Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

# Machine Learning

A.Y. 2020/2021

Prof. L. Iocchi, F. Patrizi





# 11. Artificial Neural Networks

L. Iocchi, F. Patrizi

with contributions from Valsamis Ntouskos

# Overview

- Feedforward networks
- Architecture design
- Cost functions
- Activation functions
- Gradient computation (back-propagation)
- Learning (stochastic gradient descent)
- Regularization

*References*

Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning - Chapters 6, 7, 8. `http://www.deeplearningbook.org`





# Artificial Neural Networks (ANN)

Alternative names:

- Neural Networks - (NN)
- Feedforward Neural Networks - (FNN)
- Multilayer Perceptrons - (MLP)

Function approximator using a parametric model.

Suitable for tasks described as associating a vector to another vector.

# Artificial Neural Networks (ANN)

**Goal**:

Estimate some function $f : X \to Y$, with $Y = \{C_1, \ldots, C_k\}$ or $Y = \Re$

**Data**:

$D = \{(\mathbf{x}_n, t_n)_{n=1}^N\}$ such that $t_n \approx f(\mathbf{x}_n)$

**Framework**:

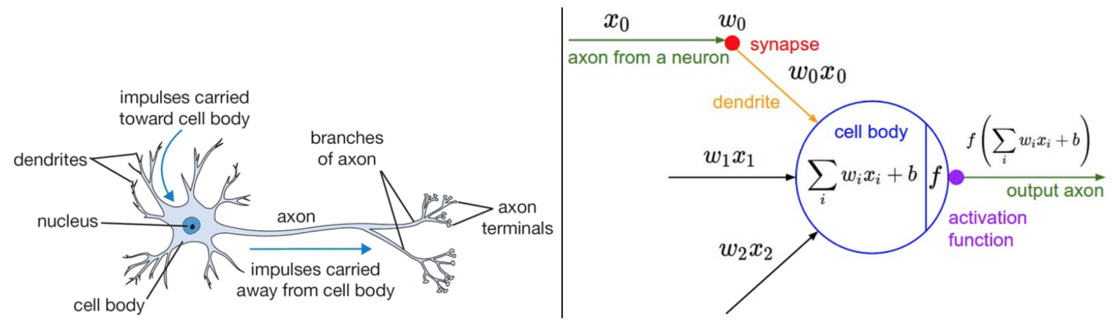
Define $y = \hat{f}(\mathbf{x}; \boldsymbol{\theta})$ and learn parameters $\boldsymbol{\theta}$ so that $\hat{f}$ approximates $f$.





# Feedforward Networks

Draw inspiration from brain structures




A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Image from Isaac Changhau `https://isaacchanghau.github.io`

Hidden layer output can be seen as an array of **unit** (neuron) activations based on the connections with the previous units

Note: Only use some insights, they are not a model of the brain!

# Feedforward Networks - Terminology

Feedforward information flows from input to output without any loop

Networks $f$ is a composition of elementary functions in an acyclic graph

Example:

$$f(\mathbf{x};\boldsymbol{\theta}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x};\boldsymbol{\theta}^{(1)});\boldsymbol{\theta}^{(2)});\boldsymbol{\theta}^{(3)})$$

where:

$f^{(m)}$ the $m$-th layer of the network

and

$\boldsymbol{\theta}^{(m)}$ the corresponding parameters





# Feedforward Networks - Terminology

FNNs are **chain structures**

The length of the chain is the **depth** of the network

Final layer also called **output layer**

**Deep learning** follows from the use of networks with a large number of layers (large depth)

# Feedforward Networks

Why FNNs?

Linear models cannot model interaction between input variables

Kernel methods require the choice of suitable kernels

- use generic kernels e.g. RBF, polynomial, etc. (convex problem)
- use hand-crafted kernels - application specific (convex problem)

FNN leaning:
complex combination of many parametric functions (non-convex problem)





# XOR Example - Linear model

Learning the XOR function - 2D input and 1D output

Dataset: $D = \{((0,0)^T, 0), ((0,1)^T, 1), ((1,0)^T, 1), ((1,1)^T, 0)\}$

Using linear regression with Mean Squared Error (MSE)

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} (t_n - y(\mathbf{x}_n))^2$$

with $y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$

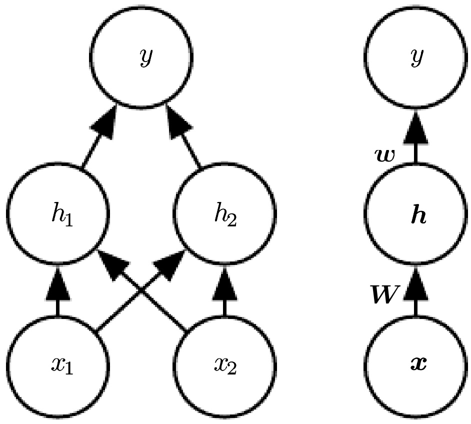
Optimal solution:

$\mathbf{w} = 0$ and $w_0 = \frac{1}{2}$, hence $y = 0.5$ everywhere!

