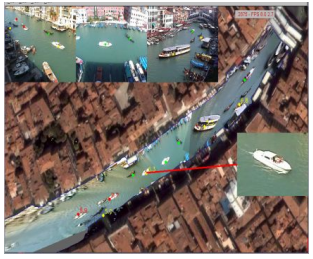
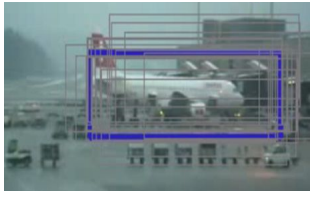
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

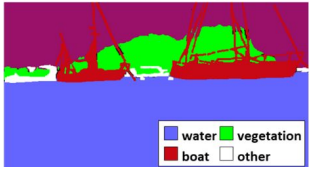


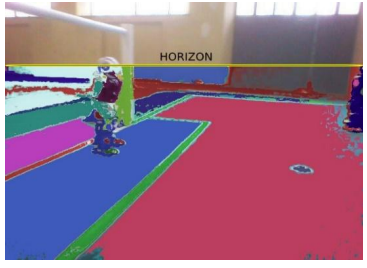
Corso di Sistemi Informativi
A.A. 2018/19

Docente
Domenico Daniele Bloisi

introduzione alle CNN

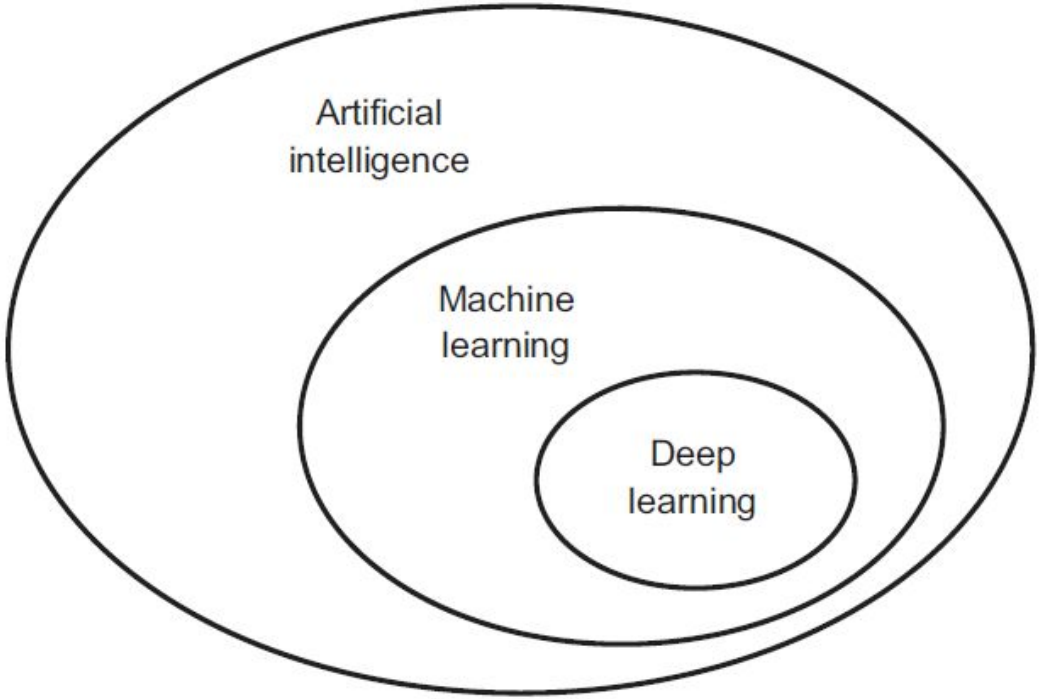





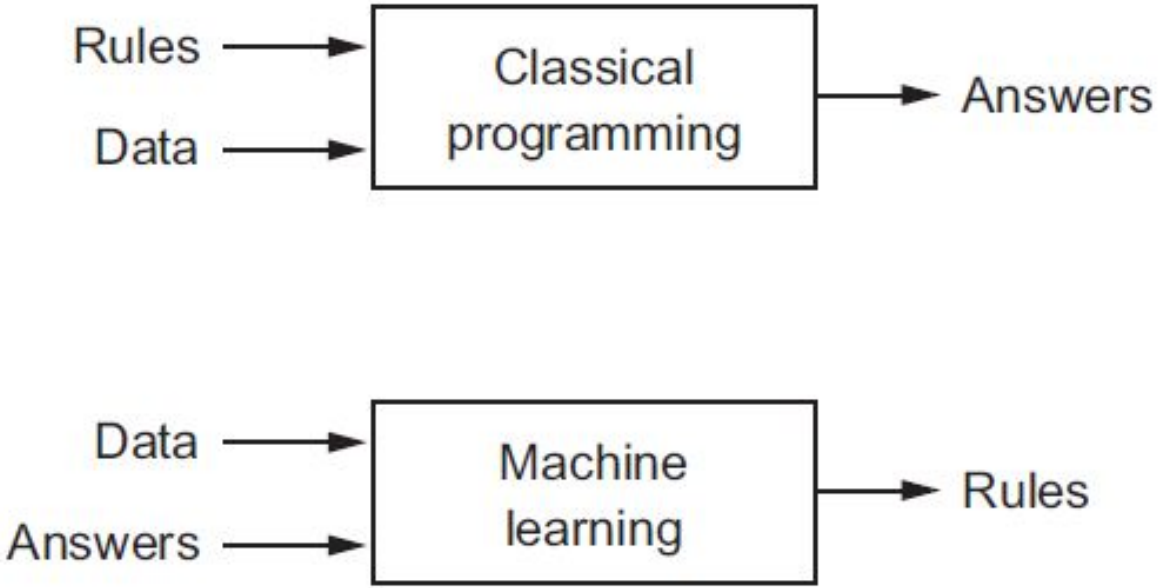

Giugno 2019

AI, ML, and DL



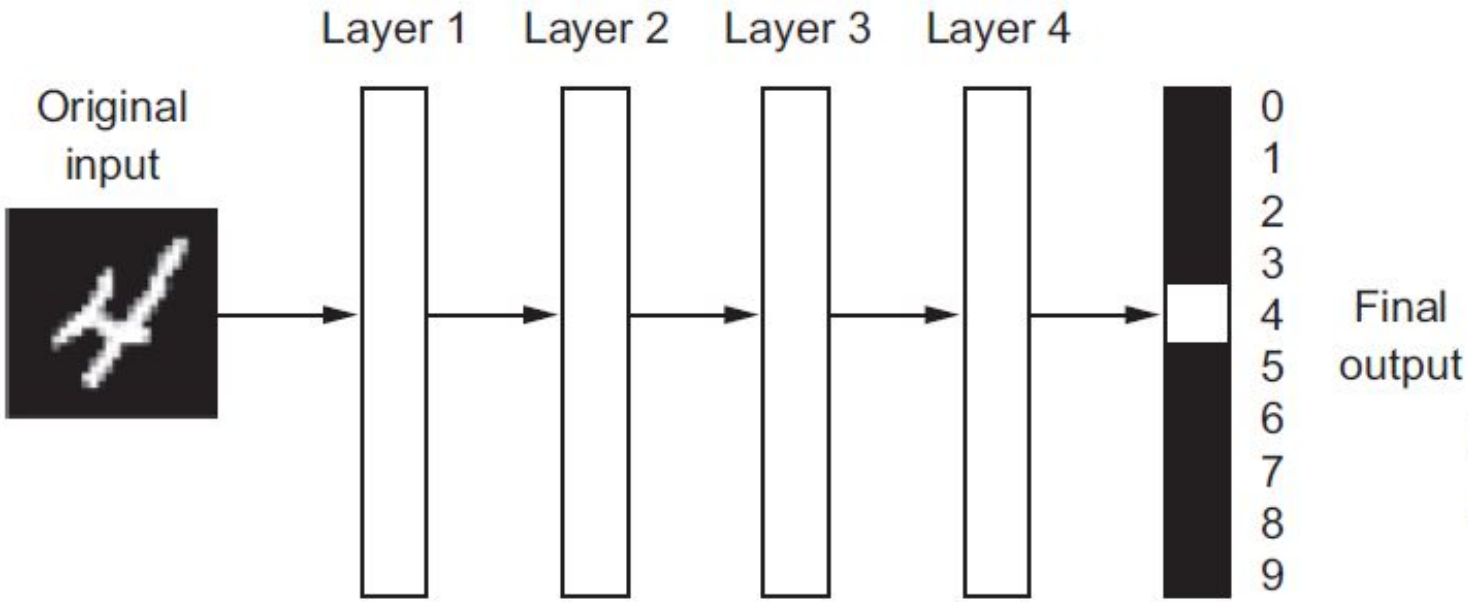
Francois Chollet "Deep Learning with Python" Manning Publications Co.

Machine Learning



Francois Chollet "Deep Learning with Python" Manning Publications Co.

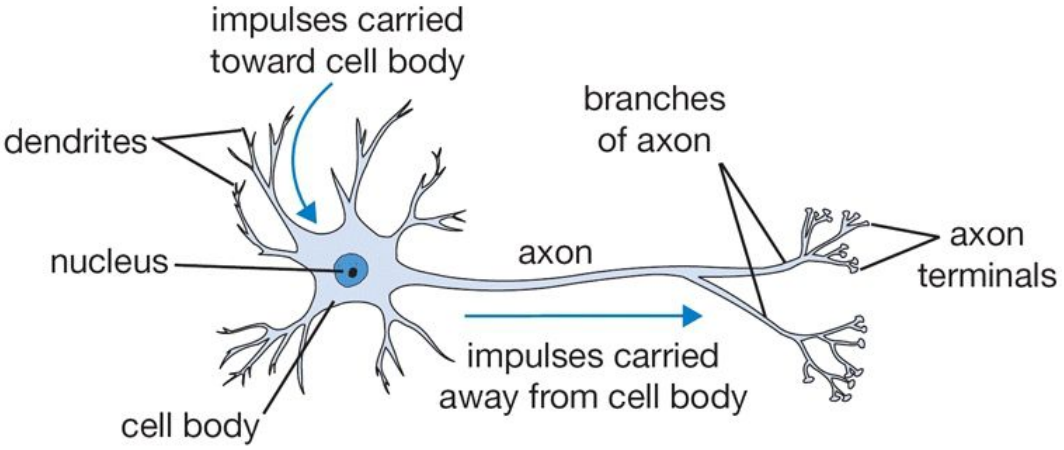
Deep Learning



Francois Chollet "Deep Learning with Python" Manning Publications Co.

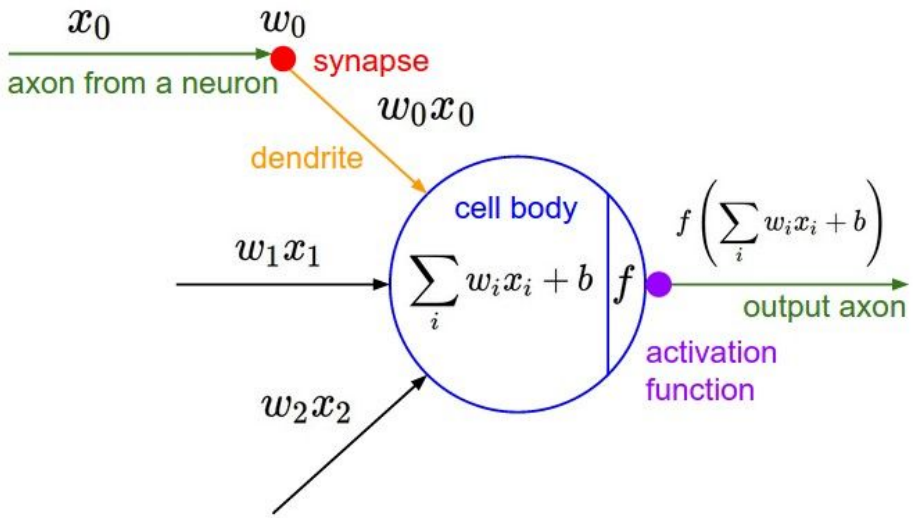
Neural Networks

- Initial goal: model biological neural systems
  - basic computational unit: neuron
  - ~86 billion neurons in the human nervous system
