ELTE | FACULTY OF INFORMATICS

Neural networks

Introduction to Machine Learning – Lecture 10
Computer Science BSc Course, ELTE Faculty of Informatics

János Botzheim

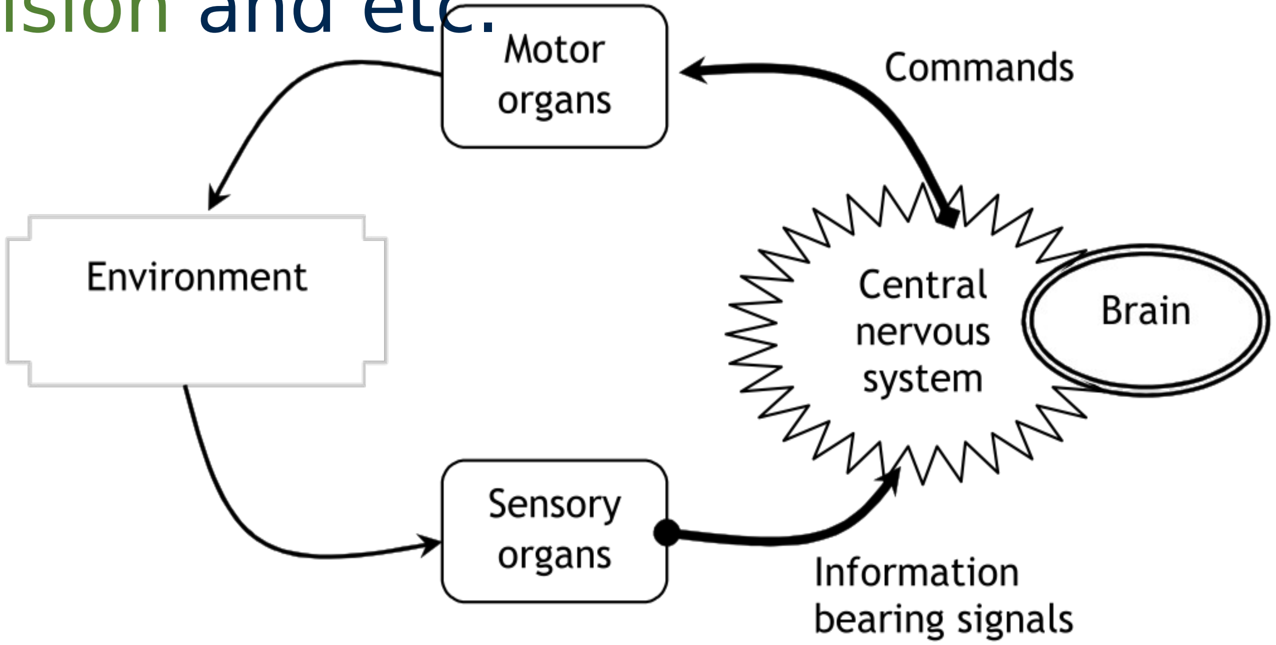
Associate Professor
Department of Artificial Intelligence
botzheim@inf.elte.hu

ELTE | IK

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

Human nervous system

- Human nervous system is a very complex system capable of thinking, remembering, problem solving, making decision and etc.

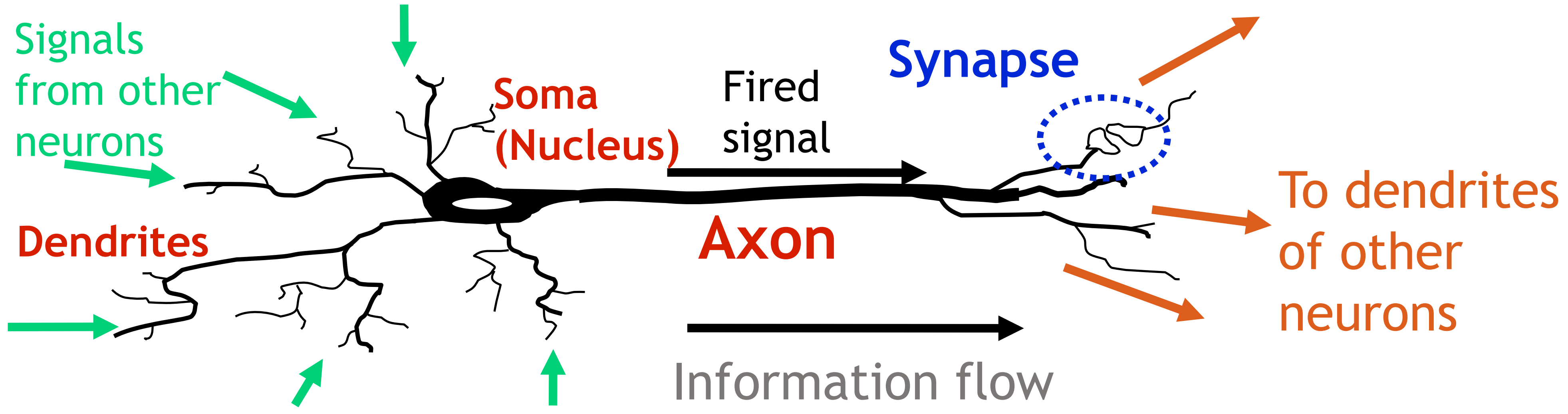


Few data

- The fundamental cellular unit of nervous system is called neuron.
- A baby is born with about 10^{11} neurons in the brain.
- Neuron is a simple processing unit connected to approximately 1000 neurons. Therefore, there are approximately 10^{14} connections or transmission paths in the nervous system.
- The estimated memory capacity of the human brain varies between 1 and 1000 terabytes (for comparison, the US Library of Congress currently stores around 10 terabytes of data).

Biological neurons

- The function of a neuron is receiving and combining signals from other neurons through input paths called dendrites.
- If the combined input signal is strong enough, the neuron fires and produces an output signal and sends the output along the axon to other neurons.

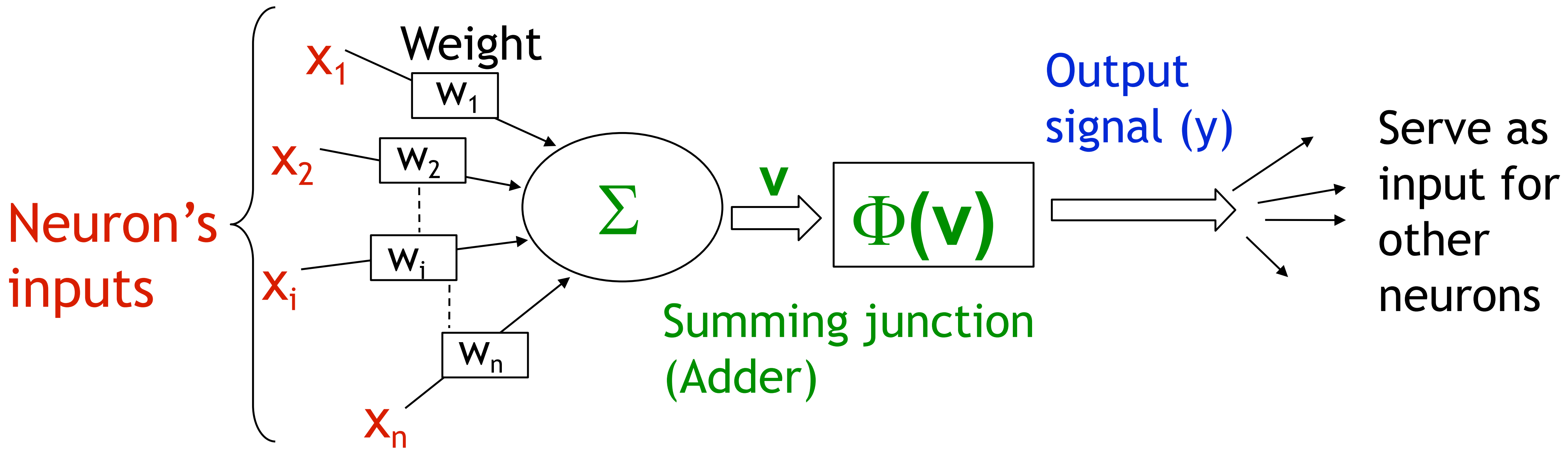


Computer vs. human nervous system

	Computer	Human
Speed	Above 4 GHz	40-50Hz (Neuron response time - 20-30 ms)
Operation mode	Serial mode	Parallel mode (Low speed of brain is compensated by high processing speed inside the brain)
Recognizing a person after seeing	A complicated task, depends on the background, viewing point, etc.	Half a second

Artificial neuron

- An information processing unit that is fundamental of an artificial neural network.
- A mimicking model for biological neuron.



Artificial neuron

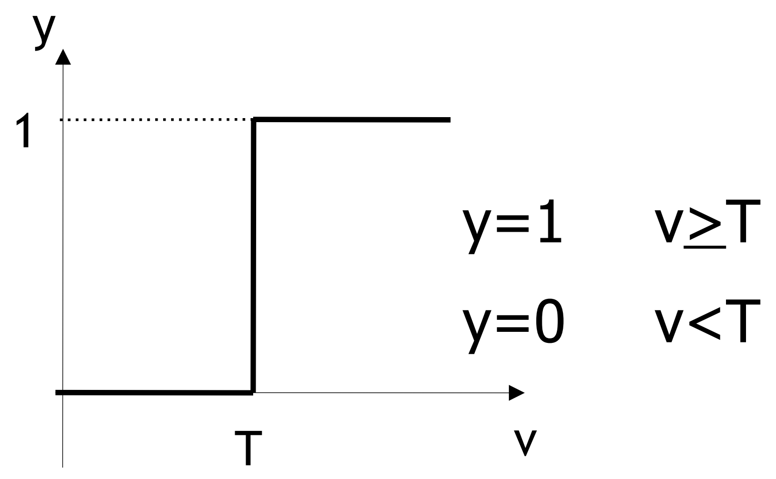
- A neuron can only perform some simple information processing; therefore it is not able to solve complex problems.
- However, neurons can be interconnected to deliver collective complex behavior and thus they can be used for solving complex problems.