Reason: No linear separator can explain the non-linear XOR function

# XOR Example - FNN

Specify a two layers network:








# XOR Example - FNN

Hidden units:

$$\mathbf{h} = g(\mathbf{W}^T\mathbf{x} + \mathbf{c})$$

with $g(\alpha) = \max(0, \alpha)$

Output:

$$y = \mathbf{w}^T\mathbf{h} + b$$

Full model:

$$y(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{w}^T \max(0, \mathbf{W}^T\mathbf{x} + \mathbf{c}) + b$$

with $\boldsymbol{\theta} = \langle \mathbf{W}, \mathbf{c}, \mathbf{w}, b \rangle$

Note: non-linear model in $\boldsymbol{\theta}$

# XOR Example - FNN

Model:

$$y(\mathbf{x}) = f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{w}^T \max(0, \mathbf{W}^T\mathbf{x} + \mathbf{c}) + b$$

Mean Squared Error (MSE) loss function:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(t_n - y(\mathbf{x}_n))^2$$

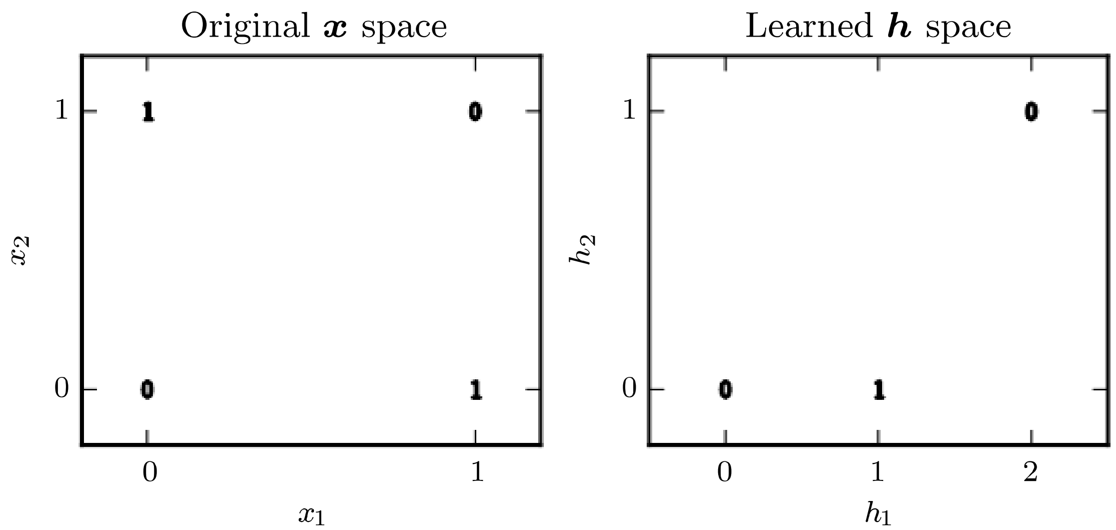
Solution:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$





# XOR Example - FNN



## Architecture design

Overall structure of the network

How many hidden layers? **Depth**

How many units in each layer? **Width**

Which kind of units? **Activation functions**

Which kind of cost function? **Loss function**





## Architecture design

How many hidden layers? **Depth**

**Universal approximation theorem**: a FFN with a linear output layer and at least one hidden layer with any "squashing" activation function (e.g., sigmoid) can approximate any Borel measurable function with any desired amount of error, provided that enough hidden units are used.

It works also for other activation functions (e.g., ReLU)

# Architecture design

How many units in each layer? **Width**

**Universal approximation theorem** does not say how many units.

In general it is exponential in the size of the input.

