*
Error	E	cost

Weight Space: Represent a Unit with its Incoming Weights



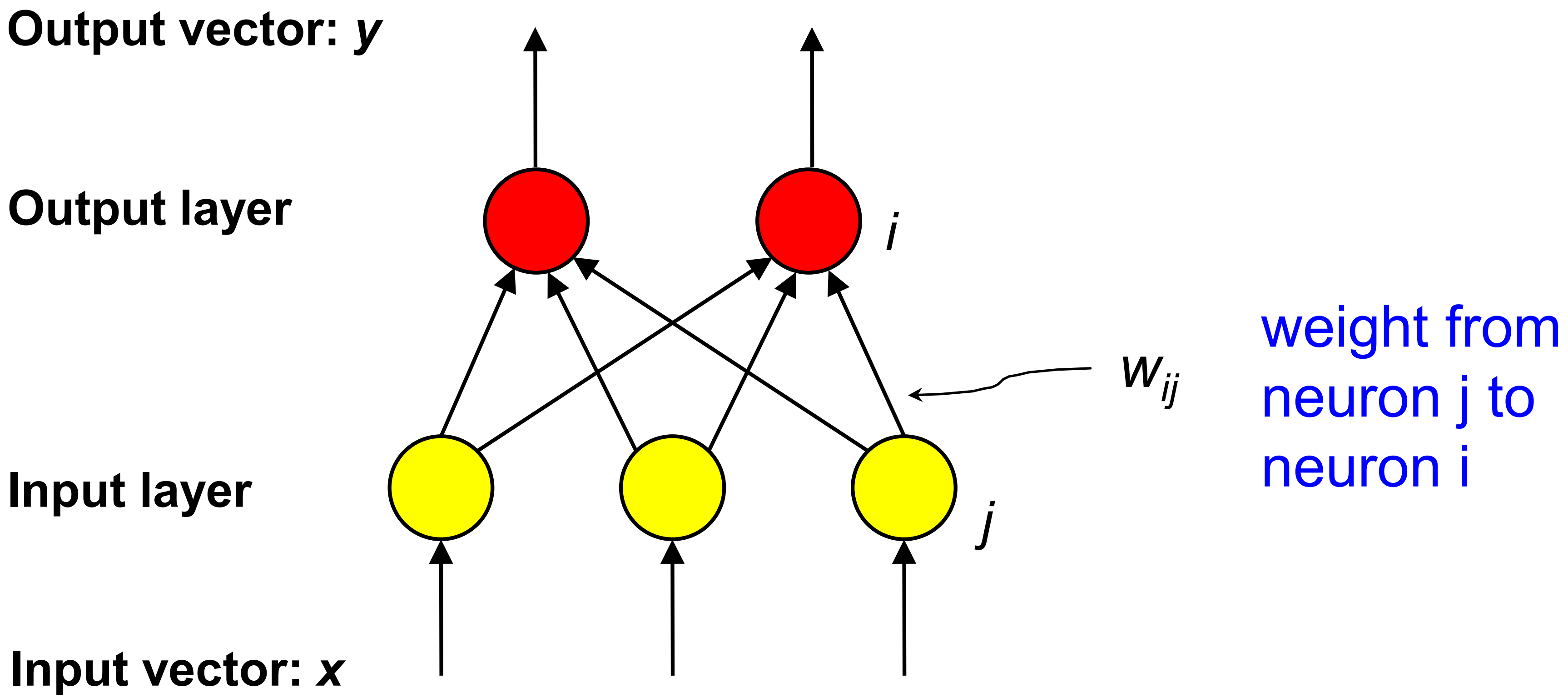
Neural Networks

- Started by psychologists and neurobiologists as computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a ***weight*** associated with it
- During supervised learning, the ***network learns by adjusting the weights*** so as to be able to predict the correct class label of the input tuples
- Also referred to as ***connectionist learning*** due to the connections between units

Overview

- Biological Background
- Perceptron
  - Perceptron Layer
  - Linear Separability

Perceptron Network



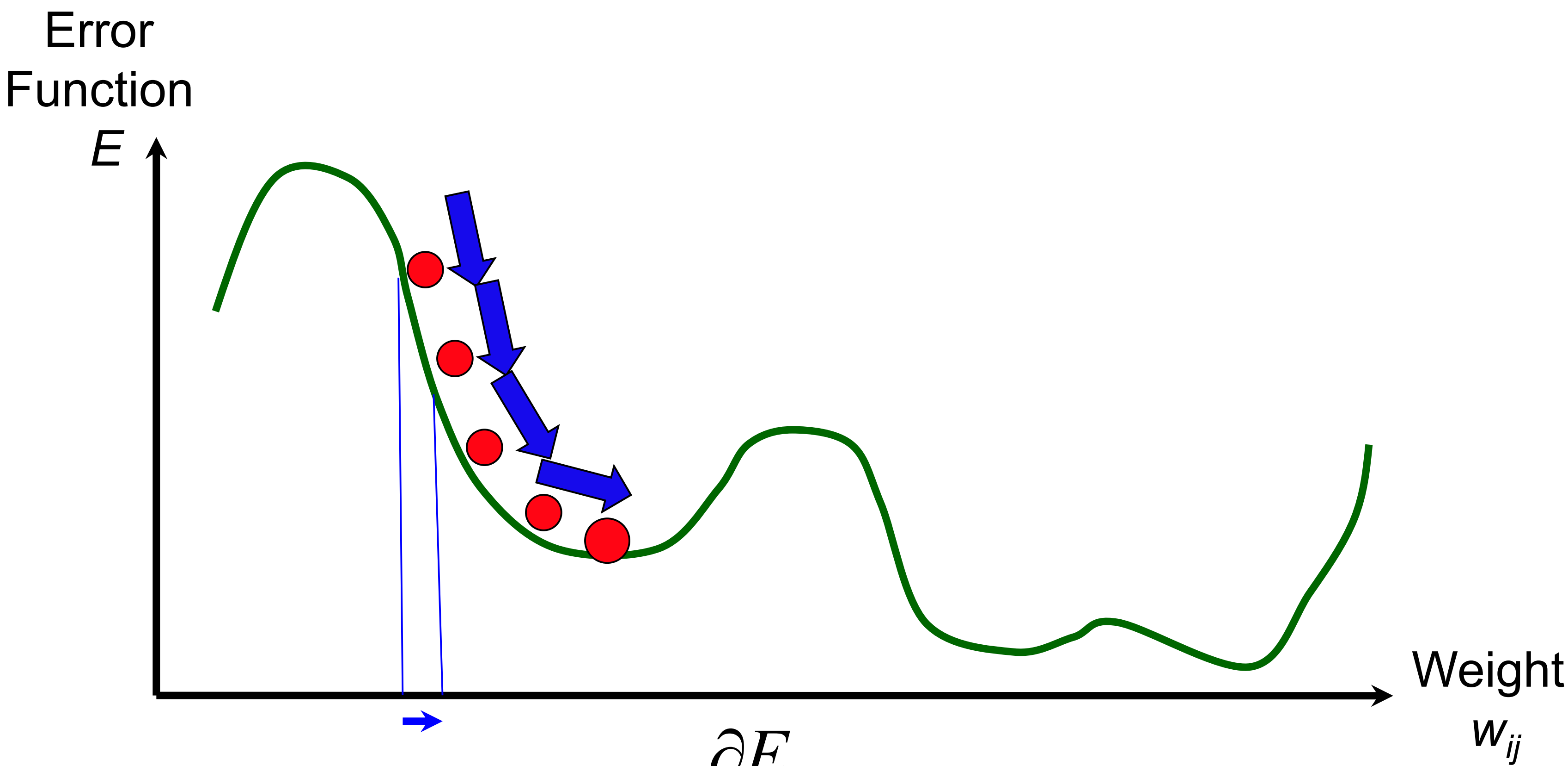
Updating the Weights

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

- We want to change the values of the weights
- Task: function approximation
  - We have training data: pairs of input vectors **x** and corresponding output vectors ***t*** (teacher/target values)
  - For each input ***x***, aim is to ***minimize the error*** between the corresponding teacher values ***t*** and the network outputs ***y***
  - This is supervised learning
- For systematic error minimization, define an always positive error function:

$$E = (t - y)^2$$

Gradient Descent



$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

We differentiate *E* to obtain the gradient.
The negative gradient is the direction of change.

An Error Function

- Sum-of-squares error

square ensures that pos. and neg. errors don't cancel out

$$E(\mathrm{w}) = \frac{1}{2}\sum_{data}\sum_{i}\left(t_i - y_i\right)^2$$

- Let's assume linear neurons: (no threshold function)

$$y_i = h_i = \sum_j w_{ij} x_j$$

- Gradient descent: $\Rightarrow$

$$-\frac{\partial E}{\partial w_{ij}} = \sum_{data}\left(t_i - y_i\right)\cdot x_j$$

- Resulting rule for the weights:

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

change of weight

learning rate

teacher

output

input

How Did We Get That Gradient?

- How to derive the term $E = \frac{1}{2}\,(t - w \cdot x)^2$ with respect to w?
- Chain rule: $d/dw\; f(g(w)) = df/dg \cdot dg/dw$ where we have:
  - $g(w) = (t - w \cdot x)$
  - $f(g) = \frac{1}{2}\, g^2$
- Derivative of n^th power: $d/dg\; g^n = n \cdot g^{n-1}$ (here, n=2)
  - $d/dg\; \frac{1}{2}\, g^2 = g$
  - $d/dw\; (t - w \cdot x) = -x$
- Together: $-\, d/dw\; E = g \cdot x = (t - w \cdot x) \cdot x$

Perceptron Algorithm

- Initialization: set all weights to small positive and negative random numbers
- For #iterations
  - Chose a new data point $(\boldsymbol{x}, \boldsymbol{t})$
  - Compute the output activation y_i of each neuron i

$$h_i = \sum_{j=1}^{n} w_{ij} x_j \qquad y_i = \begin{cases} 1 & h_i \geq \theta \\ 0 & h_i < \theta \end{cases}$$