  - connected with ~10^{14k}-10^{15} synapses
  - signals on axons interact multiplicatively with dendrites of other neurons based on some synaptic strength



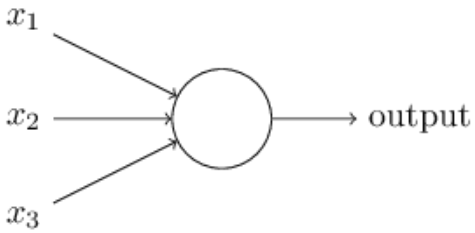
Neural Networks: Implementation

- Diverged from biological model
  - engineered to achieve good results in ML tasks (different from real neurons!)
  - idea: synaptic strengths can be learned
  - model: dendrites carry signals that get summed in the cell body; if sum is above some threshold neuron fires
  - neurons fire with a frequency that depends on the activation function



Perceptron

A perceptron takes several binary inputs, x_1,x_2,… , computes a weighted sum of the inputs and produces a single binary output using a fixed threshold:

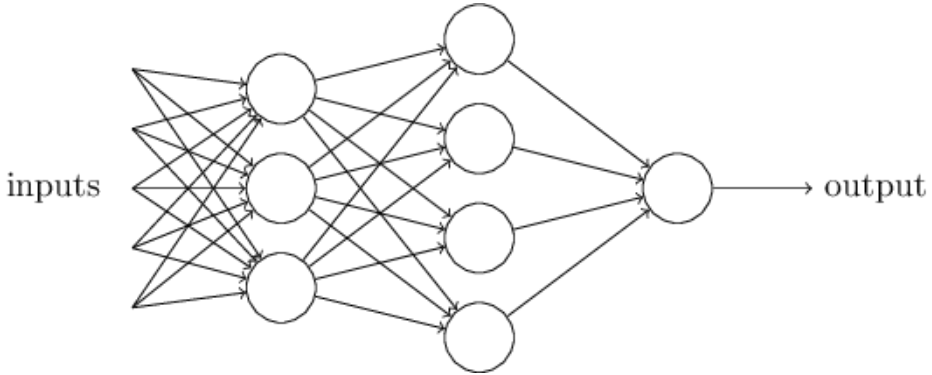


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

We can use the perceptron to take decisions: by varying the weights and the threshold, we can get different models of decision-making.