Biological vs. artificial neuron

Biological neuron	**Artificial neuron**
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

Activation function (Φ)

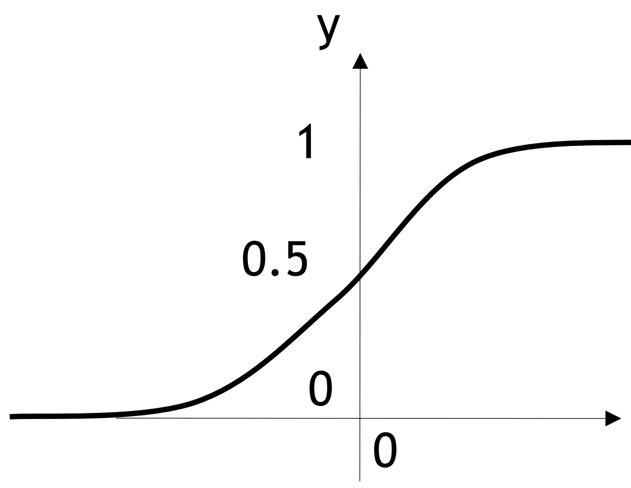
- Limits the output between two asymptotes so that it can keep neuron output within reasonable dynamic range



$y=1 \quad v \geq T$

$y=0 \quad v < T$

Threshold or step Function



$y = 1/(1+e^{-\alpha v})$

α: adjusts the abruptness of the function

Sigmoid function

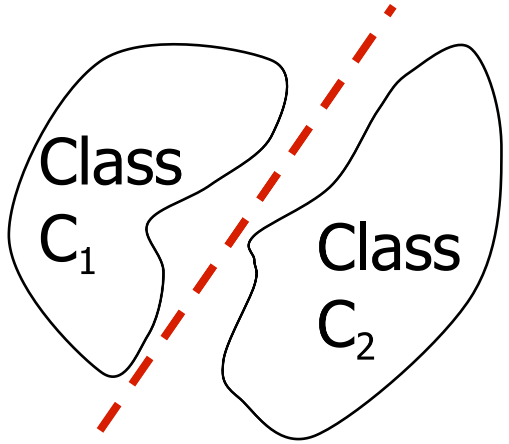
Single layer perceptron

- Single layer perceptron was proposed by Rosenblatt in 1958 as the 1st model for learning with a teacher (i.e., supervised learning).
- His perceptron was based on McCulloch-Pitts model of neuron.

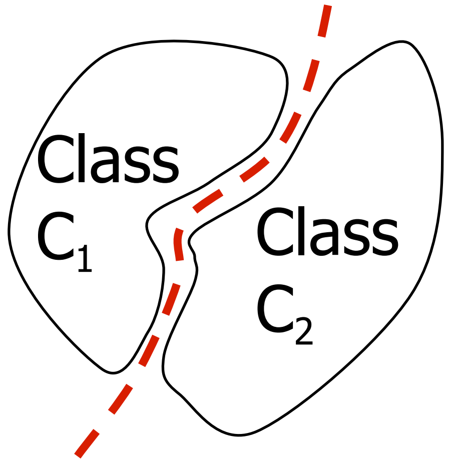
Single layer perceptron

- Used for solving linearly separable patterns.
- This caused serious disappointment in the neural network community and put neural network researchers in a dark period (1970s).
- However, this statement was not correct in general, because multilayer perceptrons are capable of separating linearly inseparable variables (including XOR problem).

linearly separable patterns

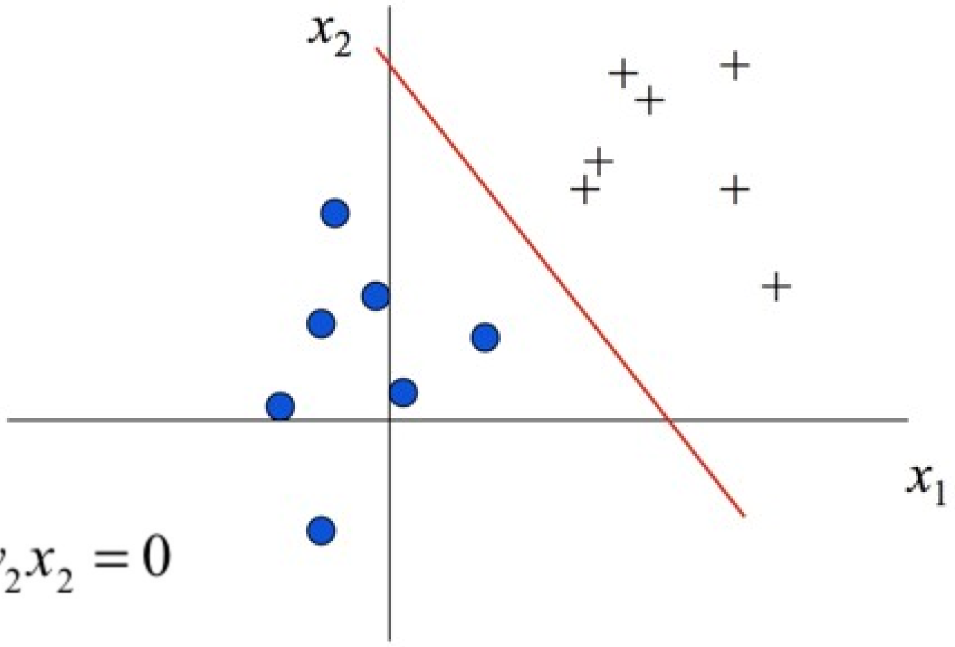


nonlinearly separable patterns



Linear separability

- Consider the following example with two inputs (x_1,x_2)
- Can the trained network define the "separation line"?

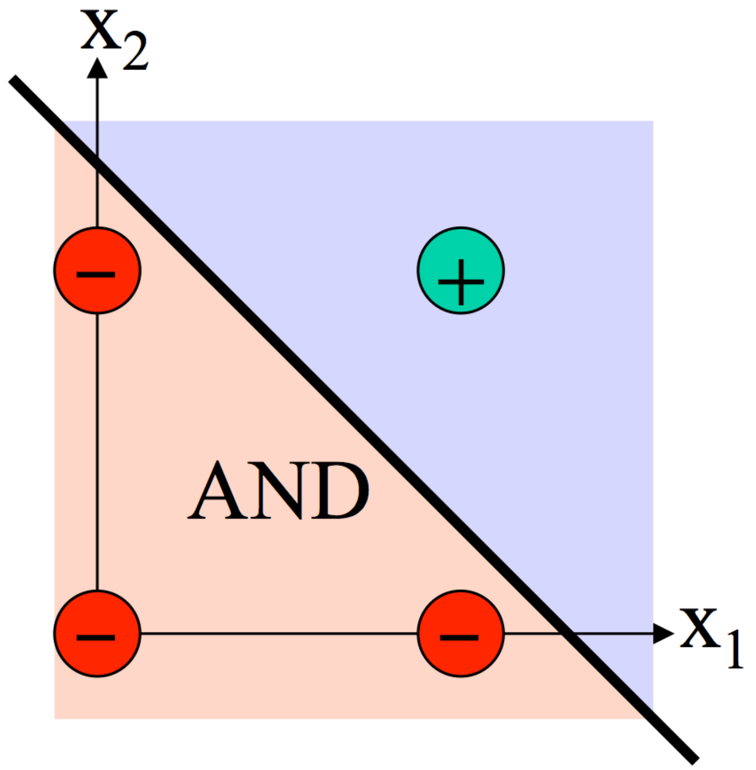


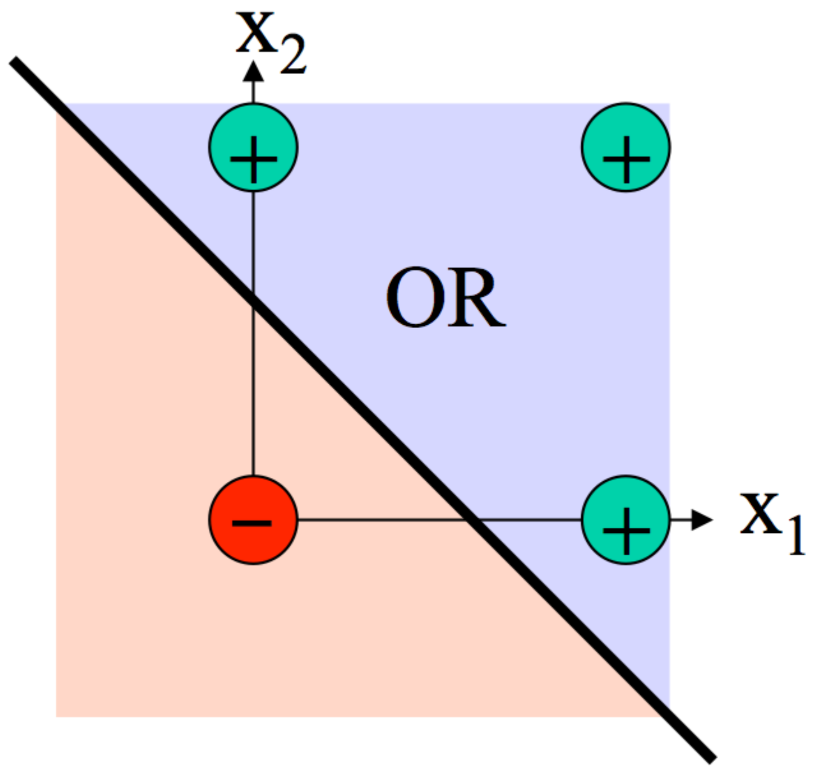
$$-w_0 + w_1 x_1 + w_2 x_2 = 0$$

- What is its equation?

