

Machine Learning and Pattern Recognition

Deep Learning

Alexandre BOULCH

Introduction

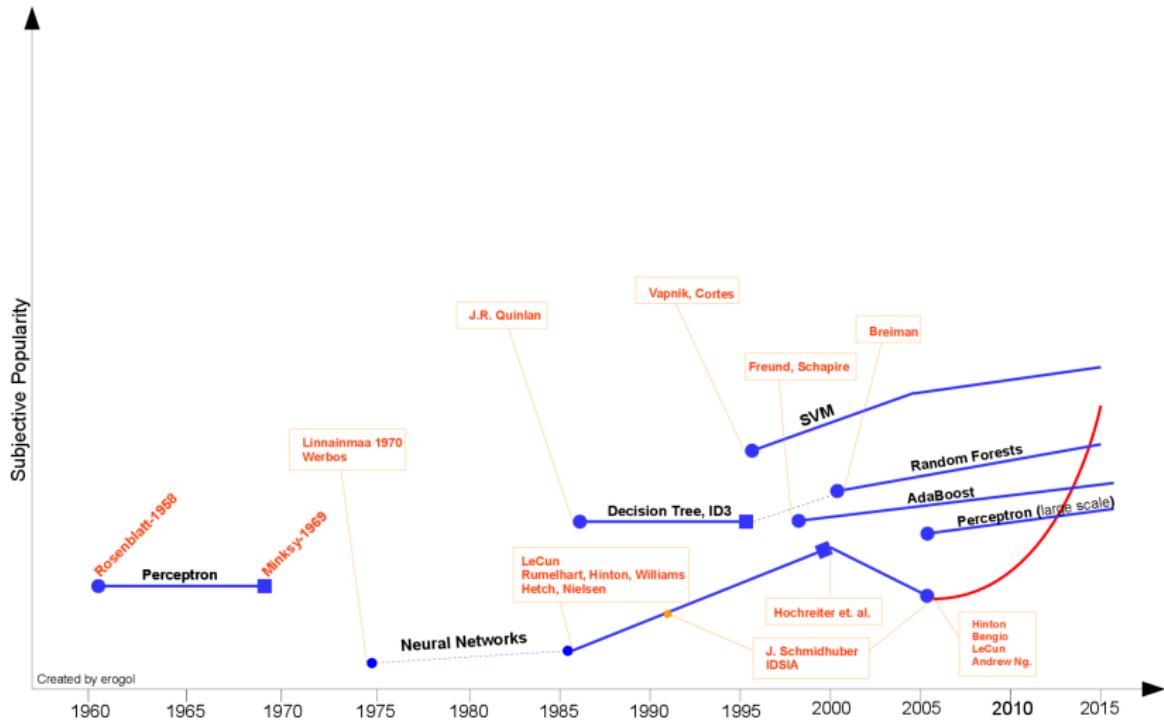


Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

2000's: better, faster, stronger

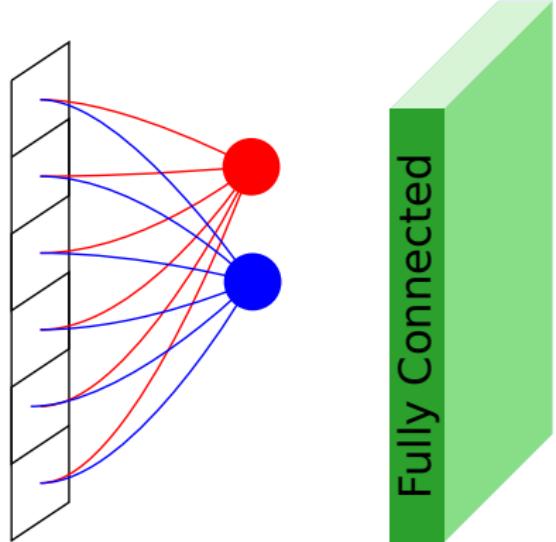
Do not forget the classics

What can we do with neural networks ?