Multi-level perceptrons

More complex networks of perceptrons can deal with more complex decision problems:



The first column (i.e., the first **layer**)of perceptrons is making simple, low level decisions, by directly weighing the inputs.

The perceptrons in the second layer is making a decision by weighing the results from the first layer:

the second layer **can make a decision at a more complex and more abstract level**.

From perceptrons to artificial neurons

1) Write the weighted sum as dot product

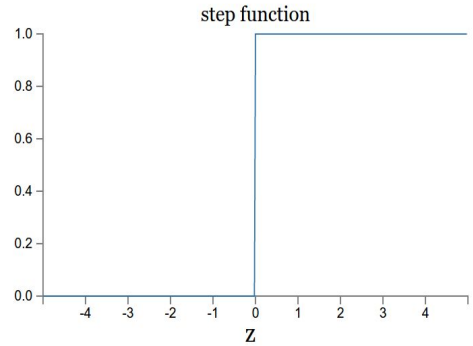
2) Replace the threshold with the bias

b = -threshold

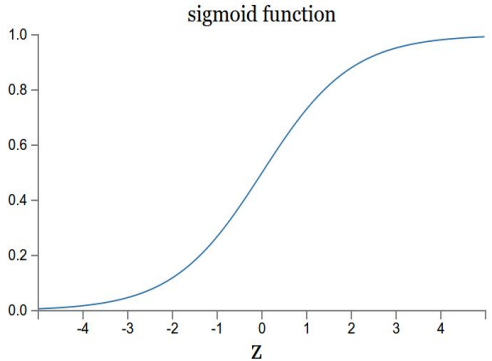
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

From perceptrons to artificial neurons

3) “Smooth” the output using the sigmoid function (a popular type of **activation function**):



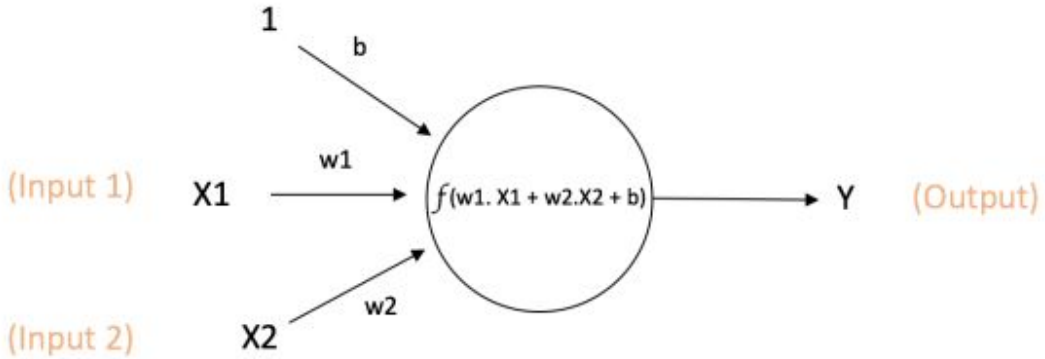
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$



This is called a **sigmoid neuron**: that small changes in the weights and bias cause only a small change in the output. That's the crucial fact which will allow a network of sigmoid neurons to *learn*.

Artificial Neuron

- takes numerical **inputs** (x)
- has a **weight** associated to each input (w)
- has a **bias** in the form of an additional input 1 with weight b
- applies an activation function (f) to the weighted sum of inputs



Output of neuron = Y= $f(w1.\ X1 + w2.X2 + b)$

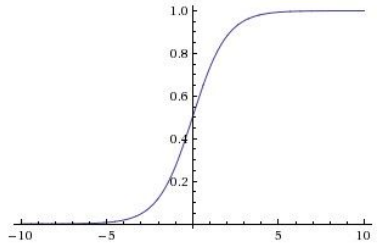
Activation Function

- It maps the resulting values into the desired range
- typically non-linear, aims at introducing non-linearity in the output of a neuron
- takes numbers as input
- performs a fixed mathematical operation on it

Activation Functions examples

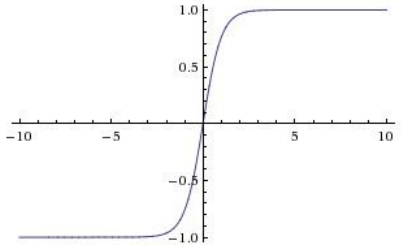
- **sigmoid** (bad!): takes a real-valued input and squashes it to range between 0 and 1

  $\sigma(x) = 1 / (1 + \exp(-x))$



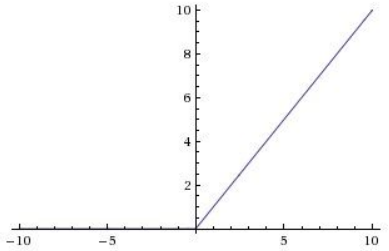
- **tanh**: takes a real-valued input and squashes it to the range [-1, 1]

  $\tanh(x) = 2\sigma(2x) - 1$



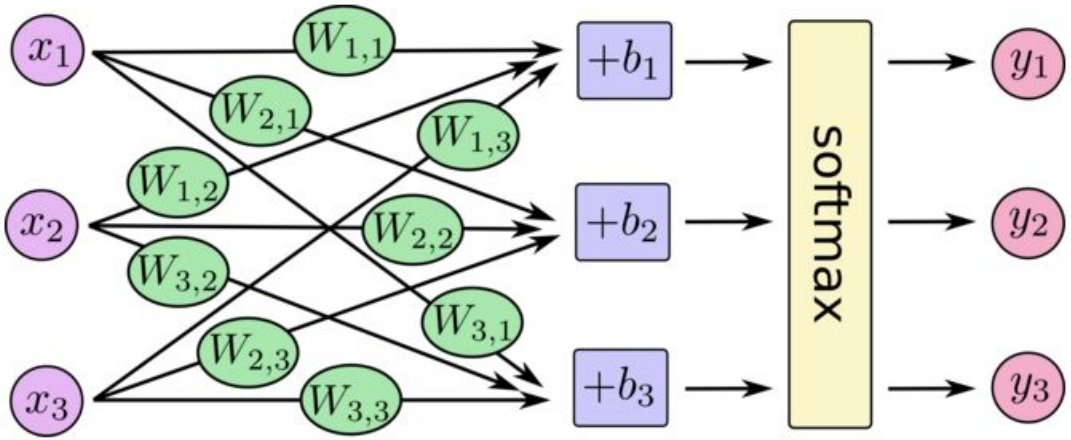
- **ReLU**: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)