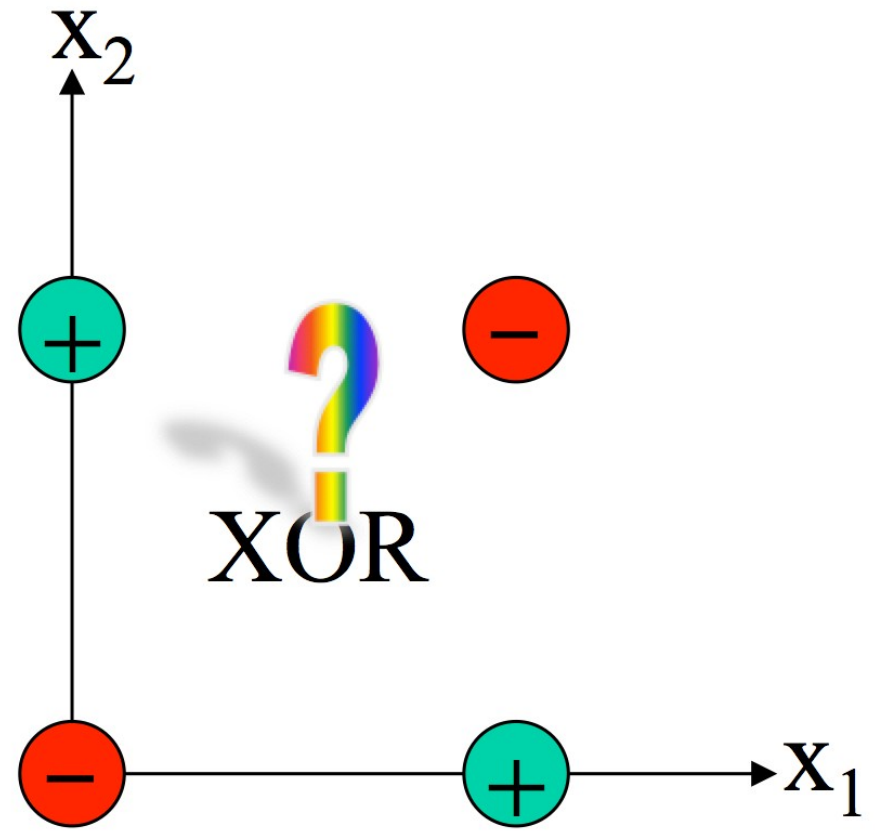
$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{w_0}{w_2}$$

Linear separability

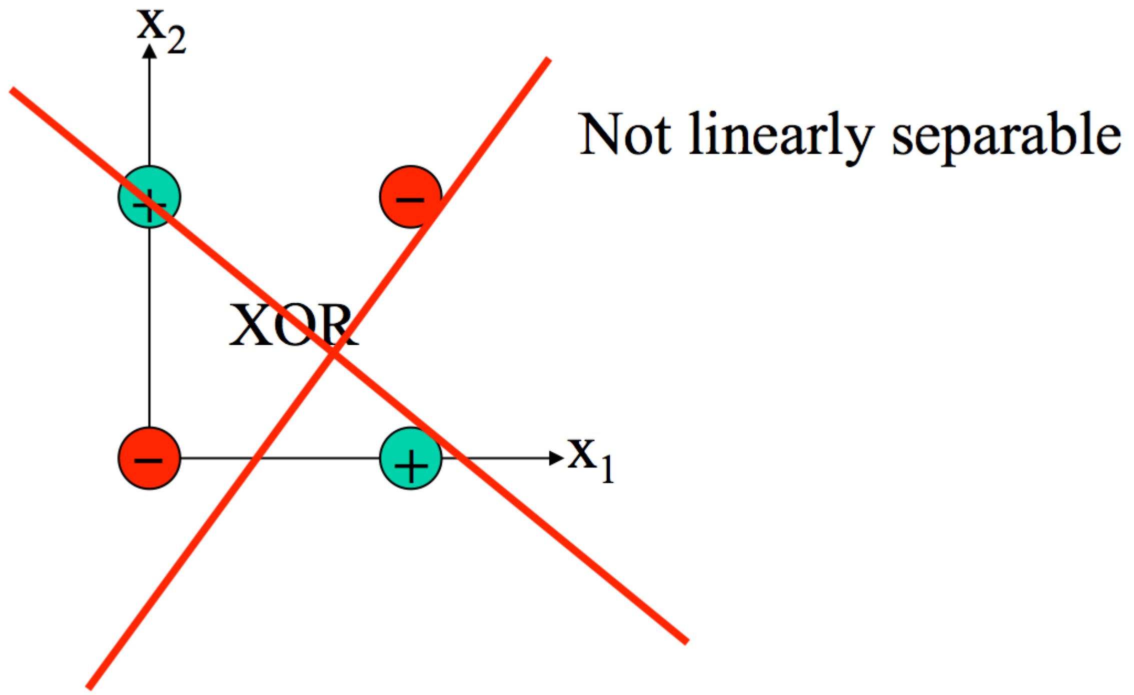




Linear separability



Linear separability



Minsky & Papert (1969)

Bad News: Perceptrons can only represent linearly separable functions.

Implementing logic gates

- We can use McCulloch-Pitts neurons to implement the basic logic gates
- All we need to do is find the appropriate connection weights and neuron threshold to produce the right outputs for each set of inputs
- We shall see explicitly how one can construct simple networks that perform NOT, AND, and OR.
- It is then a well known result from logic that we can construct any logical function from these three operations
- The resulting networks, however, will usually have a much more complex architecture that a simple Perceptron
- We generally want to avoid decomposing complex problems into simple logic gates, by finding the weights and thresholds that work directly in a Perceptron architecture

Implementation of logical NOT, AND, and OR

- In each case we have inputs *in*$_i$ and output *out*, and need to determine the weights and thresholds

NOT

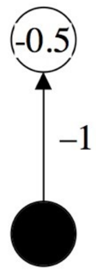
in	*out*
0	1
1	0

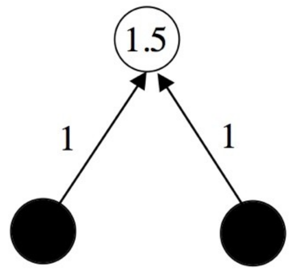
AND

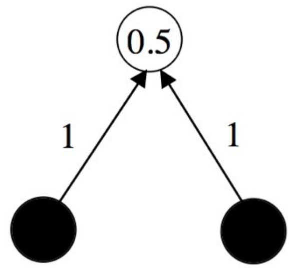
in_1	in_2	*out*
0	0	0
0	1	0
1	0	0
1	1	1

OR

in_1	in_2	*out*
0	0	0
0	1	1
1	0	1
1	1	1





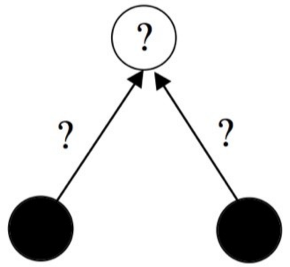


The need to find weights analytically

- Constructing simple networks by hand is one thing. But what about harder problems?
- For example:

XOR

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

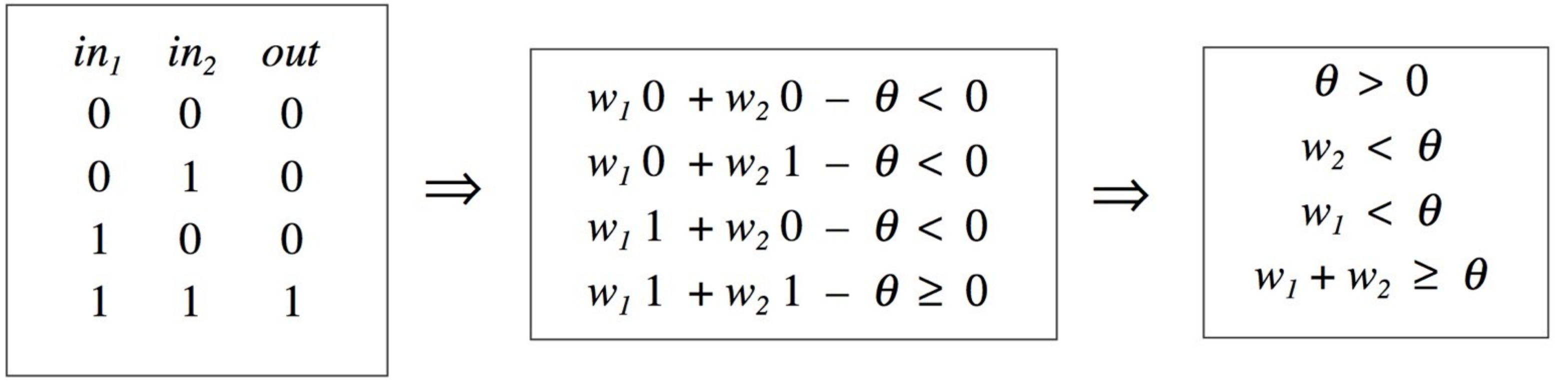


- How long do we keep looking for a solution?
- We need to be able to calculate appropriate parameters rather than looking for solutions by trial and error.
- Each training pattern produces a linear inequality for the output in terms of the inputs and the network parameters. These can be used to compute the weights and thresholds.