  - Update each of the weights according to

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

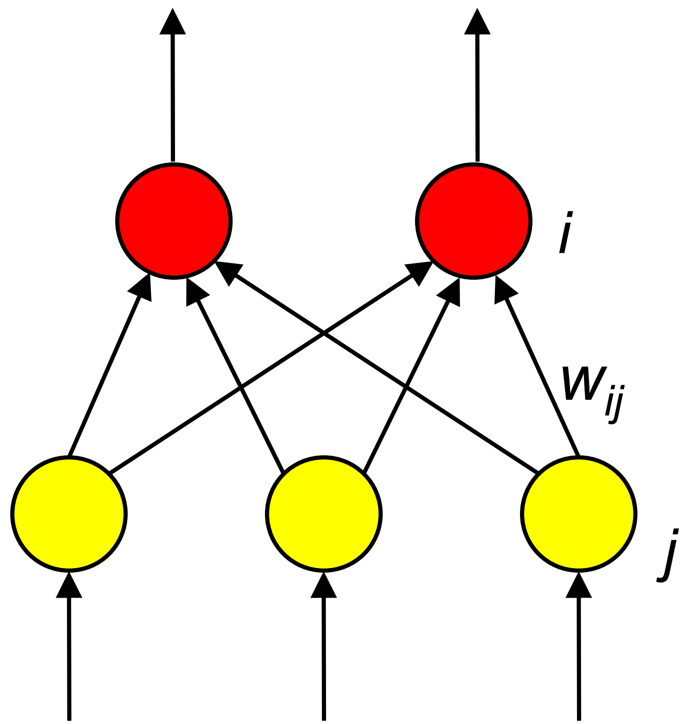
Perceptron Network

Teacher vector: ***t***

Output vector: ***y***

Output layer

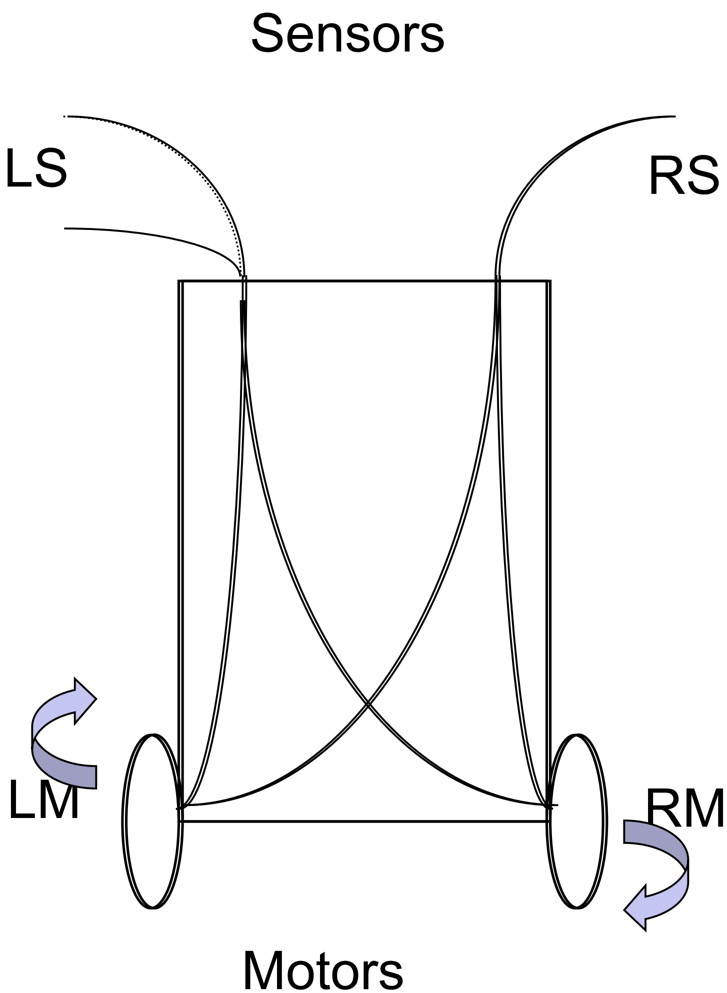
Input layer

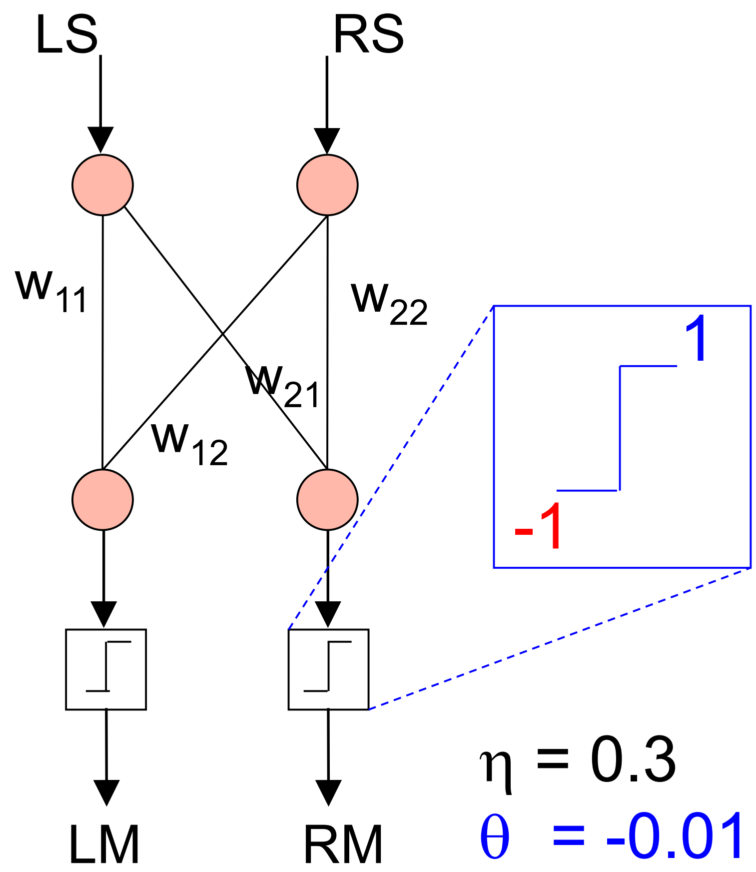


$$\boxed{w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta(t_i - y_i)x_j}$$

Input vector: ***x***

Obstacle Avoidance with the Perceptron





$\eta = 0.3$

$\theta = -0.01$

$w_{ij} = 0$

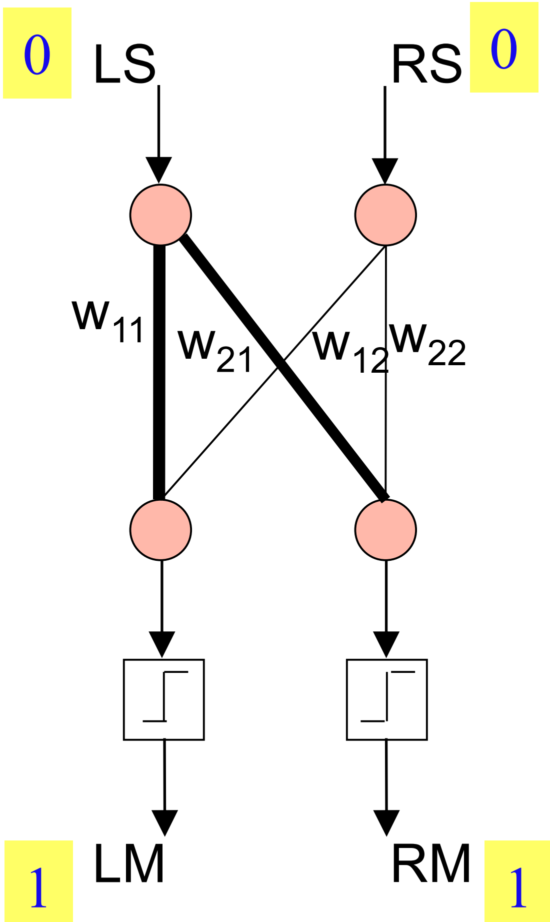
all outputs are 1

Obstacle Avoidance with the Perceptron: Behaviour we Want

Training data

LS	RS	LM	RM
0	**0**	**1**	**1**
0	1	-1	1
1	0	1	-1
1	1	X	X

Obstacle Avoidance with the Perceptron



Assume initial weights are 0.
We will see:
No update if target = actual output

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$w_{11} = 0 + 0.3 \cdot (1 - 1) \cdot 0 = 0$$

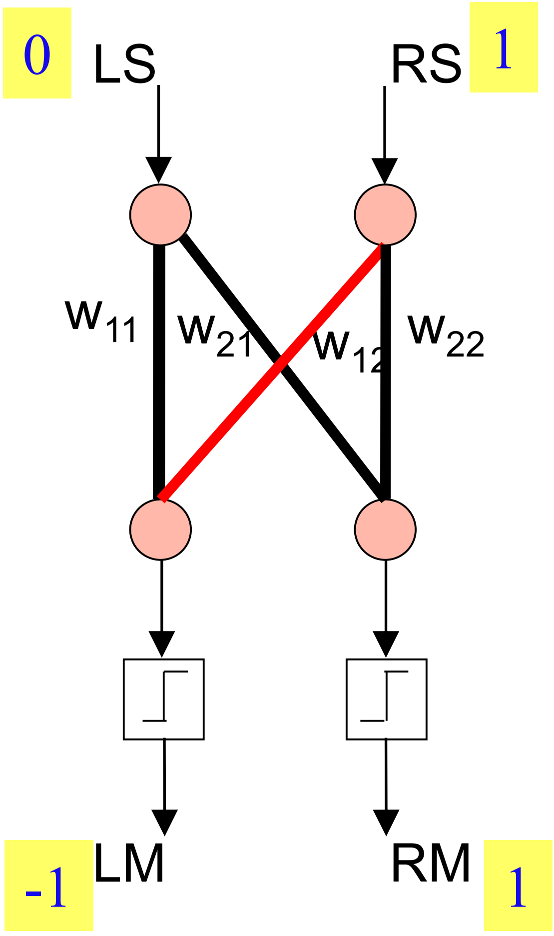
$$w_{21} = 0 + 0.3 \cdot (1 - 1) \cdot 0 = 0$$