  $f(x) = \max(0, x)$



Softmax

The **softmax** function takes a vector of arbitrary real-valued scores (in Z) and squashes it to a vector of values between 0 and 1 that sums to 1

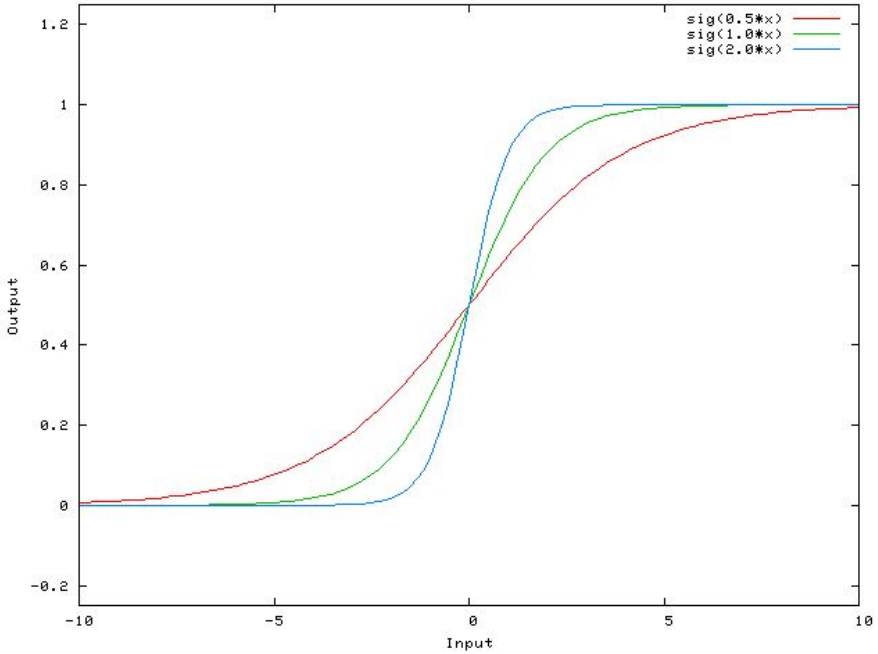


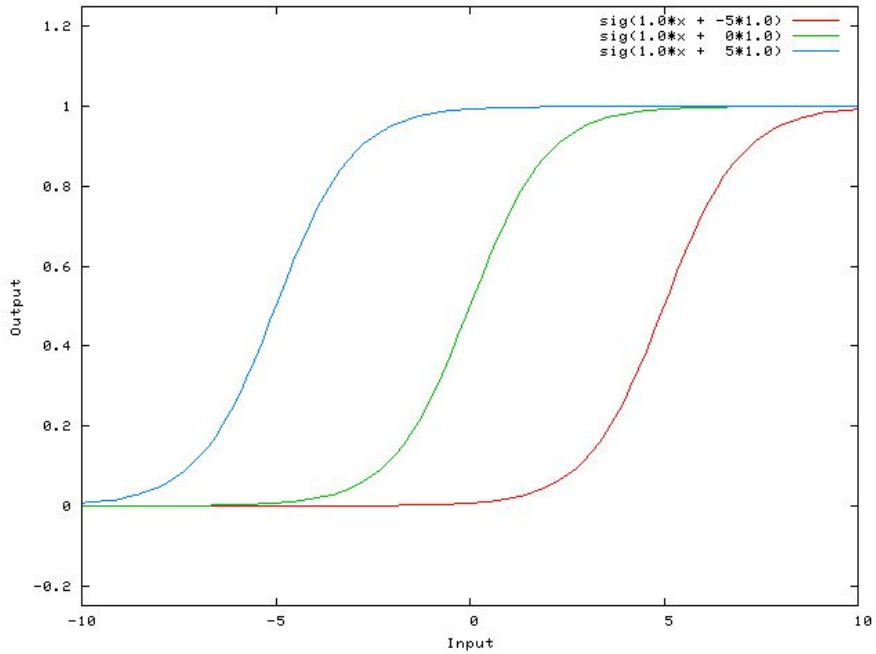
Softmax

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

The output of a softmax layer depends on the outputs of **all** the other neurons in its layer.

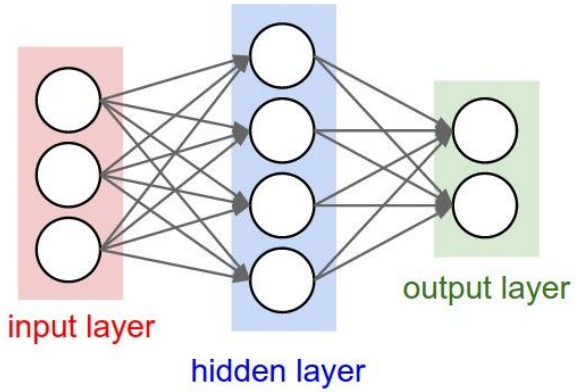
Role of Bias

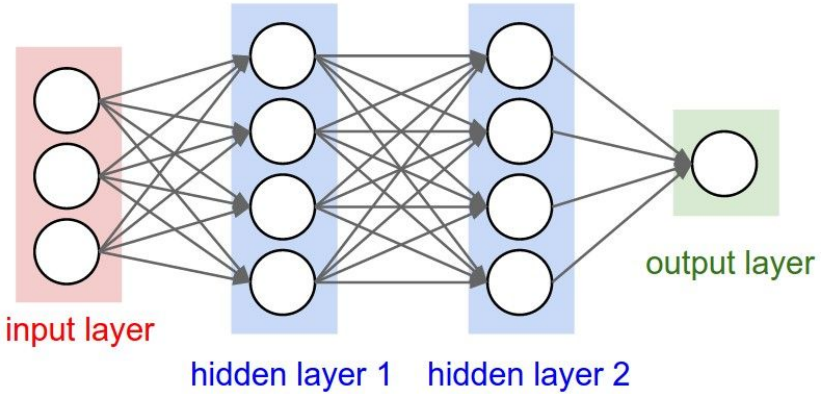

sig(0.5*x)
sig(1.0*x)
sig(2.0*x)
Output
Input
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-10
-5
0
5
10


sig(1.0*x + -5*1.0)
sig(1.0*x + 0*1.0)
sig(1.0*x + 5*1.0)
Output
Input
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-10
-5
0
5
10