Finding weights analytically for the AND network

- We have two weights, w_1 and w_2, and the threshold θ, and for each training pattern we need to satisfy:

- So the training data lead to four inequalities:



in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\Rightarrow$$

$$\begin{aligned} w_1 0 + w_2 0 - \theta &< 0 \\ w_1 0 + w_2 1 - \theta &< 0 \\ w_1 1 + w_2 0 - \theta &< 0 \\ w_1 1 + w_2 1 - \theta &\geq 0 \end{aligned}$$

$$\Rightarrow$$

$$\begin{aligned} \theta &> 0 \\ w_2 &< \theta \\ w_1 &< \theta \\ w_1 + w_2 &\geq \theta \end{aligned}$$

- It is easy to see that there are an infinite number of solutions. Similarly, there are an infinite number of solutions for the NOT and OR networks.

Limitations of simple perceptrons

- We can follow the same procedure for the XOR network:

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

$$\Rightarrow \quad \begin{array}{l} w_1 0 + w_2 0 - \theta < 0 \\ w_1 0 + w_2 1 - \theta \geq 0 \\ w_1 1 + w_2 0 - \theta \geq 0 \\ w_1 1 + w_2 1 - \theta < 0 \end{array} \quad \Rightarrow \quad \begin{array}{c} \theta > 0 \\ w_2 \geq \theta \\ w_1 \geq \theta \\ w_1 + w_2 < \theta \end{array}$$

- Clearly the second and third inequalities are incompatible with the fourth, so there is in fact no solution. We need more complex networks, e.g. that combine together many simple networks, or use different activation/thresholding/transfer functions.
- It then becomes much more difficult to determine all the weights and thresholds by hand.

Limitations of simple perceptrons

- We can follow the same procedure for the XOR network:

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

$$\Rightarrow \quad \begin{array}{l} w_1 0 + w_2 0 - \theta < 0 \\ w_1 0 + w_2 1 - \theta \ge 0 \\ w_1 1 + w_2 0 - \theta \ge 0 \\ w_1 1 + w_2 1 - \theta < 0 \end{array} \quad \Rightarrow \quad \begin{array}{c} \theta > 0 \\ w_2 \ge \theta \\ w_1 \ge \theta \\ w_1 + w_2 < \theta \end{array}$$

- Clearly the second and third inequalities are incompatible with the fourth, so there is in fact no solution. We need more complex networks, e.g. that combine together many simple networks, or use different activation/thresholding/transfer functions.
- It then becomes much more difficult to determine all the weights and thresholds by hand.

Training of a perceptron

- The algorithm for training a perceptron
    1. Initialization
    2. Activation
    3. Weight learning
    4. Iteration

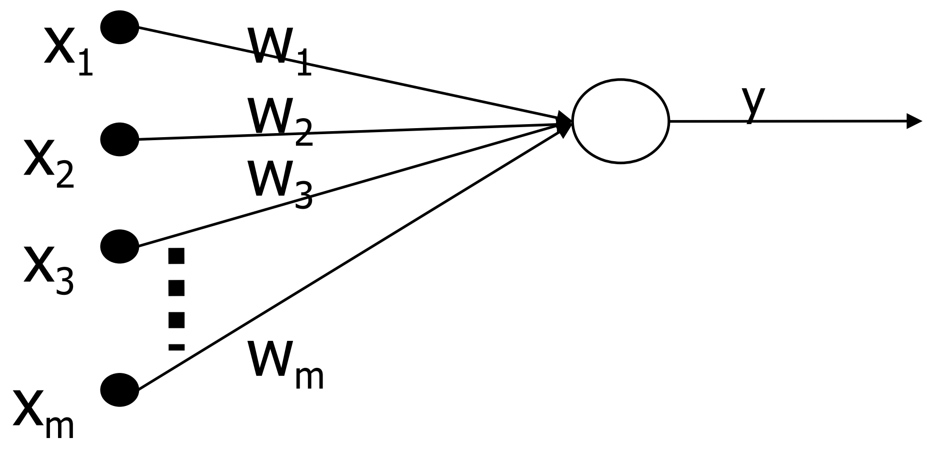
Perceptron training

- 1) Initialization

Set initial weights $w_1, w_2, \ldots, w_m$

Set the threshold θ to a random number in the range [-0.5, 0.5]

Set the learning rate η to a positive value less than 1



Perceptron training

- 2) Activation

Calculate the actual output at the iteration p=1

output: $$y = \Phi\left(\sum_{i=1}^{m} x_i \cdot w_i\right)$$

Activation function:

Threshold/step function (or sign function)

output: $$y(p) = step\left[\left(\sum_{i=1}^{m} x_i \cdot w_i\right) - \theta\right]$$

Perceptron training

- 3) Weight training

  Update the weights of the perceptron

new weight $w_i(p+1) = w_i(p) + \Delta w_i(p)$

weight correction $\Delta w_i(p) = \eta . x_i(p) . e(p)$

error $e(p) = d(p) - y(p)$

Perceptron training

- 4) Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

p=2

Activation

$$y(p) = step\left[\left(\sum_{i=1}^{m} x_i \cdot w_i\right) - \theta\right]$$

Weight training)

p=3

Activation

Weight training

……

An example

- Train a perceptron to perform logical operation AND

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1	0	0	0	0.3	-0.1
	2	0	1	0.3	-0.1	0	0	0	0.3	-0.1
	3	1	0	0.3	-0.1	0	1	-1		
	4	1	1							

An example

- 1)Initialization

1 epoch has 4 possible input patterns: 00, 01, 10, 11

Initial weights are given $w_1 = 0.3$, $w_2 = -0.1$; $\theta=0.2$, $\eta=0.1$

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1					
	2	0	1							
	3	1	0							
	4	1	1							

An example

- 2)Activation

Actual output

$y = step[\ (0(0.3)+0(-0.1)) - 0.2] = step[-0.2] = 0$

$$y(p) = step\left[\left(\sum_{i=1}^{m} x_i \cdot w_i\right) - \theta\right]$$

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1		**0**			
	2	0	1							
	3	1	0							
	4	1	1							

Example

- 3) Weight training

AND operation for inputs 00 is 0

Error e(p) = d(p) – y(p) = 0 – 0 = 0

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1	**0**	0	**0**		
	2	0	1							
	3	1	0							
	4	1	1							

Example

- 3) Weight training

For w_1: $\Delta w_1(p) = \eta . x_1(p) . e(p) = 0.1(0)(0) = 0$

$w_1(p+1) = w_1(p) + \Delta w_1(p) = 0.3 + 0 = 0.3$

For w_2 in similar way, $w_2(p+1) = -0.1$

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1	0	0	0	**0.3**	**-0.1**
	2	0	1							
	3	1	0							
	4	1	1							

Example

- 4) Iteration

p+1 = 2

Repeat steps 2, 3, 4……until convergence

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1	0	0	0	0.3	-0.1
	2	0	1	0.3	-0.1					
	3	1	0							
	4	1	1							

Example

- Answer for Epoch 1

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
1	1	0	0	0.3	-0.1	0	0	0	0.3	-0.1
	2	0	1	0.3	-0.1	0	0	0	0.3	-0.1
	3	1	0	0.3	-0.1	0	1	-1	0.2	-0.1
	4	1	1	0.2	-0.1	1	0	1	0.3	0.0

Example

- Answer for Epoch 2

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	$d(p)$	$y(p)$	$e(p)$	$w_1(p+1)$	$w_2(p+1)$
2	5	0	0	0.3	0.0	0	0	0	0.3	0.0
	6	0	1	0.3	0.0	0	0	0	0.3	0.0
	7	1	0	0.3	0.0	0	1	-1	0.2	0.0
	8	1	1	0.2	0.0	1	1	0	0.2	0.0