And the same for w_{12}, w_{22}

Obstacle Avoidance with the Perceptron

LS	RS	LM	RM
0	0	1	1
0	**1**	**-1**	**1**
1	0	1	-1
1	1	X	X

Obstacle Avoidance with the Perceptron



w_{12}: the robot turns left by reversing the left motor

We will see:
No update if input = 0

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$w_{11} = 0 + 0.3 \cdot (-1 - 1) \cdot 0 = 0$$

$$w_{21} = 0 + 0.3 \cdot (1 - 1) \cdot 0 = 0$$

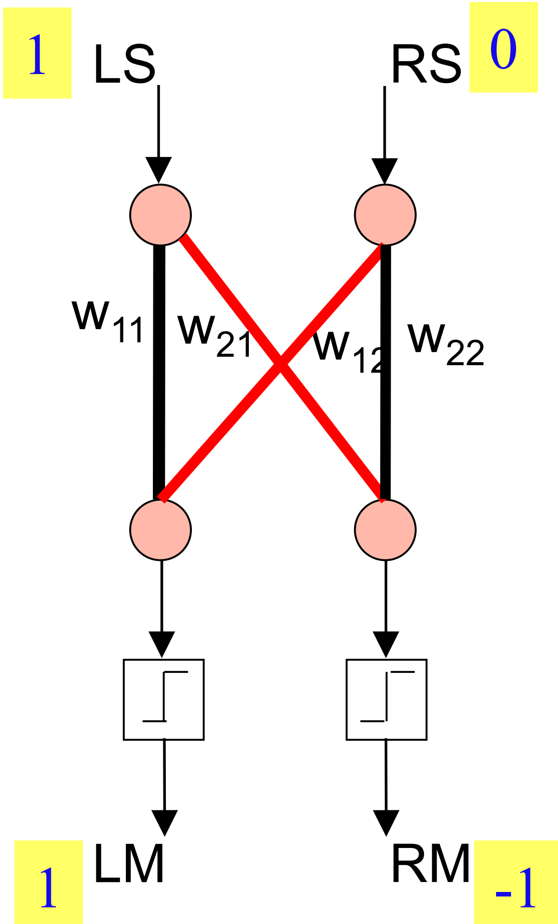
$$w_{12} = 0 + 0.3 \cdot (-1 - 1) \cdot 1 = -0.6$$

$$w_{22} = 0 + 0.3 \cdot (1 - 1) \cdot 1 = 0$$

Obstacle Avoidance with the Perceptron

LS	RS	LM	RM
0	0	1	1
0	1	-1	1
1	**0**	**1**	**-1**
1	1	X	X

Obstacle Avoidance with the Perceptron



$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

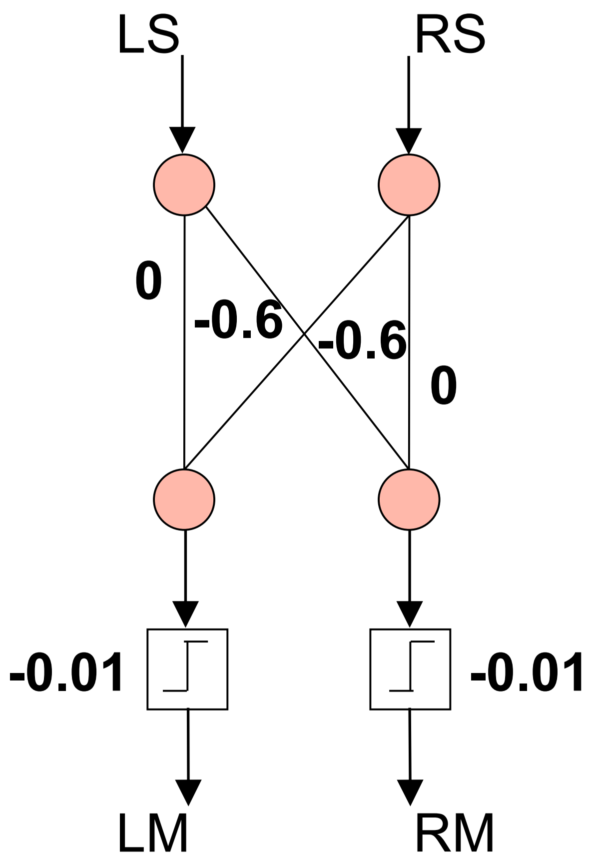
$$w_{11} = 0 + 0.3 \cdot (1 - 1) \cdot 1 = 0$$

$$w_{21} = 0 + 0.3 \cdot (-1 - 1) \cdot 1 = -0.6$$

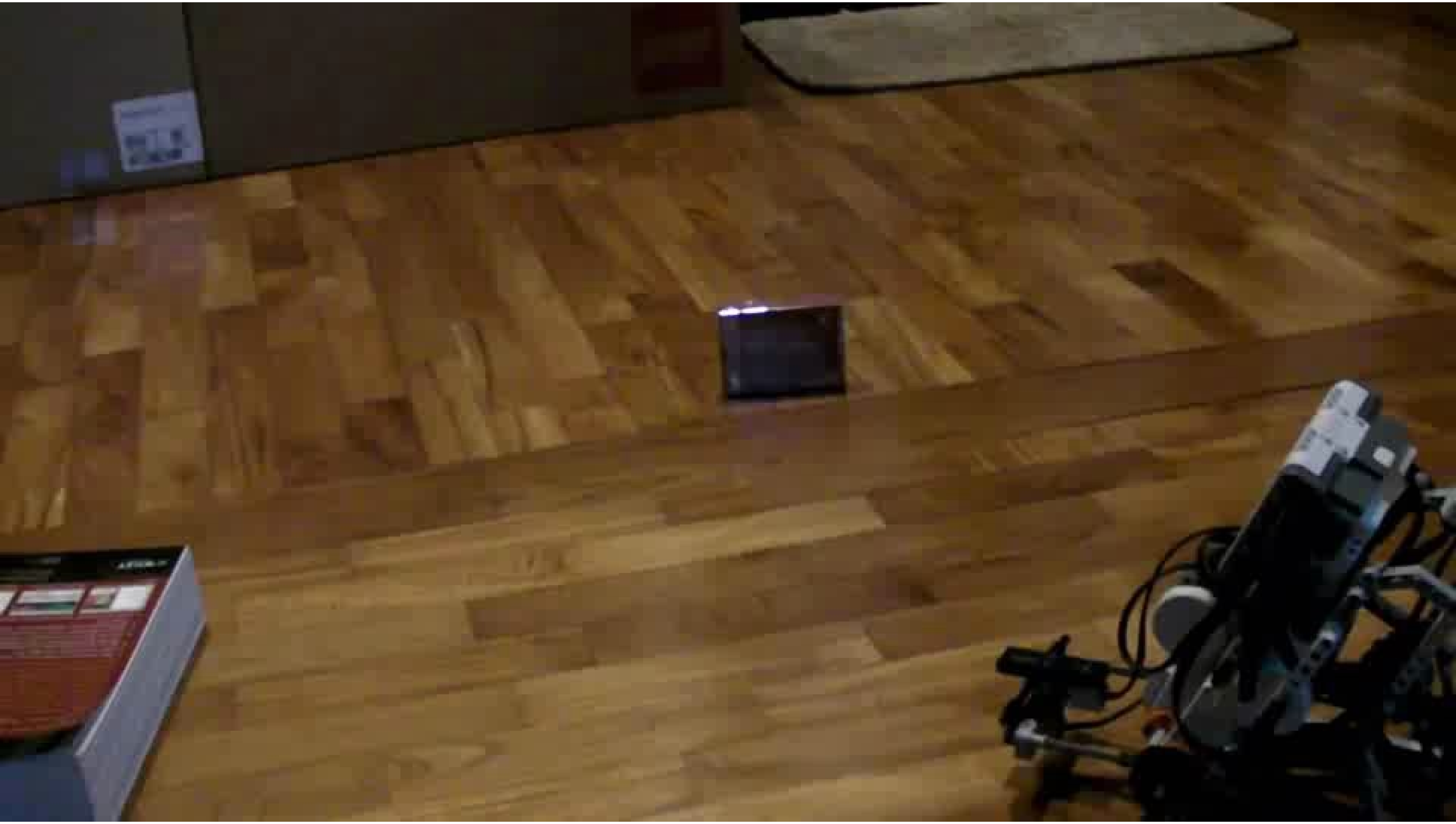
$$w_{12} = -0.6 + 0.3 \cdot (1 - 1) \cdot 0 = -0.6$$

$$w_{22} = 0 + 0.3 \cdot (-1 - 1) \cdot 0 = 0$$

Obstacle Avoidance with the Perceptron



Obstacle Avoidance with a Mindstorm Vehicle



Overview

- Biological Background
- Perceptron
  - Perceptron Layer
  - Linear Separability