Network Architecture

- regular neural networks are neurons connected in an acyclic graph
- 1 or more layers
- typically fully connected layers (no connection inside the same layer)
- output layer typically without activation function
- naming convention: input layer is not counted
  - single-layer networks directly map input to output

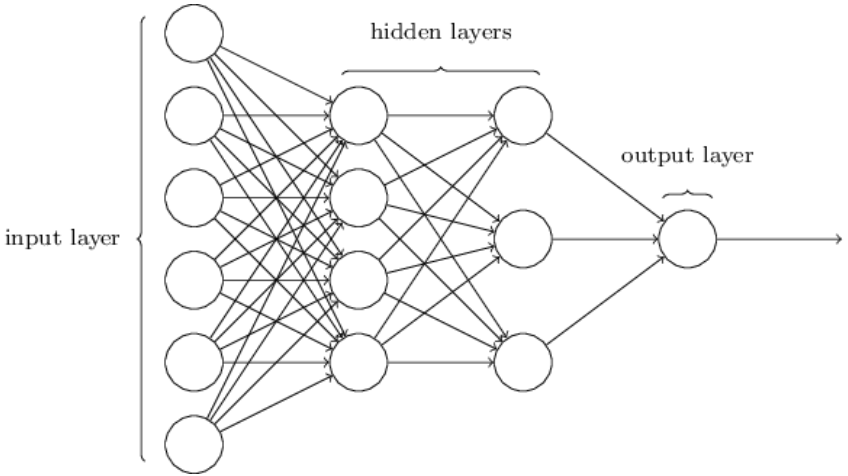




Deep Learning

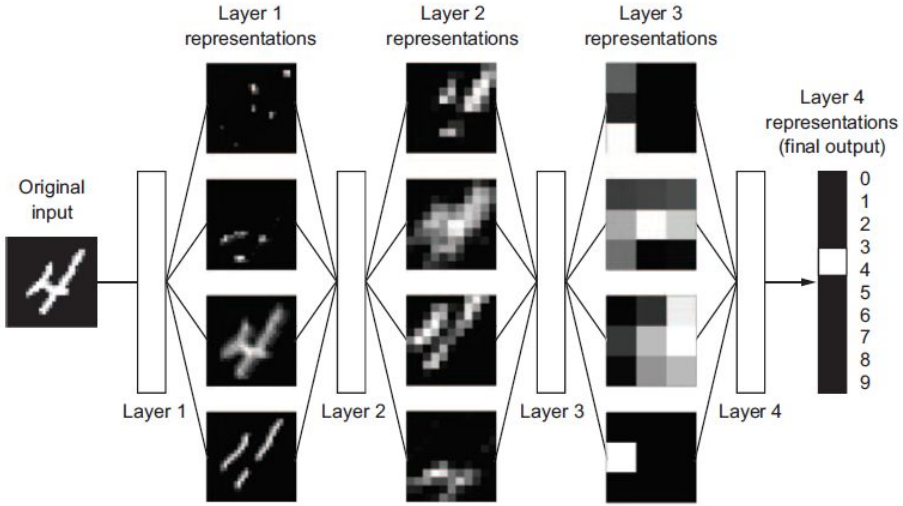
A network with hidden layers can break down a complicated question (e.g., does this image show a face or not?) into simpler questions

Two or more hidden layers → **deep neural networks**



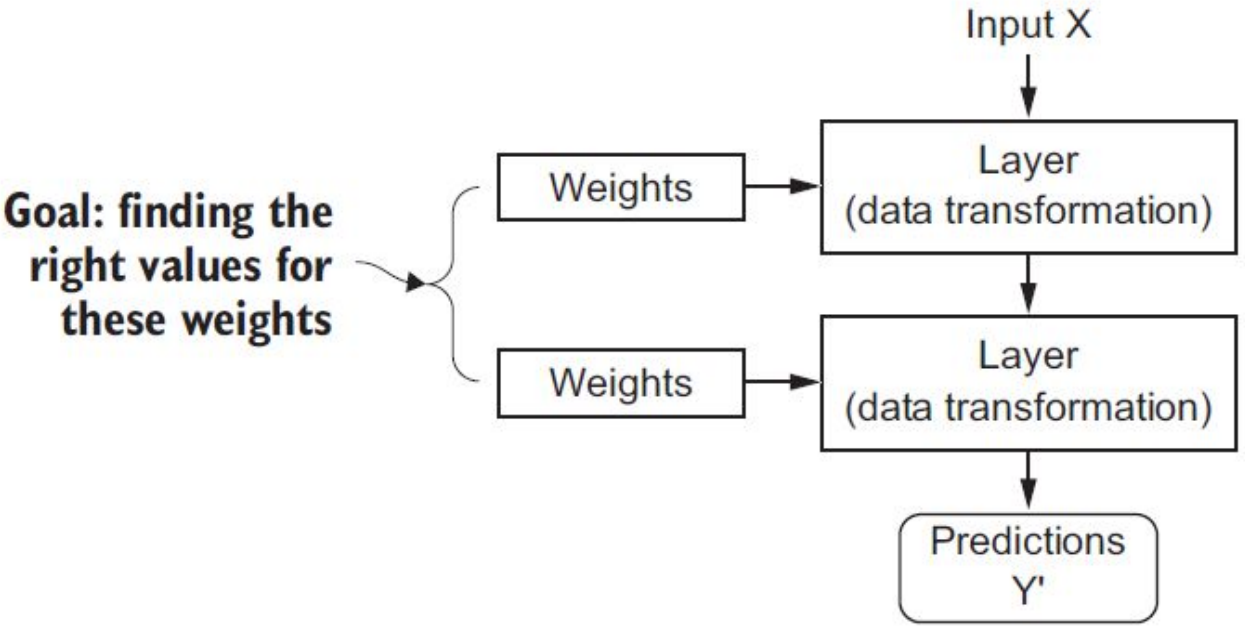
Deep Learning

Deep learning methods aim at learning “feature” hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. [Glorot and Bengio]