Frameworks

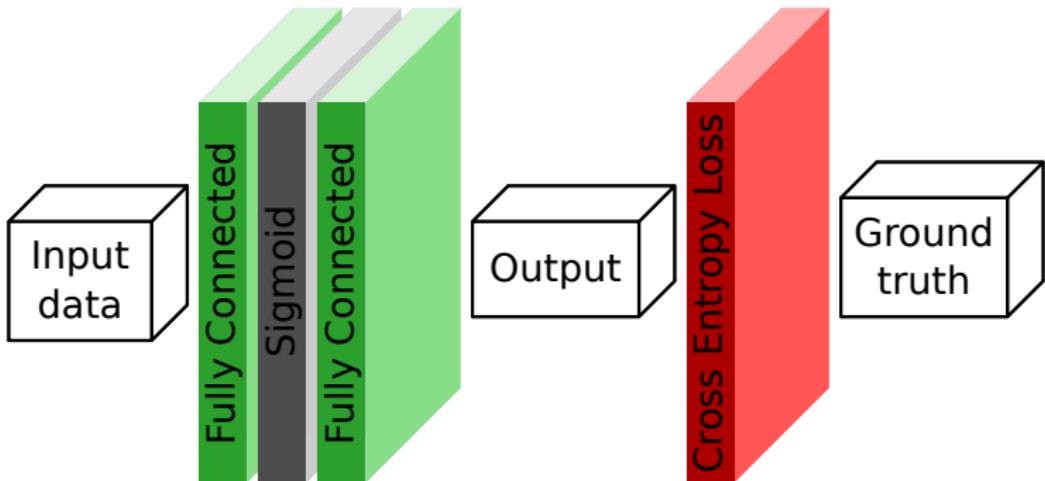
Fully connected

- ▶ Also called Linear layer or perceptron
- ▶ A neuron is connected to every input.
- ▶ High number of parameters ($|inputs| * |outputs|$)



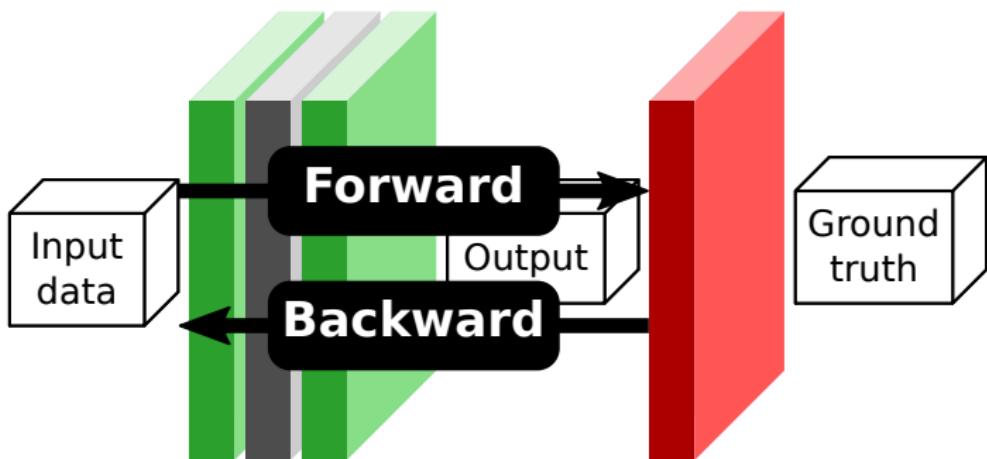
Multi Layer Perceptron

A stack of linear layers and activation functions (e.g. sigmoid).



Forward-Backward algorithm

- ▶ **Forward:** iteratively compute the output of each layer from data to prediction
- ▶ **Backward:** iteratively compute the derivative of the parameters starting from the loss



Forward-Backward algorithm

- ▶ **Forward:** iteratively compute the output of each layer from data to prediction
- ▶ **Backward:** iteratively compute the derivative of the parameters starting from the loss

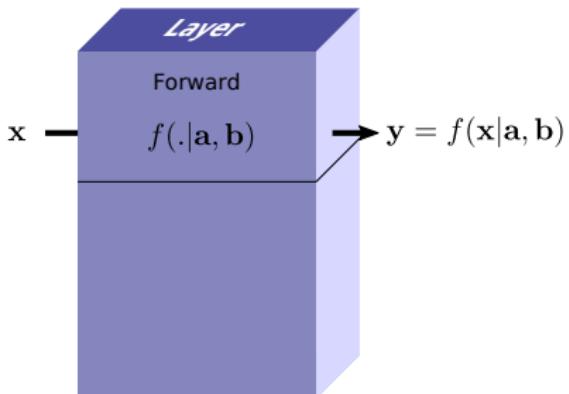
Chain rule

Easy differentiation is obtain by derivation of function composition:

$$\frac{\partial f(a = g(x, y, z), b, c)}{\partial x} = \frac{\partial g(x, y, z)}{\partial x} \frac{\partial f(a = g(x, y, z), b, c)}{\partial a}$$