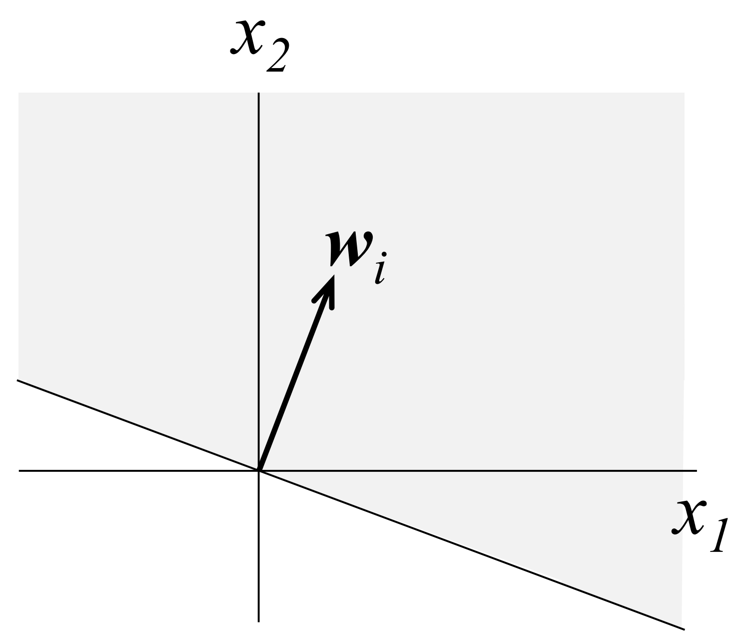
Linear Separability

- Outputs are:

dot product

$$y_i = \mathrm{sign}(\mathrm{h_i}), \ \text{where}\ \mathrm{h}_i = \sum_{j=1}^{n} w_{ij} x_j = |w_i| \cdot |x| \cdot \cos(w_i, x)$$

  - Positive output +1 if:
    $$w_i \cdot x > 0$$
  - Negative output -1 if:
    $$w_i \cdot x < 0$$
  - Output = 0 if $w_i \cdot x = 0$



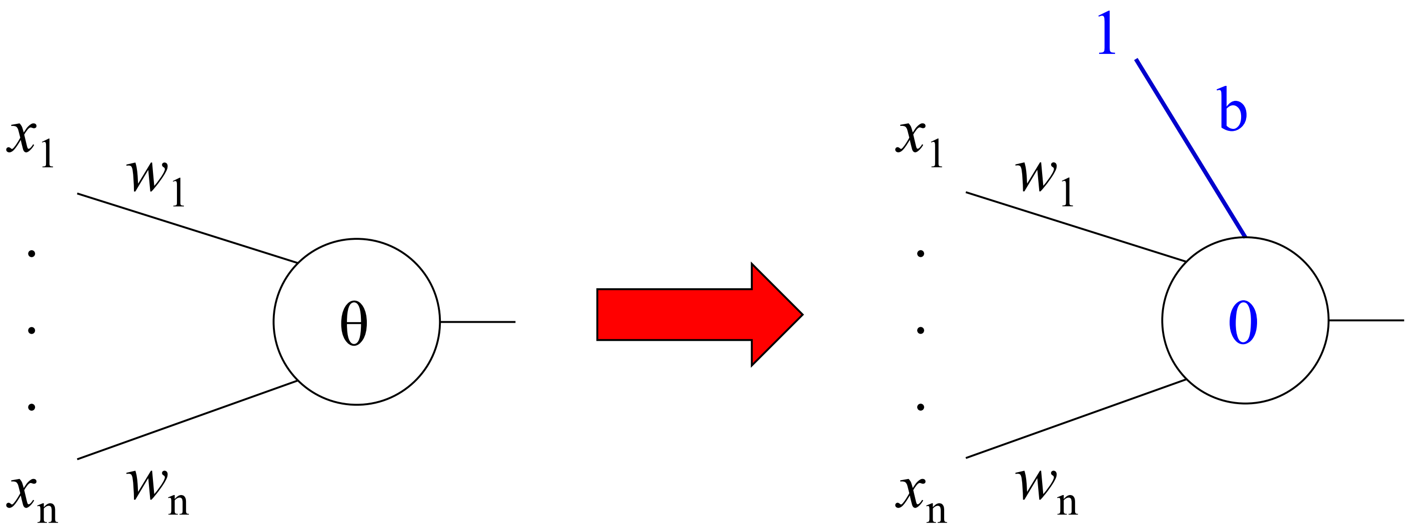
The region in input space where $\boldsymbol{x}$ yield positive output y is a half-plane.