Example

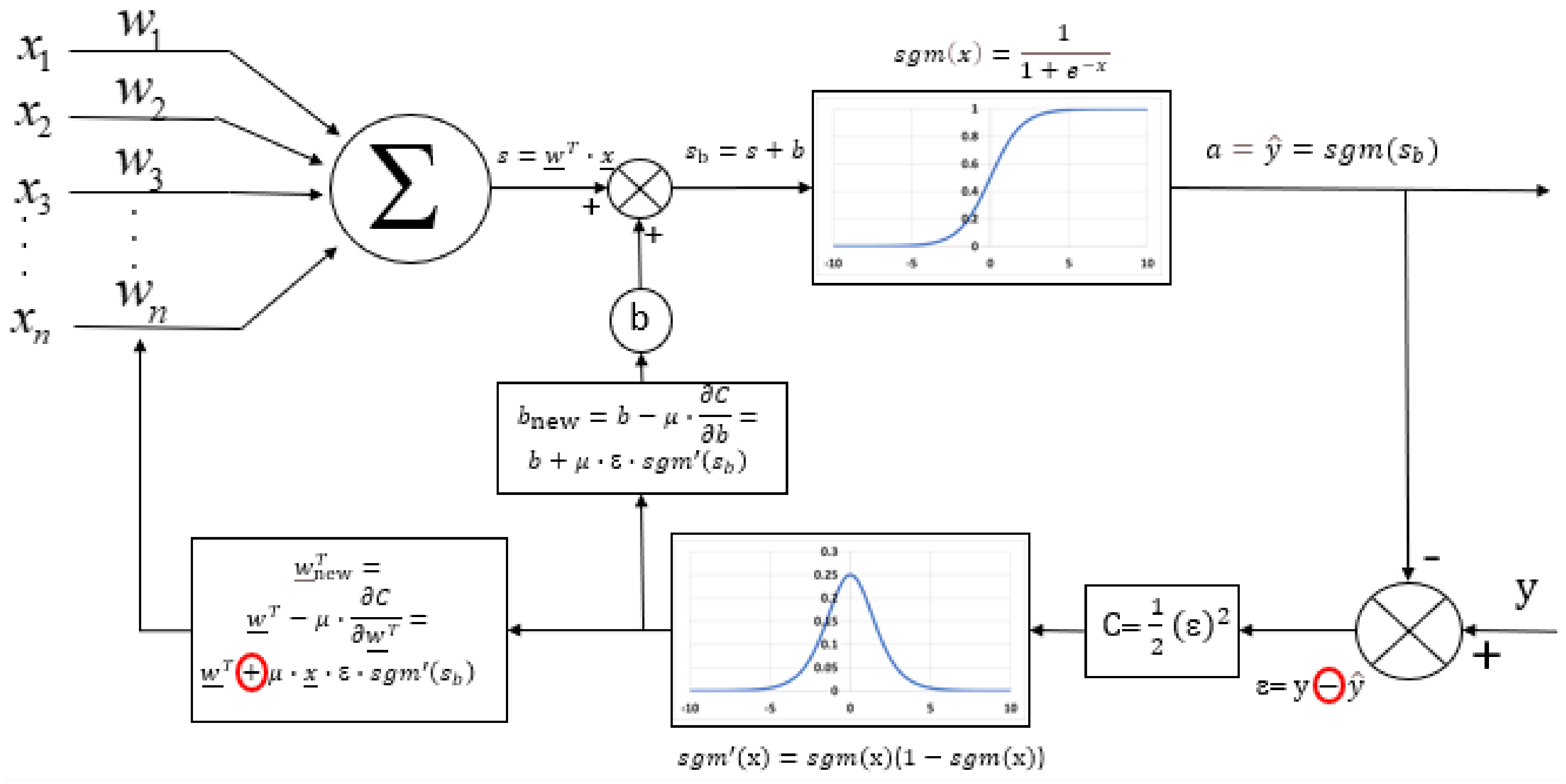
- Answer for Epoch 5

Epoch	Iteration	Inputs		Initial weights		Desired output	Actual output	Error	Final weights	
	p	$x_1(p)$	$x_2(p)$	$w_1(p)$	$w_2(p)$	d(p)	y(p)	e(p)	$w_1(p+1)$	$w_2(p+1)$
5	17	0	0	0.1	0.1	0	0	0	0.1	0.1
	18	0	1	0.1	0.1	0	0	0	0.1	0.1
	19	1	0	0.1	0.1	0	0	0	0.1	0.1
	20	1	1	0.1	0.1	1	1	0	0.1	0.1

Artificial neural networks

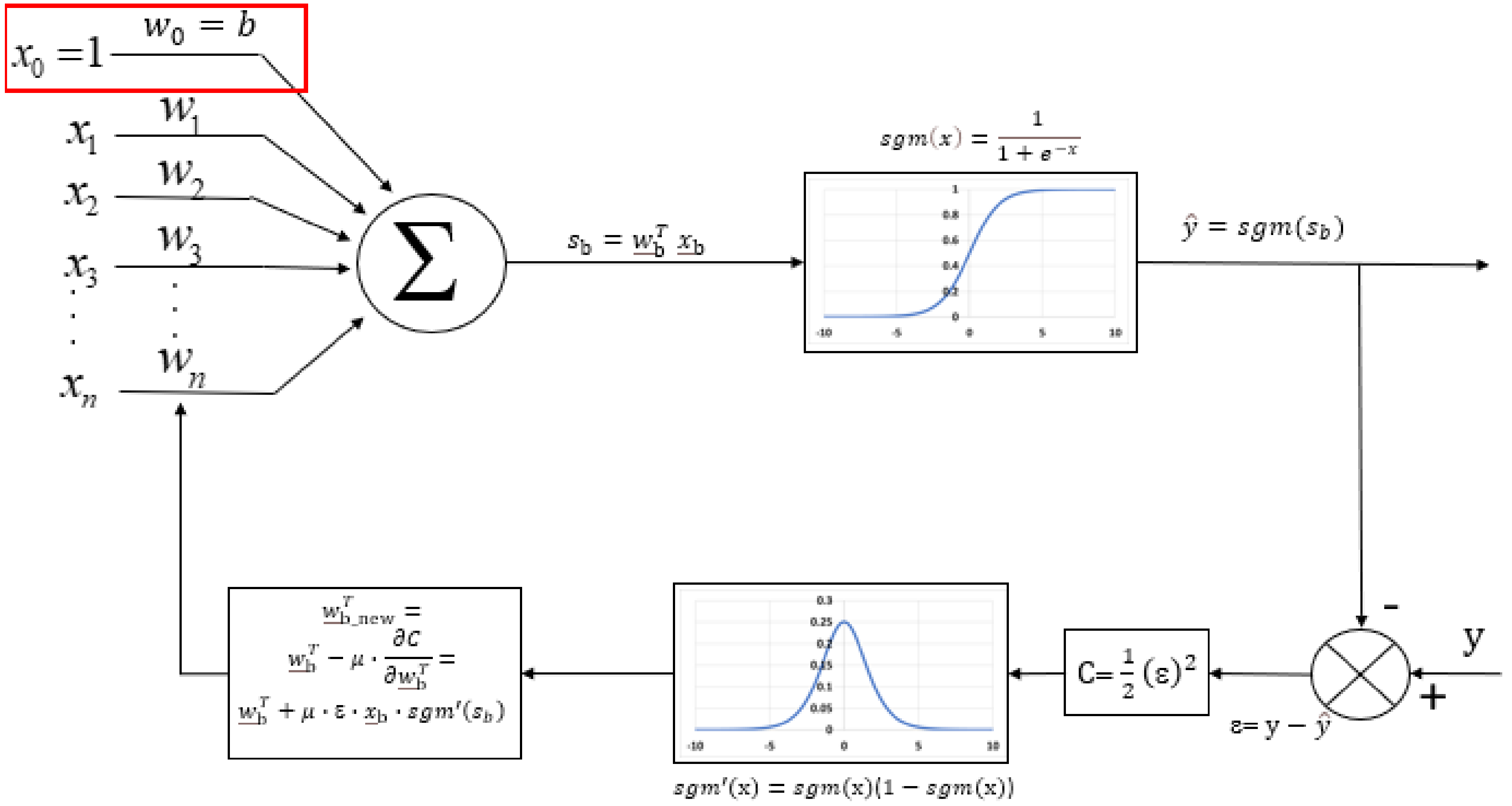
- Artificial neural network is a data processing system consisting of a large number of simple, highly interconnected processing elements (artificial neurons) in an architecture inspired by the structure of the cerebral cortex of the brain to yield a network with rather complex behavior.
- In ANN, neurons (processing elements) are organized into a sequence of layers with full or partial connection between the layers.
- For simplicity, usually the neurons in the same layer have the same activation function.

Neuron



x_1 w_1
x_2 w_2
x_3 w_3
x_n w_n
Σ
$s = \underline{w}^T \cdot \underline{x}$
+
+
$s_b = s + b$
b
$sgm(x) = \frac{1}{1+e^{-x}}$
$a = \hat{y} = sgm(s_b)$
$b_{new} = b - \mu \cdot \frac{\partial C}{\partial b} = b + \mu \cdot \varepsilon \cdot sgm'(s_b)$
$\underline{w}^T_{new} = \underline{w}^T - \mu \cdot \frac{\partial C}{\partial \underline{w}^T} = \underline{w}^T + \mu \cdot \underline{x} \cdot \varepsilon \cdot sgm'(s_b)$
$C = \frac{1}{2}(\varepsilon)^2$
-
y
+
$\varepsilon = y - \hat{y}$
$sgm'(x) = sgm(x)(1 - sgm(x))$

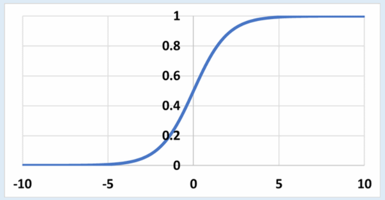
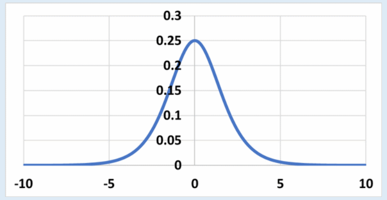
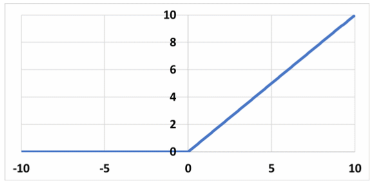
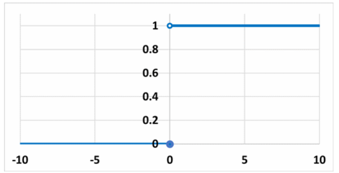
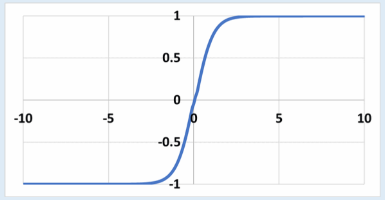
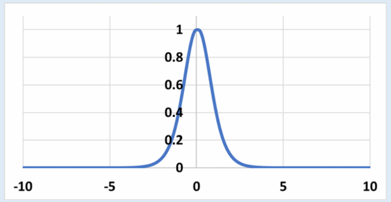
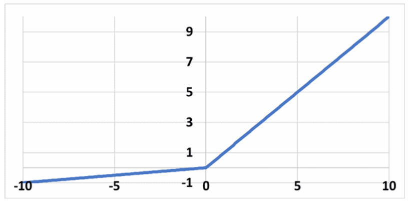
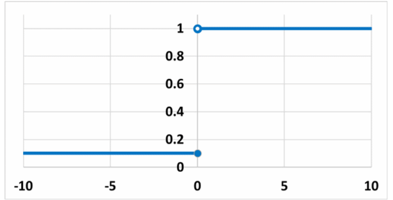
Neuron