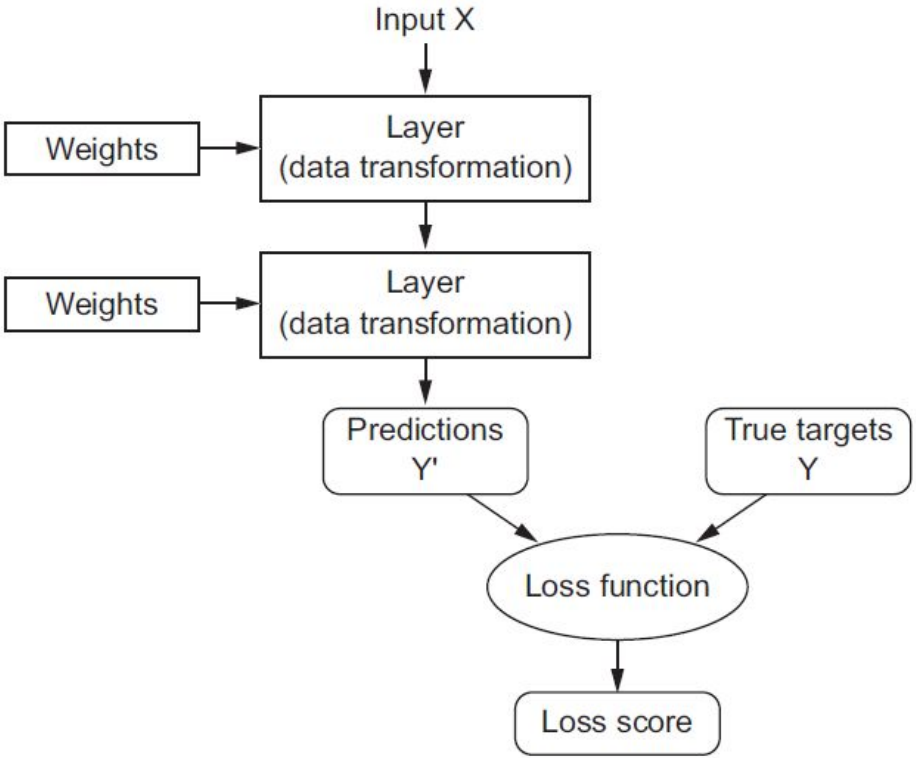
Francois Chollet "Deep Learning with Python" Manning Publications Co.

Weights (parameters)



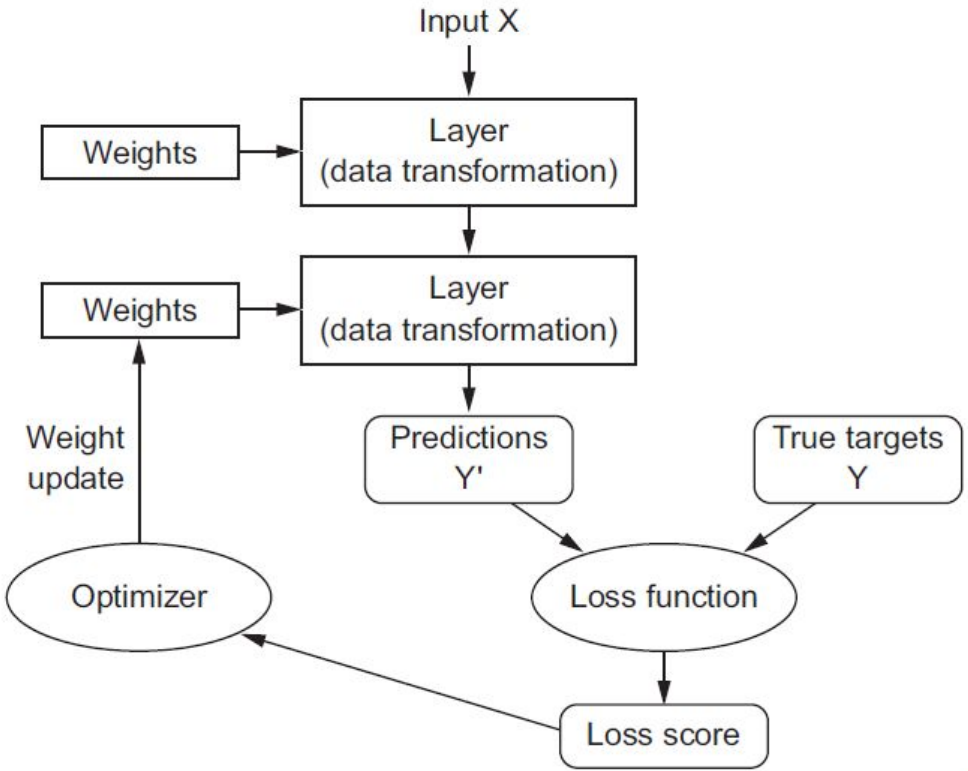
Francois Chollet "Deep Learning with Python" Manning Publications Co.

Loss Function



Francois Chollet "Deep Learning with Python" Manning Publications Co.

Adjust the weights



Francois Chollet "Deep Learning with Python" Manning Publications Co.

Learning the network parameters

Problem

Given a labeled dataset x = {x1, x2, …} of inputs with associated outputs y(x) = {y(x1), y(x2), …}, find the weights *w* and biases *b* that minimize the **cost function** (i.e., the error).

a is the output of the network given the current parameters *w* and *b*:

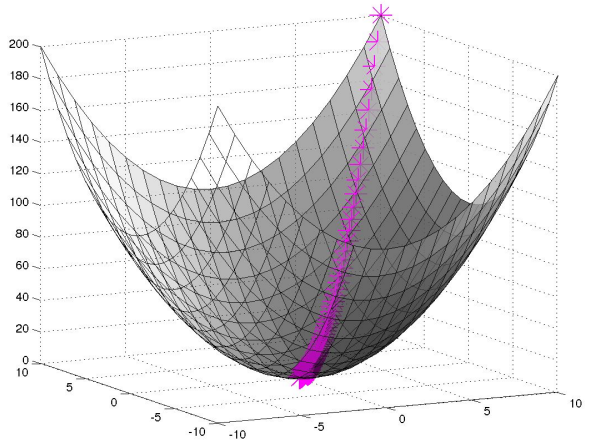
$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

Learning the network parameters

Easy answer! Gradient descent!

Correct but … very difficult implementation in practice, due to:

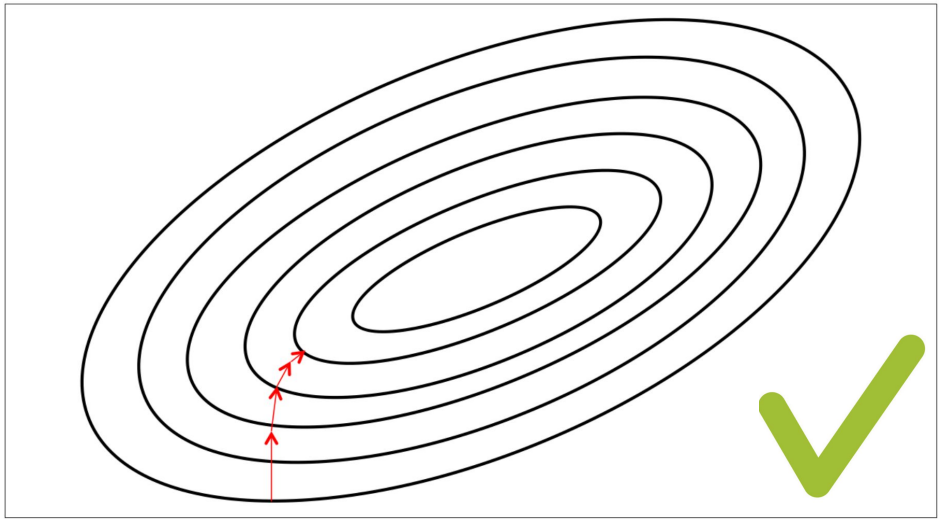
- Very large parameters set
- Very slow convergence rate
- Huge amount of data.
- Weight saturation
- ….