Bias *b*

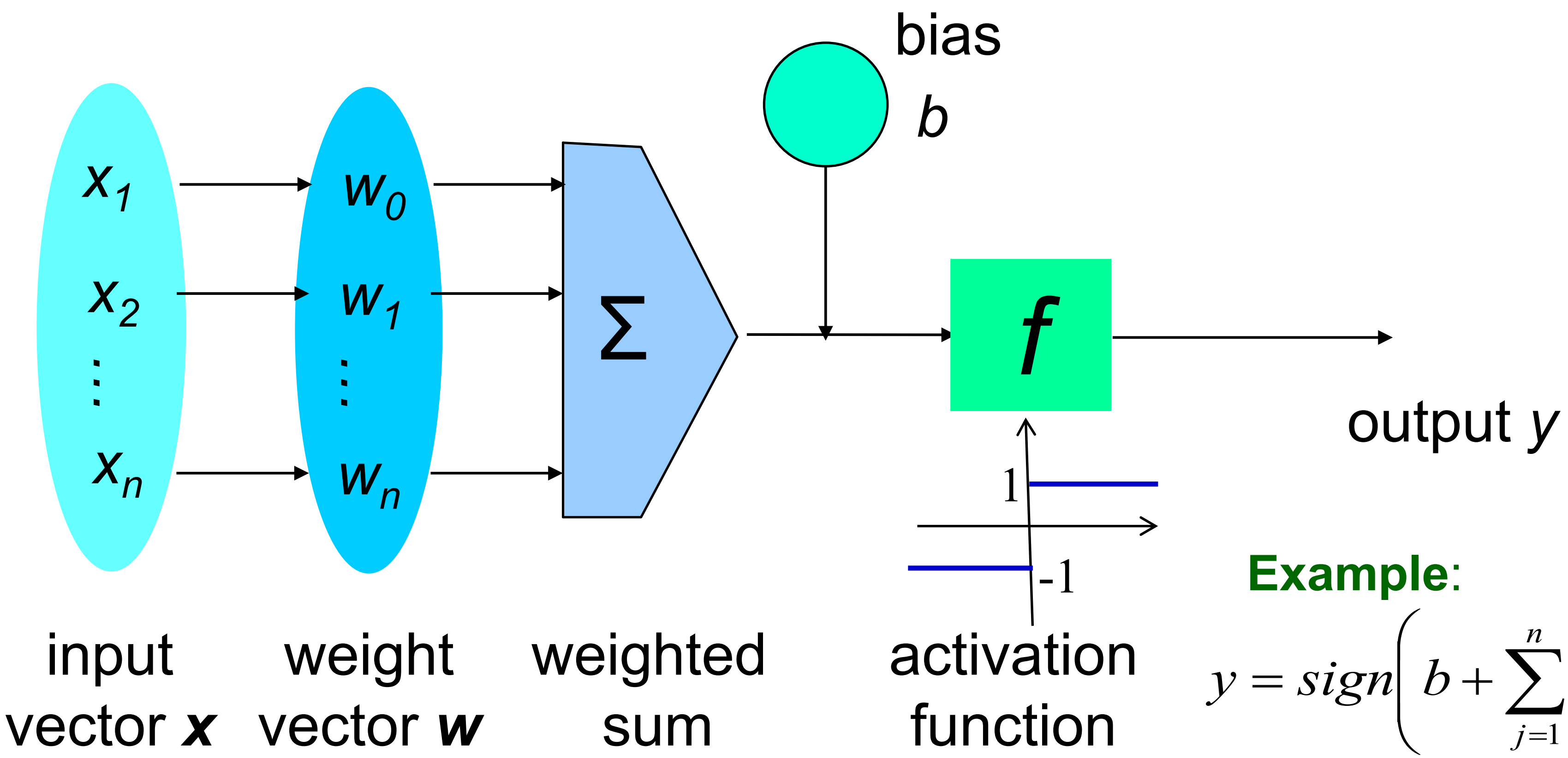
- An extra input – adds to (b > 0) or subtracts from (b < 0) the net input

$$y = sign\left(\boxed{b} + \sum_{j=1}^{n} w_j x_j \right)$$

- If b < 0, acts like a positive threshold Θ = -b
  - the weighted input must be larger than Θ for the neuron to activate



Perceptron with Bias

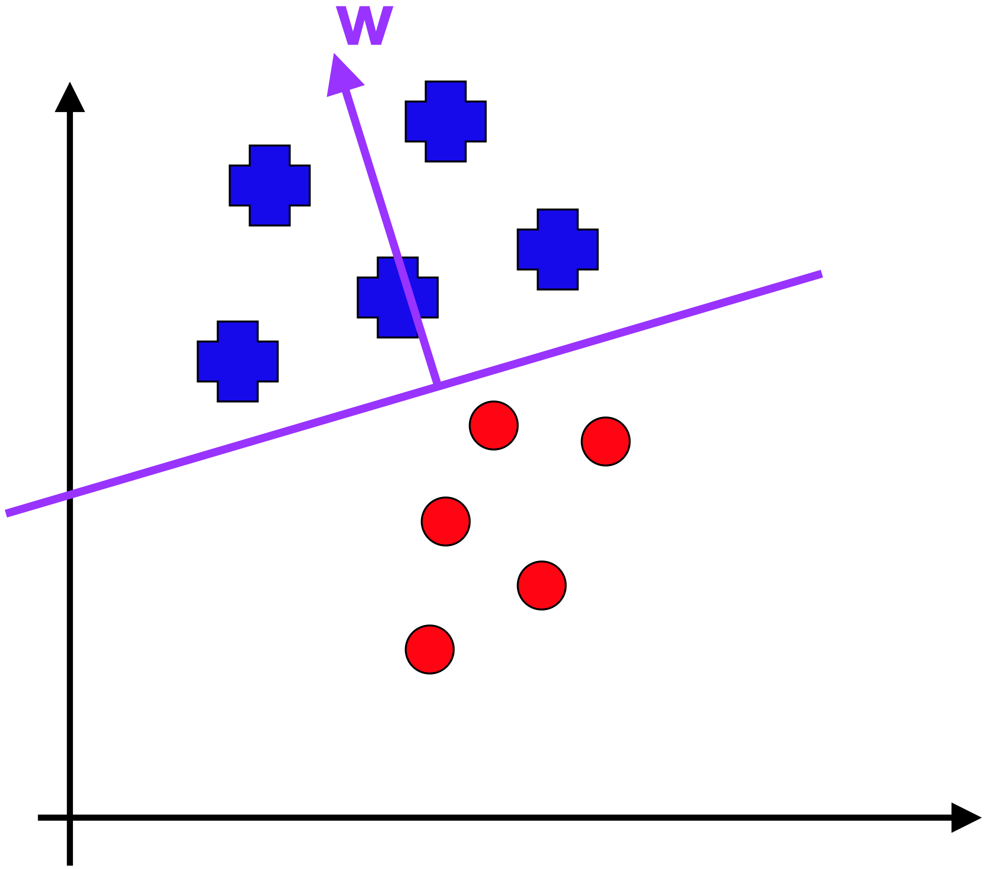


Example:

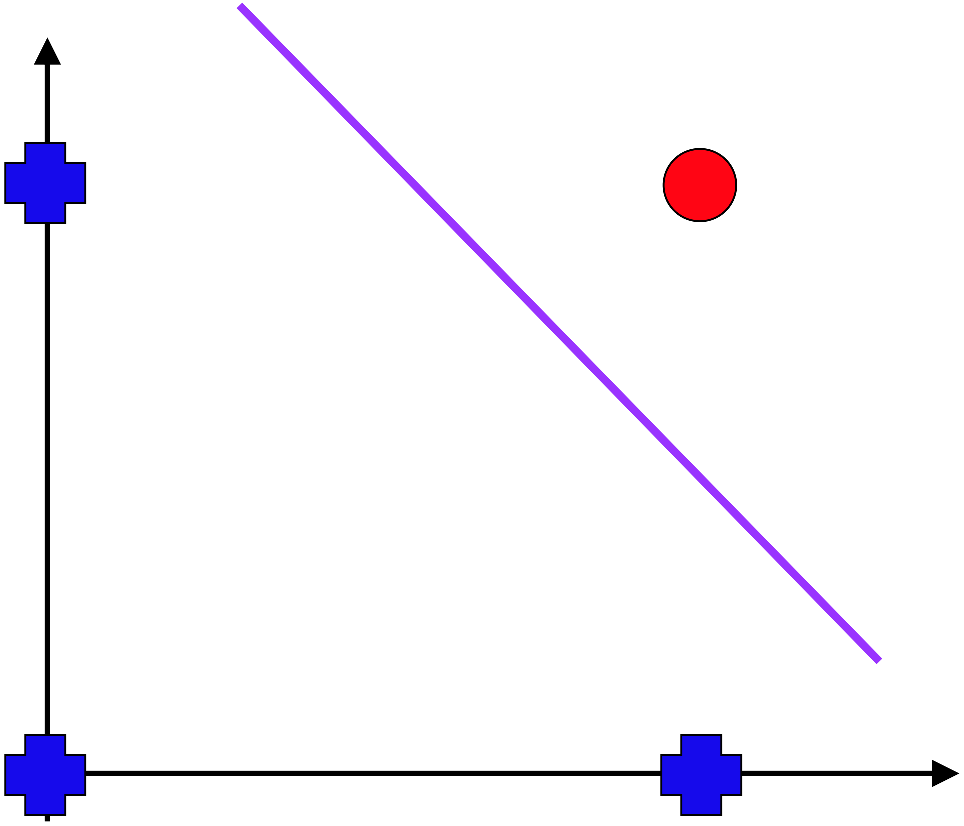
$$y = sign\left(b + \sum_{j=1}^{n} w_j x_j\right)$$