$x_0 = 1$
$w_0 = b$
x_1
w_1
x_2
w_2
x_3
w_3
x_n
w_n
Σ
$s_b = w_b^T x_b$
$sgm(x) = \frac{1}{1+e^{-x}}$
$\hat{y} = sgm(s_b)$
-
+
y
$\varepsilon = y - \hat{y}$
$C = \frac{1}{2}(\varepsilon)^2$
$sgm'(x) = sgm(x)(1 - sgm(x))$
$w_{b\_new}^T = w_b^T - \mu \cdot \frac{\partial C}{\partial w_b^T} = w_b^T + \mu \cdot \varepsilon \cdot x_b \cdot sgm'(s_b)$

In case of number of layers > 2 during the backpropagation

Activation function

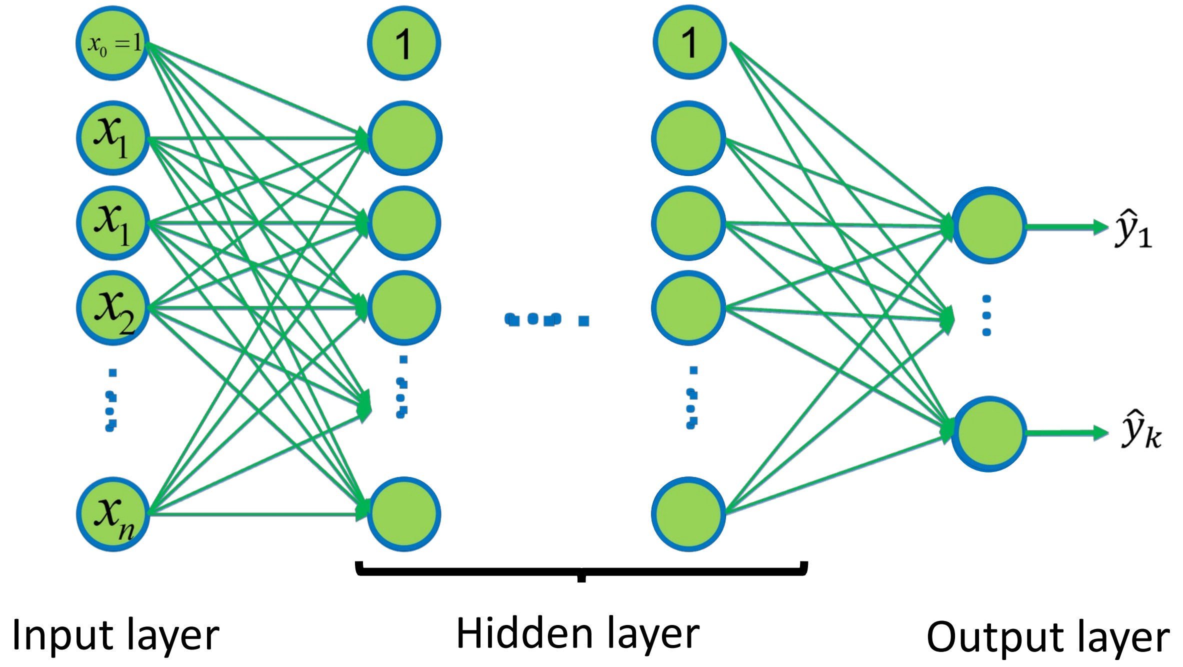
Equation	Graph	Derivative	Graph
Sigmoid $s\,gm(x)=\frac{1}{1+e^{-x}}$		()	
Rectifier Linear Unit (ReLU) $ReLU(x)=\begin{cases}0, if\ x<0\\ x,\ if x\geq 0\end{cases}$		$ReLU'(x)=\begin{cases}0, if\ x<0\\ 1, if\ x\geq 0\end{cases}$	
Tanh $t\,anh(x)=\frac{e^{2x}-1}{e^{2x}+1}$		$t\,anh'(x)=1-\tanh(x)^2$	
Leaky ReLU $LReLU(x)=\begin{cases}0.1x, if\ x<0\\ x,\ \ if x\geq 0\end{cases}$		$LReLU'(x)=$¿ $\begin{cases}0.1, if\ x<0\\ 1\end{cases}$	

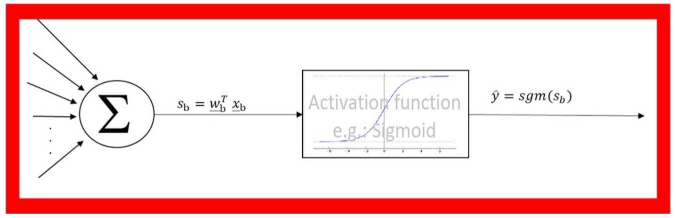
Artificial neural networks

- In terms of density, 2 important groups can be distinguished in a first approximation
  - **Fully Connected Artificial Neural Networks**
    All neurons in a layer are connected to all neurons in the next layer.
    - **Advantage:**
      - simple architecture
      - maximizes the potential of the network size
    - **Disadvantage: (for large networks)**
      - requires a lot of memory
      - requires a lot of computation
  - **Partially coupled artificial neural networks**
    All neurons in a layer are connected to only a few neurons in the next layer.

Fully connected ANN

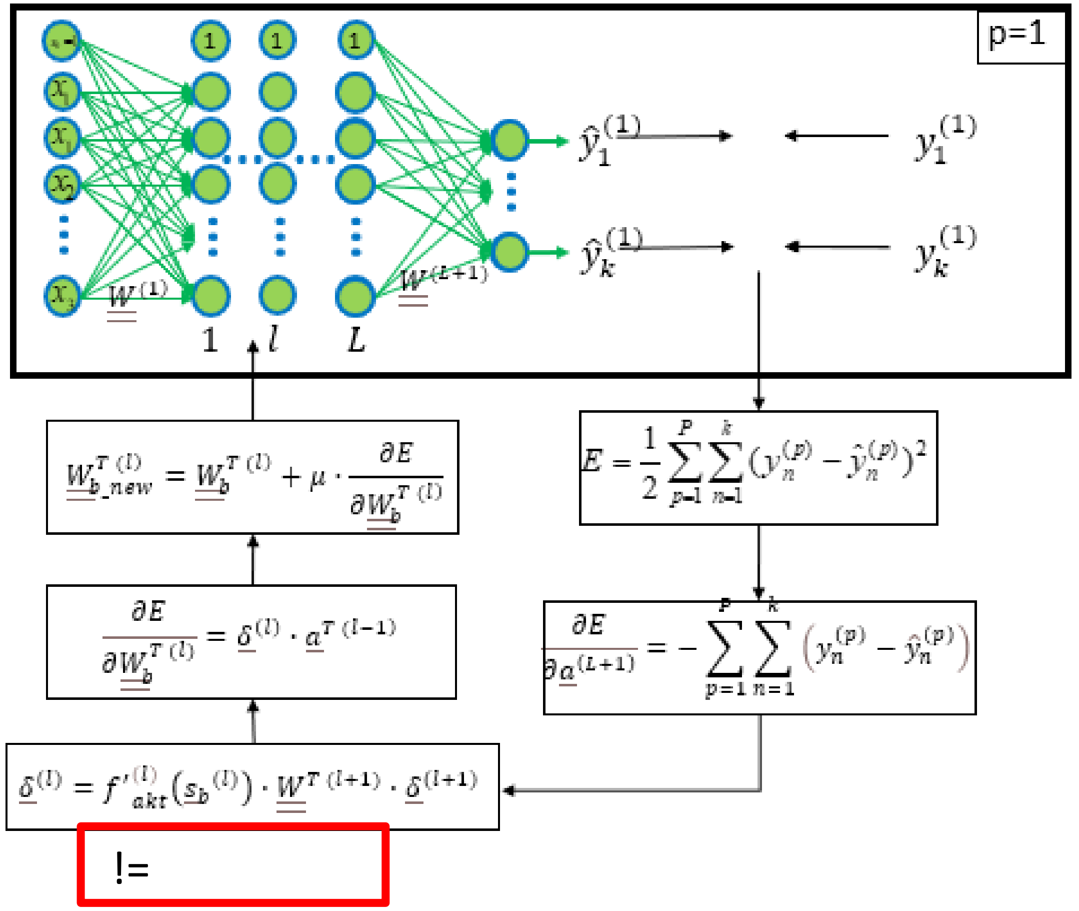
- The input to neural networks is determined by the available data and the preparation of the data.
- Its output is determined by the target task.
- The depth of its hidden layer is determined by its decision complexity.
- Its width is determined by the amount of data needed for decisions.
- In general, the appropriate dimensions of the network cannot be determined in advance.