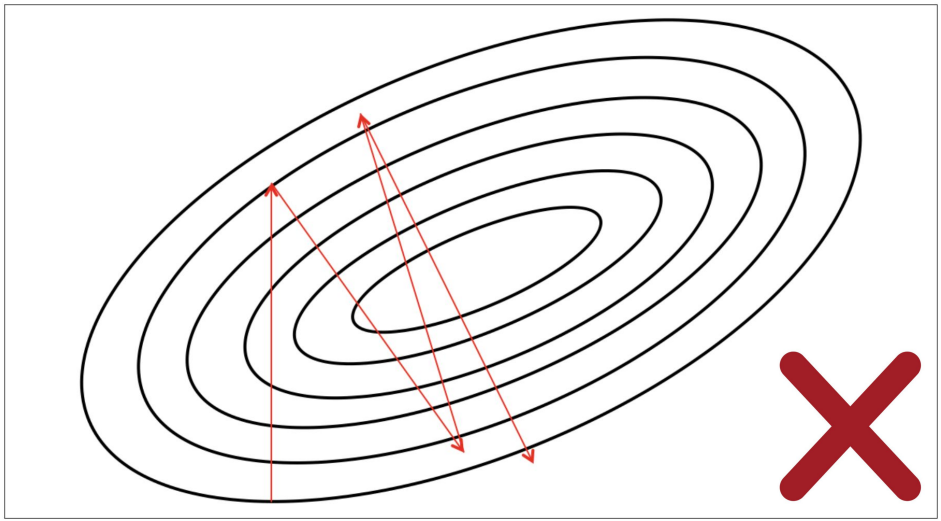


Learning rate

The closer we are to the minimum, the shorter we want to step forward.

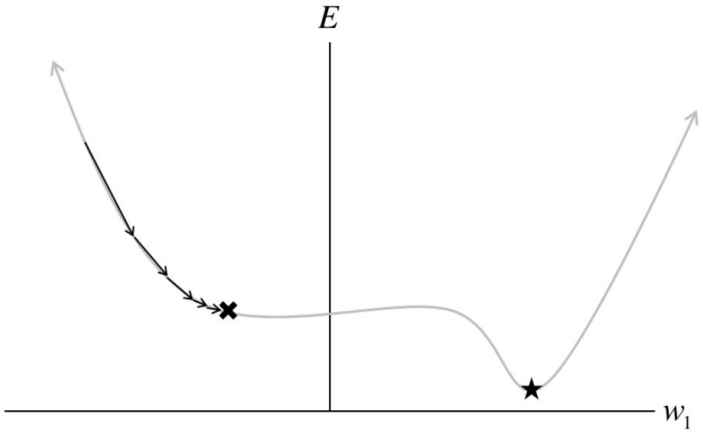
– Better use *adaptive learning rates*



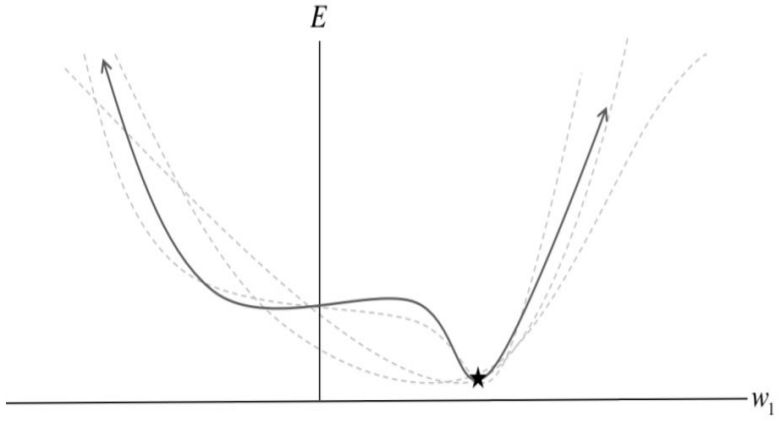


Stochastic Gradient Descent

Batch gradient descent is sensitive to saddle points, which can lead to premature convergence



In stochastic gradient descent, at every iteration, we compute the error surface with respect to some subset (minibatch) of the total dataset



Validation Set

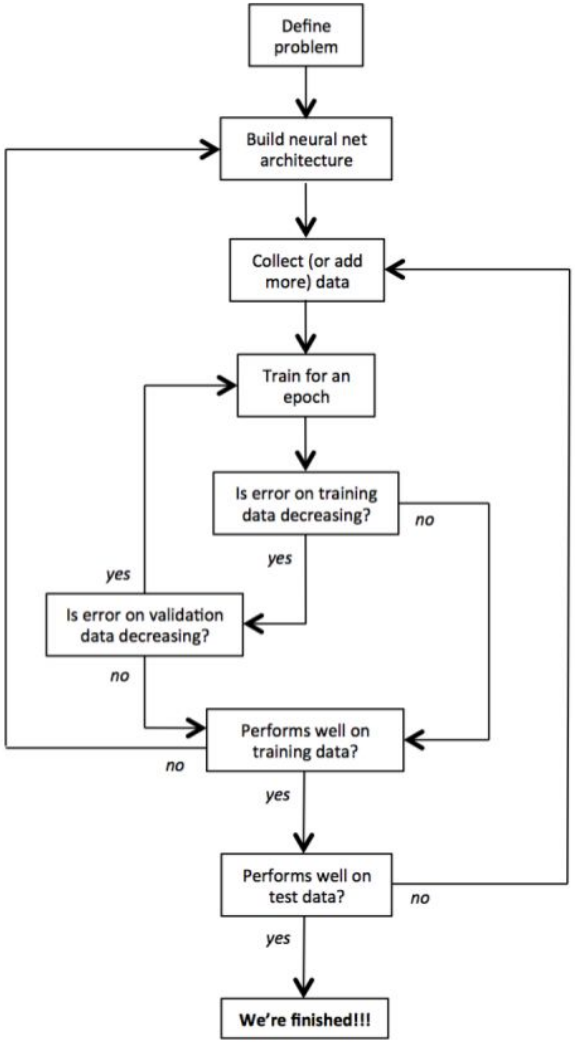
A validation set is used to prevent overfitting during the training process

Full Dataset:

Training Data	Validation Data	Test Data

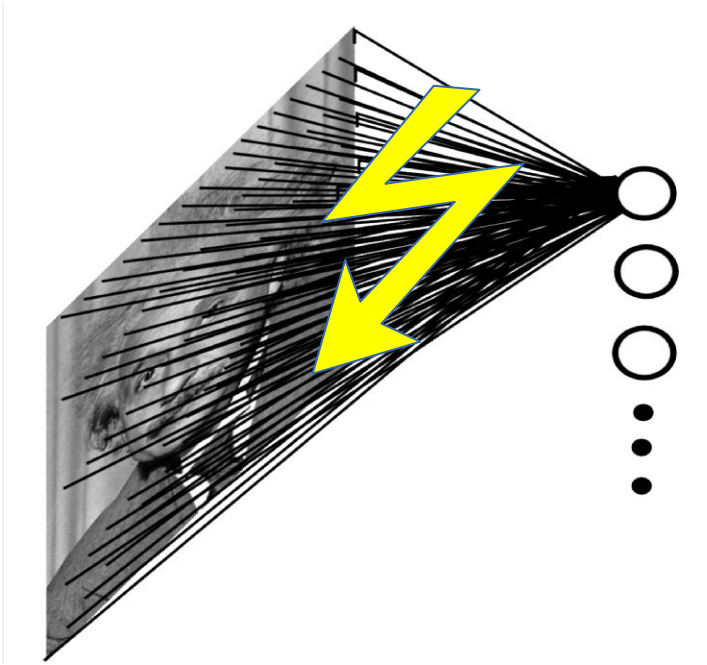
We divide our training process into **epochs**, i.e. is a single iteration over the entire training set.

If the accuracy on the training set continues to increase while the accuracy on the validation set stays the same (or decreases)→ **overfit!**



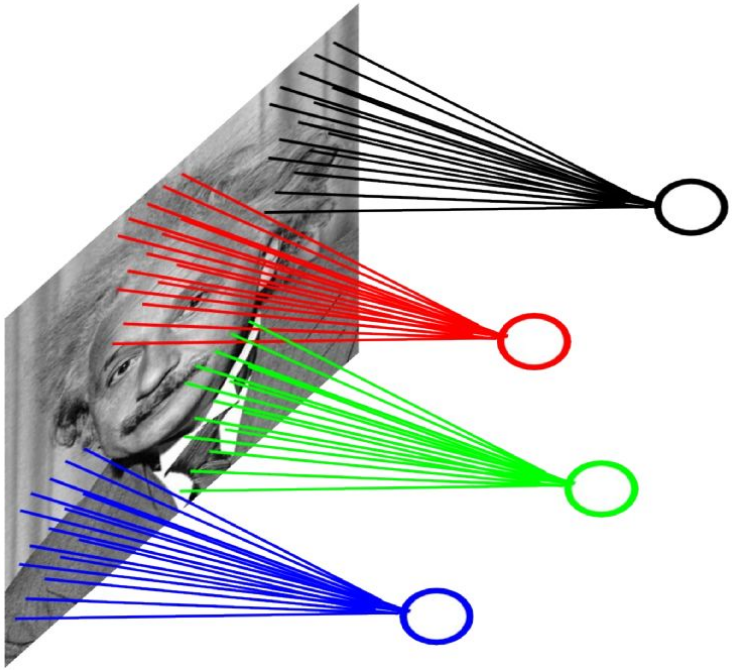
From Neural Network to CNNs

Apply NN to images to perform classification, detection,etc… using the classical "fully connected layers" but … for an RGB 200x200 image = **120000 parameters for each node**!!



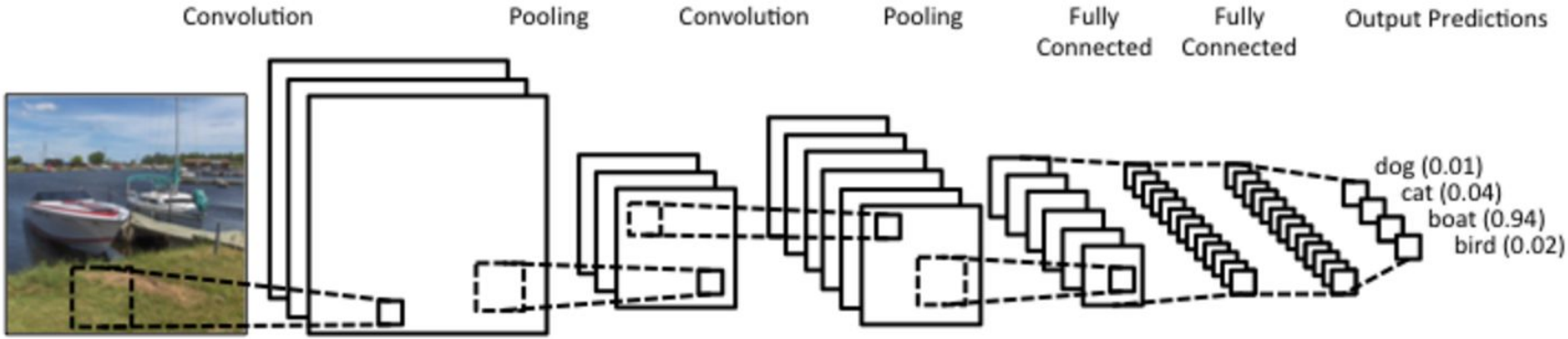
Convolutional Neural Networks

Convolutional neural n*etworks use three basic ideas: **local receptive fields**, **shared weights**, and **pooling**.



Convolutional Neural Networks

- unit connectivity pattern inspired by the organization of the visual cortex
- units respond to stimuli in a restricted region of space known as the receptive field
- receptive fields partially overlap, over-covering the entire visual field



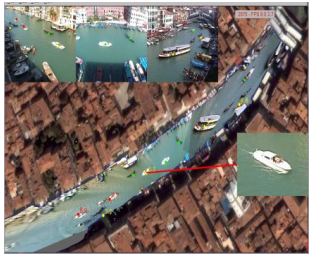
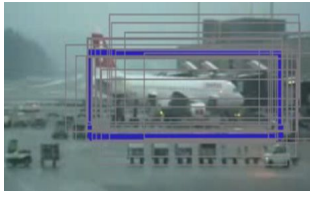
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

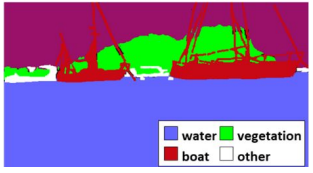


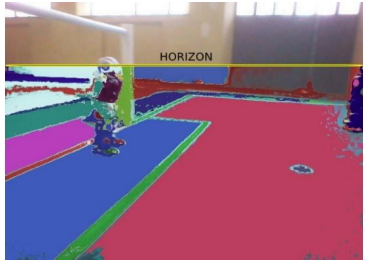
Corso di Sistemi Informativi
A.A. 2018/19

Docente
Domenico Daniele Bloisi

introduzione alle CNN




Giugno 2019