The n-dimensional input vector $\boldsymbol{x}$ is mapped into a variable y by means of the dot product with weights $\boldsymbol{w}$ and addition of bias b, and a nonlinear function mapping

Linear Separability



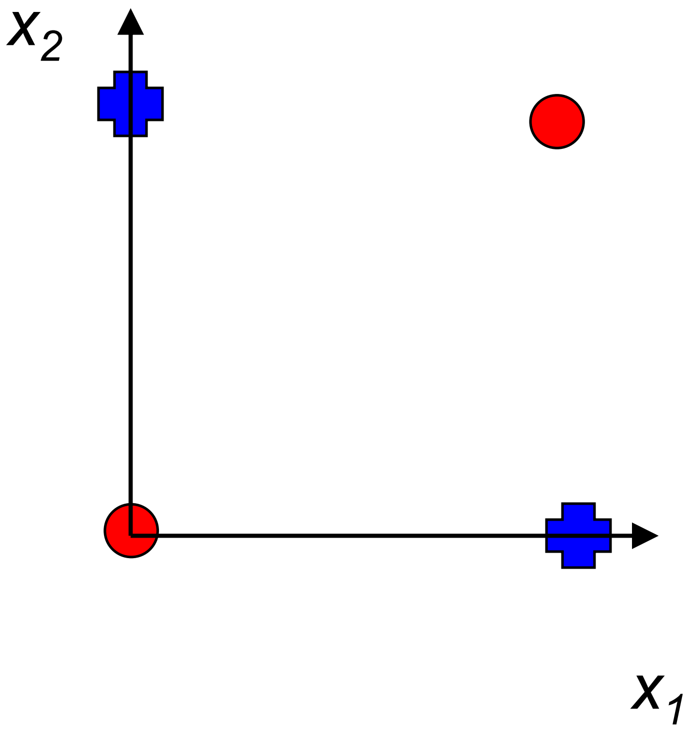
Linear Separability



The Binary
AND Function

Limitations of the Perceptron

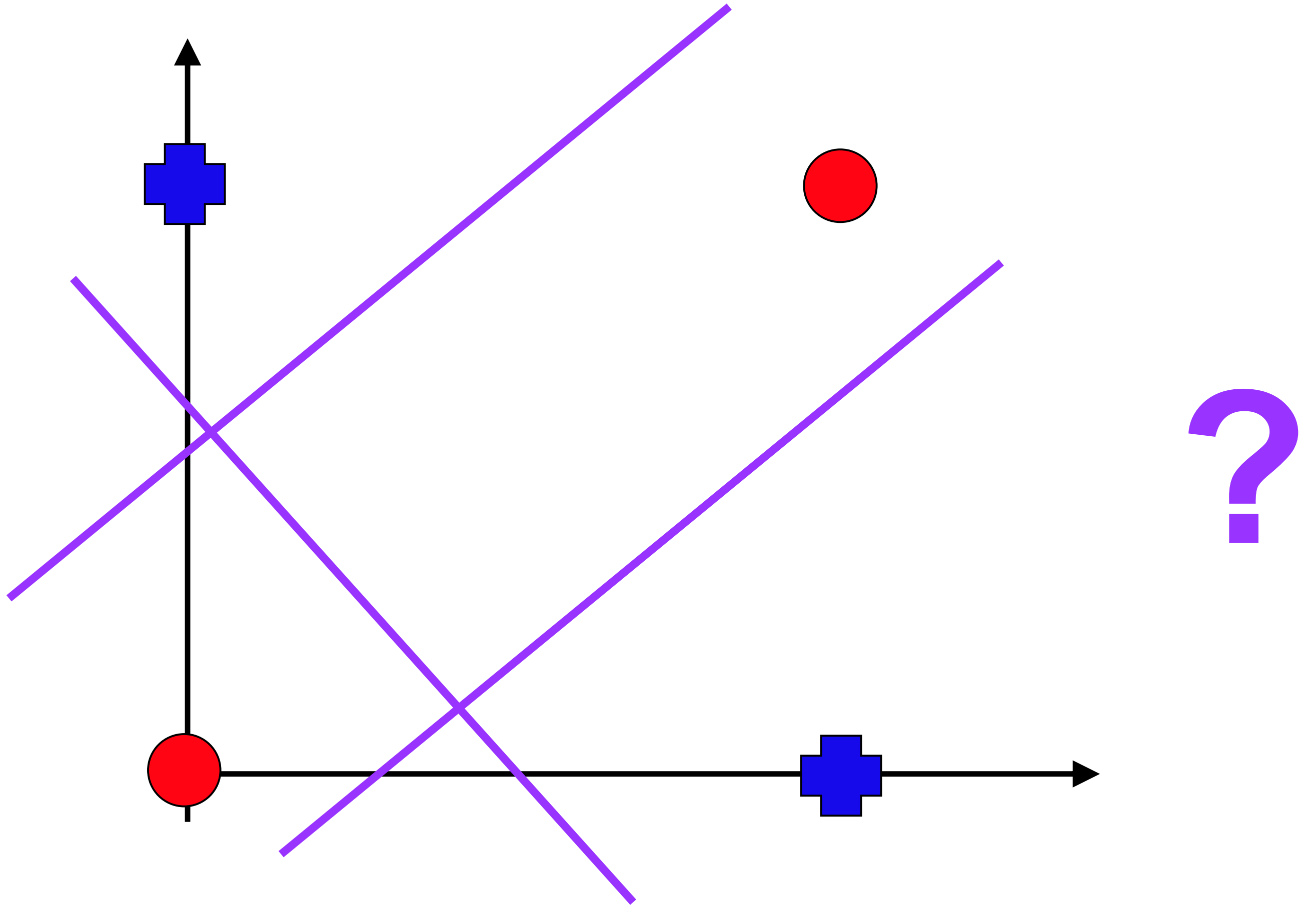
Linear Separability?