Training of fully connected ANN

- In a learning cycle, we consider a number "P" of inputs (x) and corresponding outputs ().
- The pair of output and expected output (y) is compared and an error function (E) is defined based on it.
- The error is propagated backwards from layer to layer (L->1) taking advantage of the fact that the auxiliary parameter can be computed iteratively.



p=1
x_1
x_2
x_j
1
$\underline{\underline{W}}^{(1)}$
$\underline{\underline{W}}^{(L+1)}$
1
l
L
$\hat{y}_1^{(1)}$
$\hat{y}_k^{(1)}$
$y_1^{(1)}$
$y_k^{(1)}$
$\underline{\underline{W}}_{b\_new}^{T\,(l)} = \underline{\underline{W}}_b^{T\,(l)} + \mu \cdot \frac{\partial E}{\partial \underline{\underline{W}}_b^{T\,(l)}}$
$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{n=1}^{k}(y_n^{(p)} - \hat{y}_n^{(p)})^2$
$\frac{\partial E}{\partial \underline{\underline{W}}_b^{T\,(l)}} = \underline{\delta}^{(l)} \cdot \underline{a}^{T\,(l-1)}$
$\frac{\partial E}{\partial \underline{a}^{(L+1)}} = -\sum_{p=1}^{P}\sum_{n=1}^{k}\left(y_n^{(p)} - \hat{y}_n^{(p)}\right)$
$\underline{\delta}^{(l)} = f'^{(l)}_{akt}\left(\underline{s_b}^{(l)}\right) \cdot \underline{\underline{W}}^{T\,(l+1)} \cdot \underline{\delta}^{(l+1)}$
!=

Training of fully connected ANN

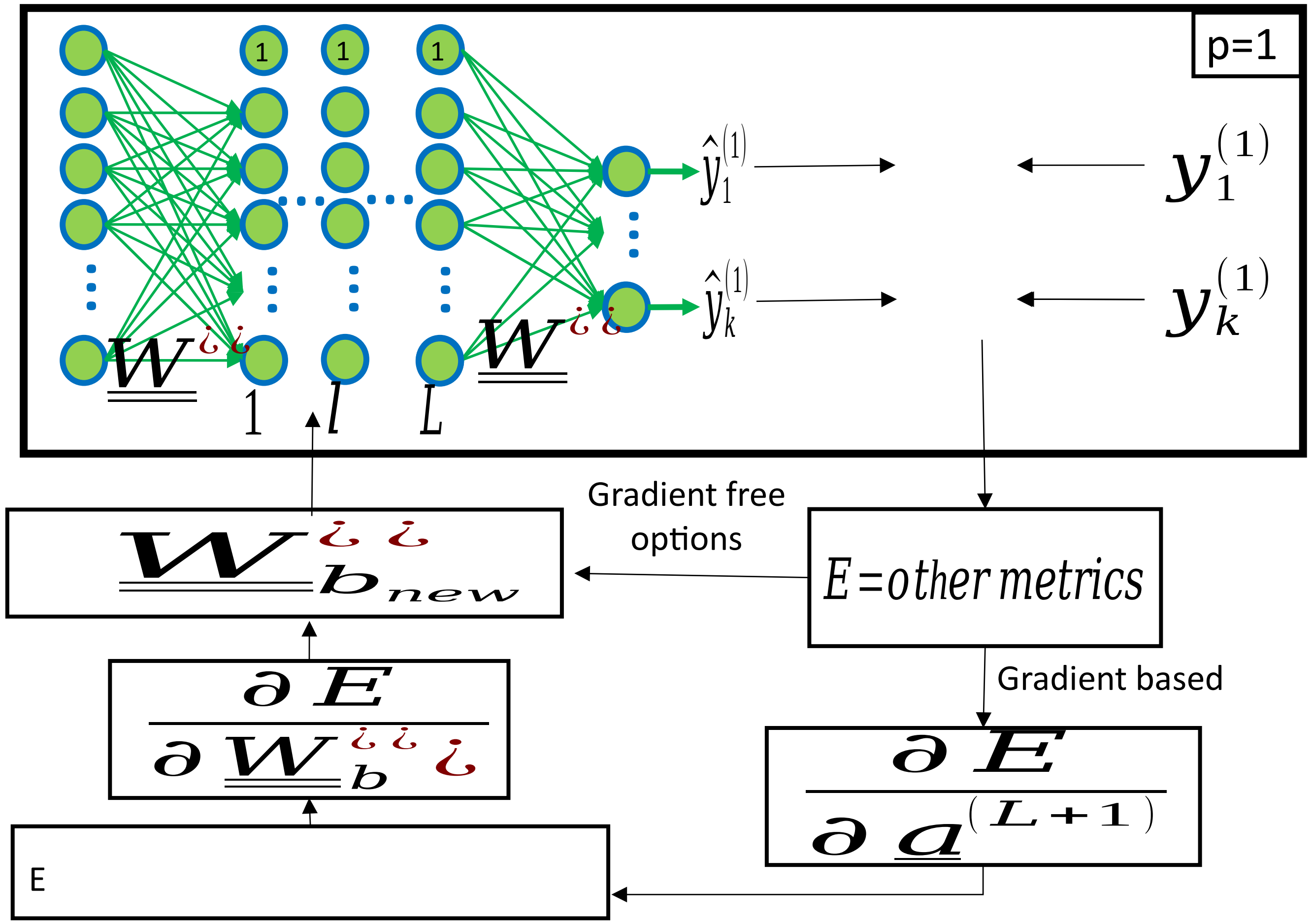
!=

Fully connected ANN

- The followings can be chosen when teaching a neural network:
  - Other error functions (E)
  - Other optimization methods

The optimization methods can be:

- Gradient-based
  - Other methods can be based on gradient descent, Gauss-Newton, Quasi-Newton, Levenberg-Marquardt, ADAM, ...
- Gradient-free
  - Random search, Evolutionary algorithms (later lecture)...
- Miscellaneous
  - Memetic algorithms (later lecture)...



Artificial Neural Networks

Partially coupled artificial neural networks

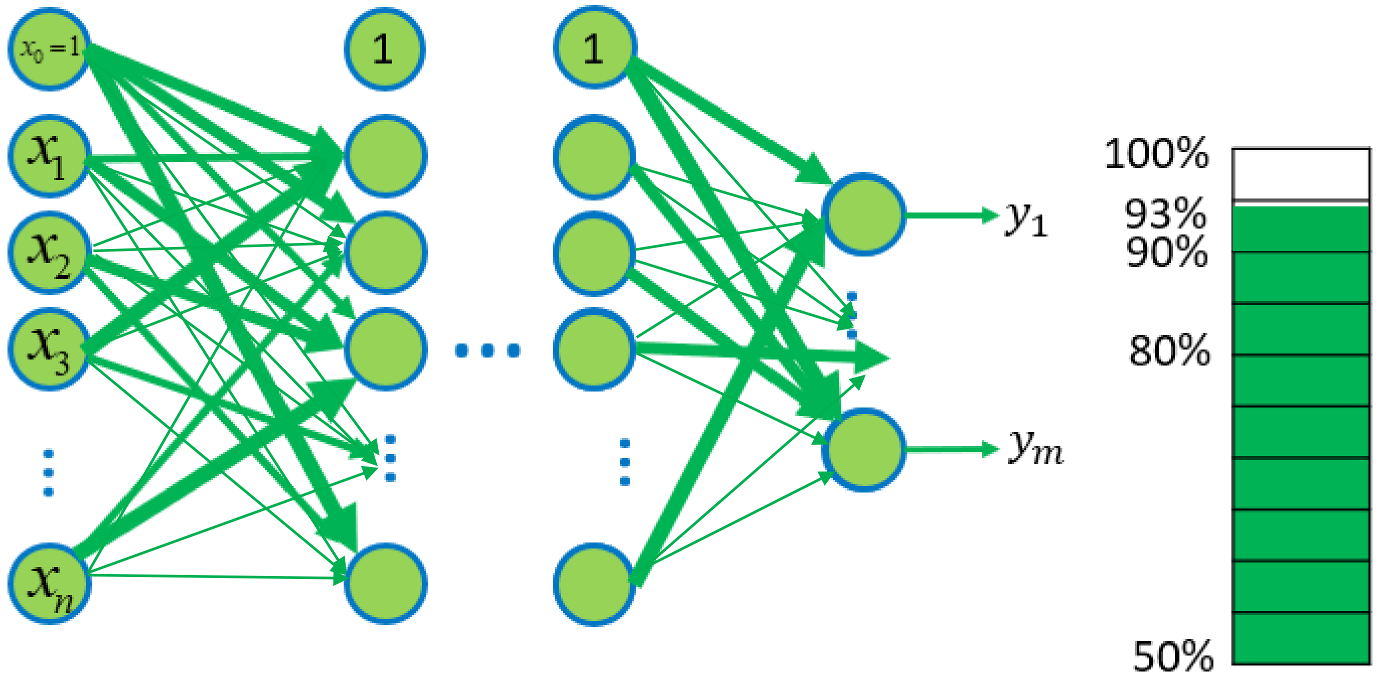
All neurons in a layer are connected to only a few neurons in the next layer.

- Advantage:
  - lower memory requirements
  - less computational effort
- Disadvantage:
  - it is usually not known in advance which weights should be omitted

- Can be generated e.g.:
  - From fully coupled artificial neural networks by dropping less significant weights
  - By engineering considerations (e.g.: stronger association of nearby words/pixels)
  - Neural network growth (Neuroevolution)

Omission of less significant weights

- After training, only a minimal reduction in the accuracy of the neural network is achieved by dropping the weights of minor importance, in exchange for a significant reduction in the number of parameters.
- In some cases (e.g., extreme learning machine, more typical for wide structures), a partially coupled ANN can achieve competitive learning speed by adding and subtracting weights from the initial state even using CPU.
- In between dropping weights, it may be worthwhile to retrain the model, in which case other paths may be strengthened to take the role of the dropped links, in which case the final accuracy is less degraded.