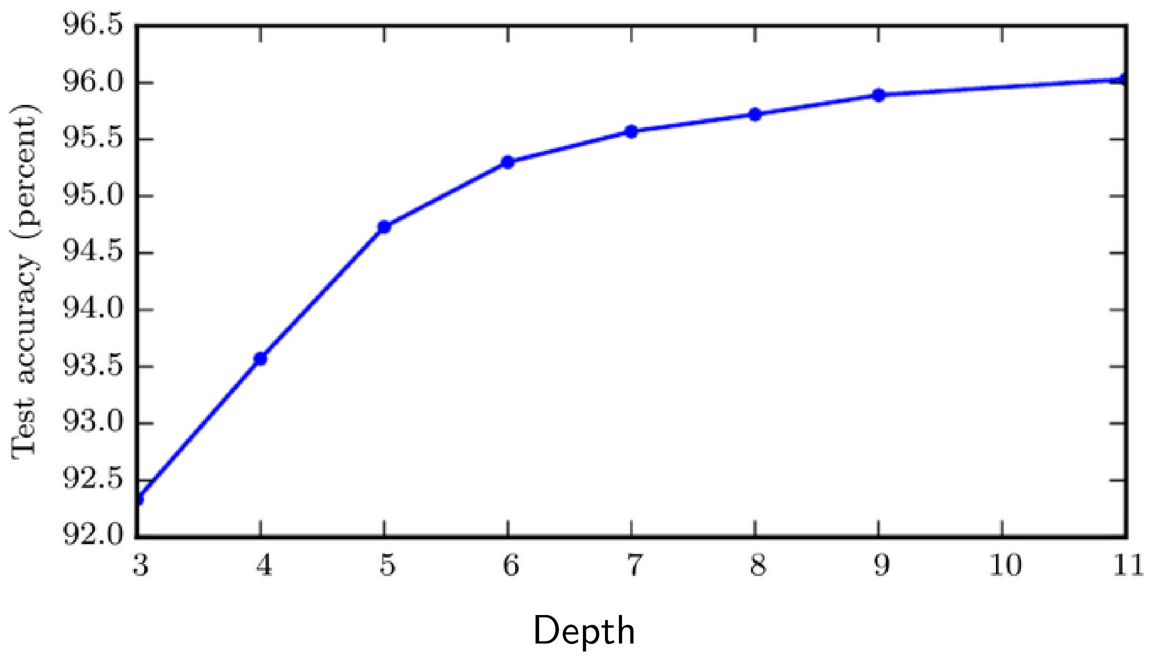
In theory, a short and very wide network can approximate any function.

In practice, a deep and narrow network is easier to train and provides better results in generalization.

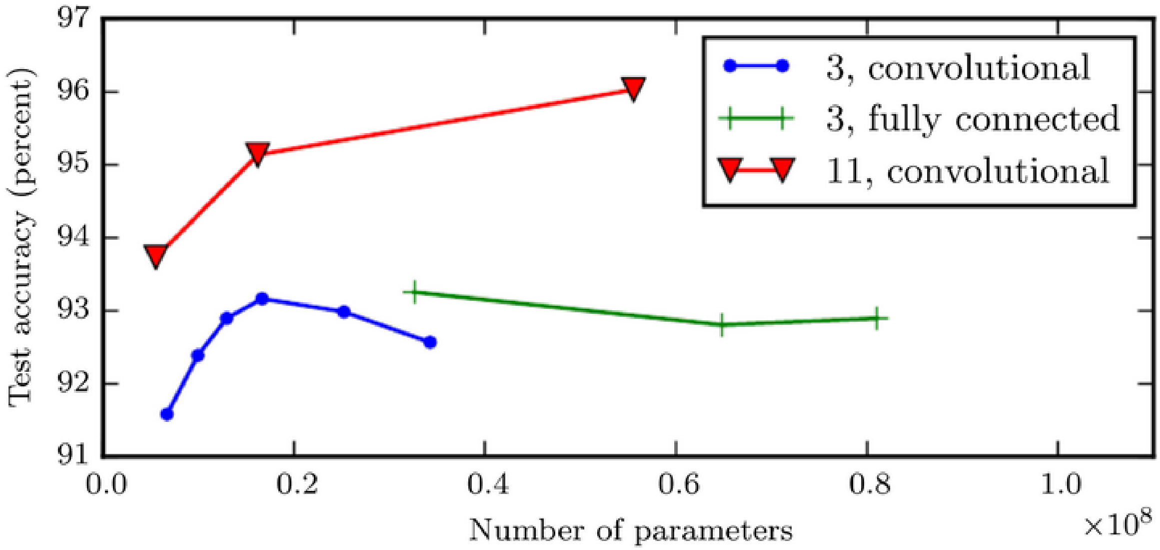


Sapienza University of Rome, Italy - Machine Learning (2020/2021)

Architecture design



# Architecture design








# Architecture design

Which kind of units? **Activation functions**

Which kind of cost function? **Loss function**

Gradient-based learning remarks

- Unit saturation can hinder learning
- When units saturate gradient becomes very small
- Suitable cost functions and unit nonlinearities help to avoid saturation