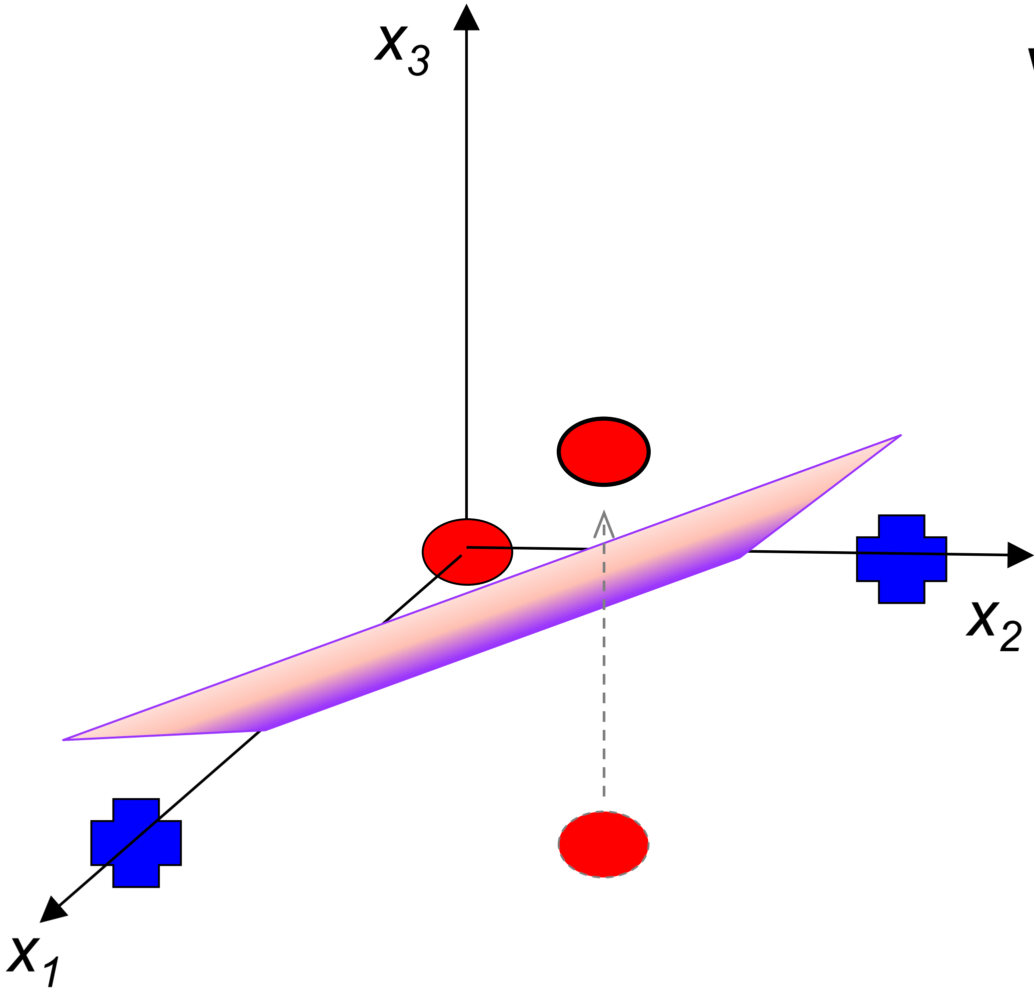
Exclusive Or (XOR) function

x_1	x_2	**Out**
0	0	0
0	1	1
1	0	1
1	1	0

Limitations of the Perceptron



Limitations of the Perceptron



Ways around the problem:

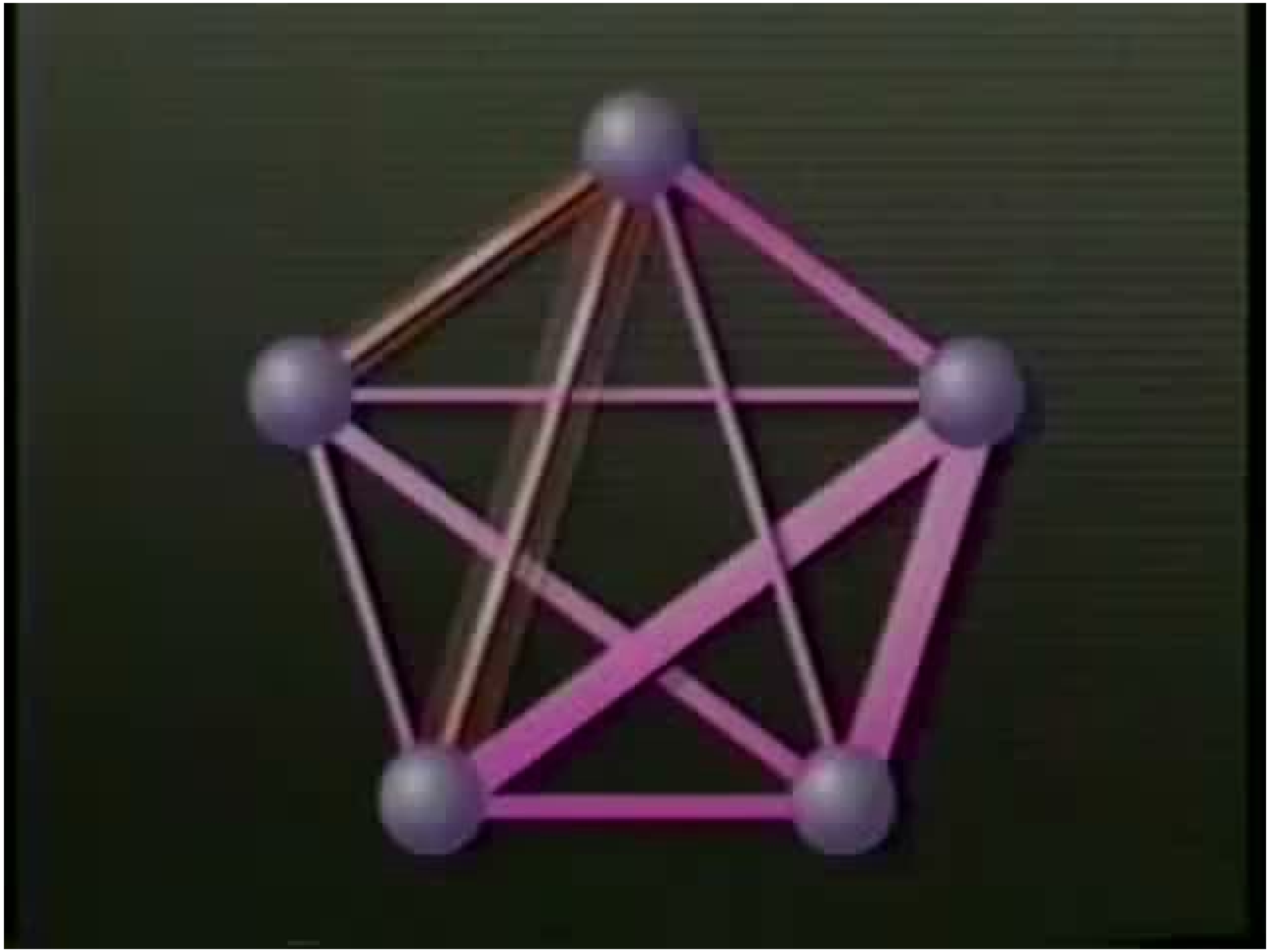
1. Use a more complex input set,

   e.g. $\boldsymbol{x} = (x_1, x_2, x_3)$

   with $x_3 = x_1 \cdot x_2$

2. Use a more complex network.

Perceptrons – Early Successes(?)



Overview

- Biological Background
- Perceptron
  - Perceptron Network
  - Linear Separability
- Multilayer Perceptron