Neuroevolution

The essence of the method is to optimize the structure of the network in addition to the parameters.

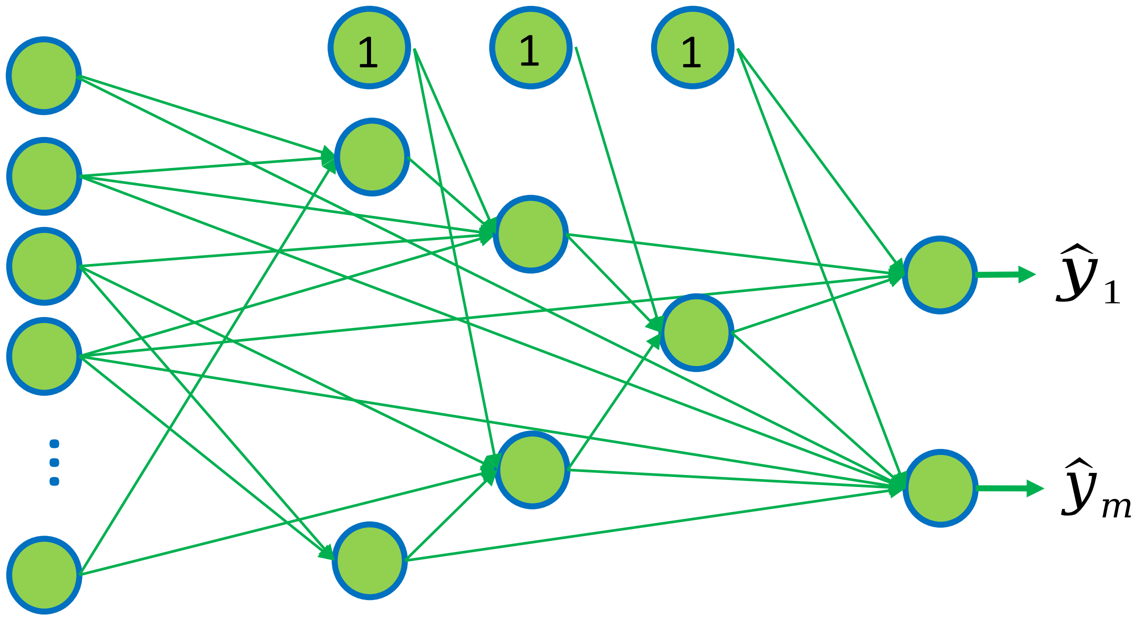
This is true automatic machine learning for neural networks.

Advantage:

- Optimal structure generation
- Automatic operation

Disadvantage:

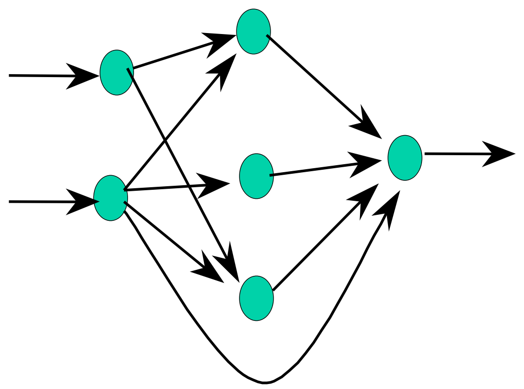
- Resource intensive



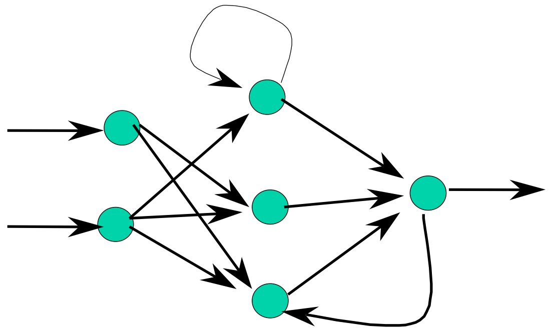
Simplest case is when individual neurons and the connections between them are modified by an evolutionary algorithm.

Under development (explorable): optimization through multi-level mapping that can use its own substructures. (further theory: genetic programming) Here too, it is worth working with structured input data.

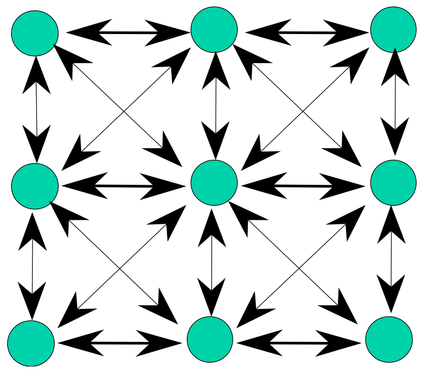
Grouping by signal flow



Feedforward



Recurrent



Cellular

Feedforward neural networks

Use of:

- fast processing of time-independent data,
- non real-time accurate processing of time-dependent data

Recurrent neural networks

Use of:

- fast processing of time-dependent data

A **cellular neural network** is a network where there is a connection only between neighboring units.

Usage:

- mainly image processing

Grouping of certain layers

Different layer types have been developed for each (sub)task

- 1 layer usually contains the same type of neurons

Evaluation

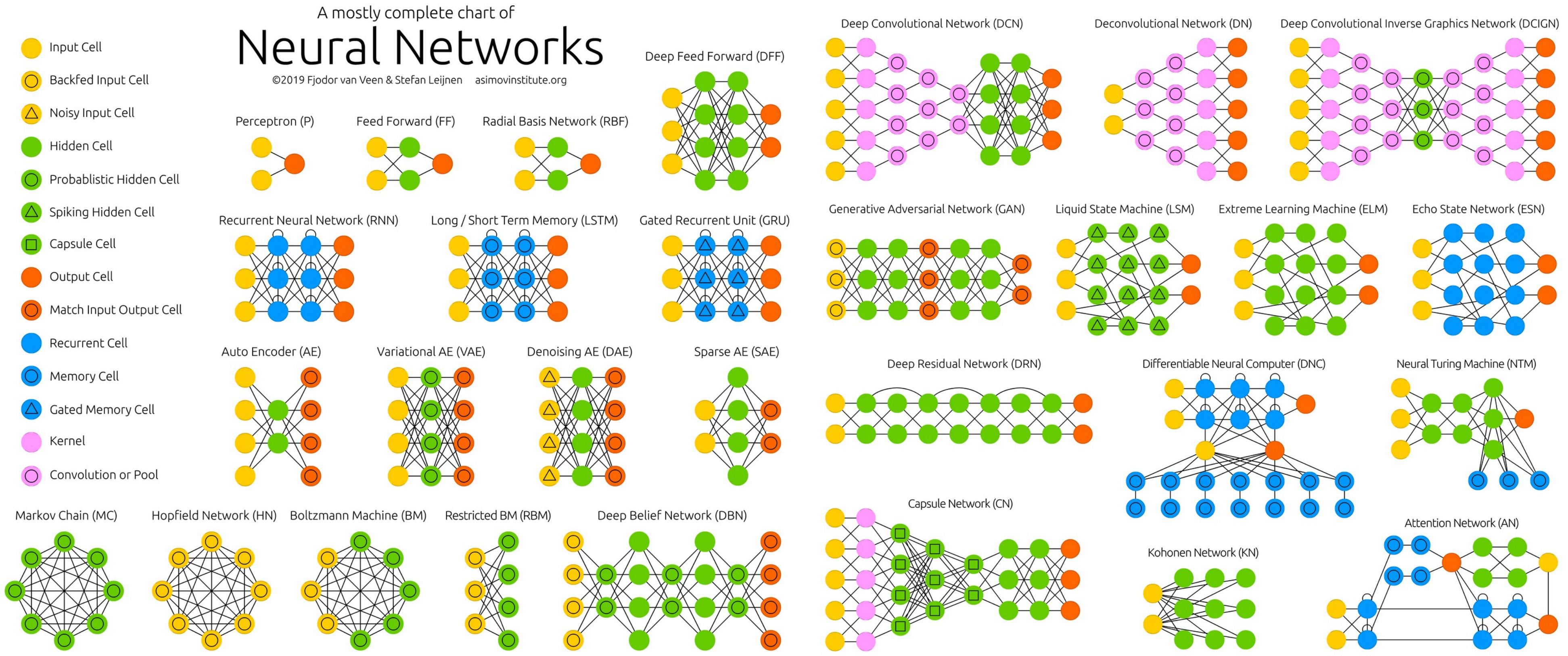
- Input
- Output
- Fully coupled layer
- Partially coupled layer
- Activation layer
- Link layers (to link more than 2 layers)
- Convolutional
- Pooling layer
- Reconnected layer
- Memory cell layers
- Highlighting type (ROI, Attention)

Training specific

- Normalization layers
  - Batch normalization layer (on sample batch)
  - Batch normalization layer (on individual samples)
- Regularization layers
  - Noise generation layers (random noise addition)
  - Dropout layer: Punctuated (random neurons) or Contiguous (random but contiguous neurons)

An overview of NNs

Different neural networks have been developed for different tasks



A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org
Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Differentiable Neural Computer (DNC)
Neural Turing Machine (NTM)
Capsule Network (CN)
Kohonen Network (KN)
Attention Network (AN)

Source