Algorithm

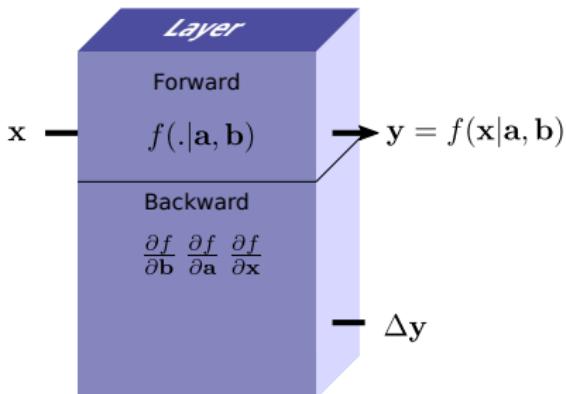
- ▶ Forward pass



Optimization: differentiation

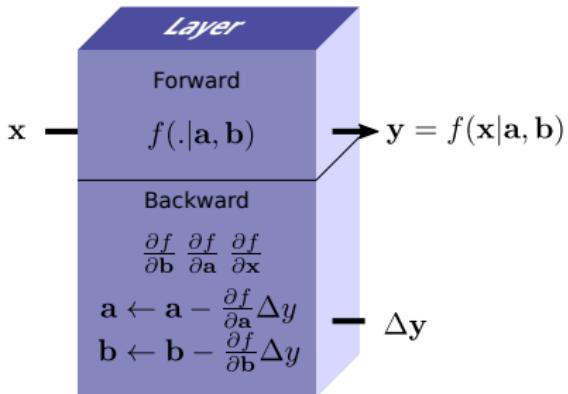
Algorithm

- ▶ Forward pass
- ▶ Get the output differential (Δy)
- ▶ Compute partial derivatives for parameters and input



Algorithm

- ▶ Forward pass
- ▶ Get the output differential (Δy)
- ▶ Compute partial derivatives for parameters and input
- ▶ Update the parameters



Algorithm

- ▶ Forward pass
- ▶ Get the output differential (Δy)
- ▶ Compute partial derivatives for parameters and input
- ▶ Update the parameters
- ▶ Compute differential for input

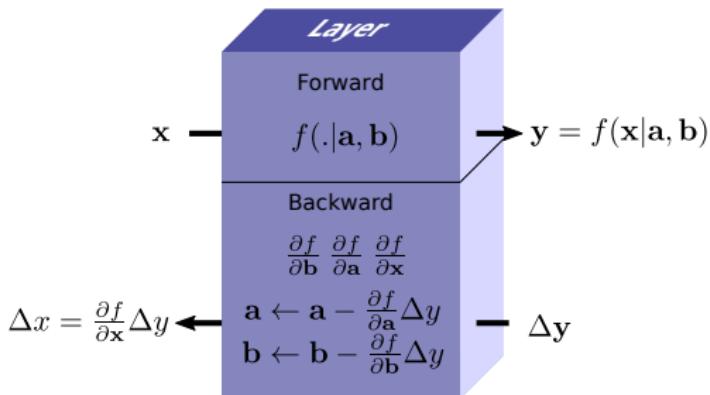


Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

2000's: better, faster, stronger

Do not forget the classics

What can we do with neural networks ?

Frameworks

Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

Concept

Convolutional networks

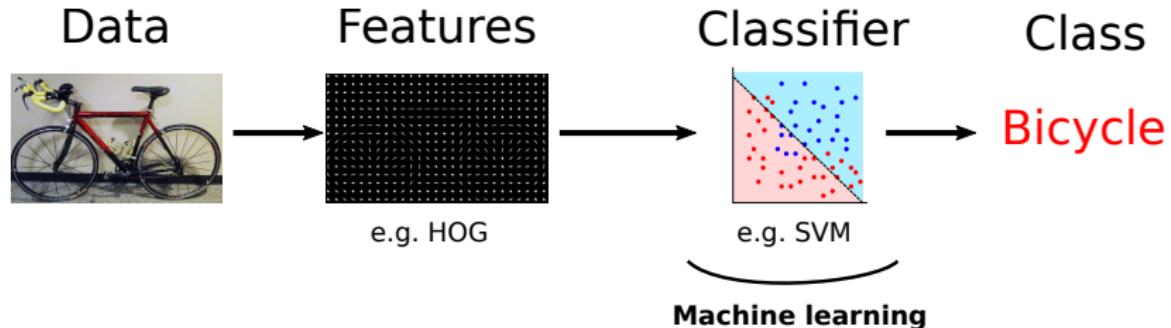
2000's: better, faster, stronger

Do not forget the classics

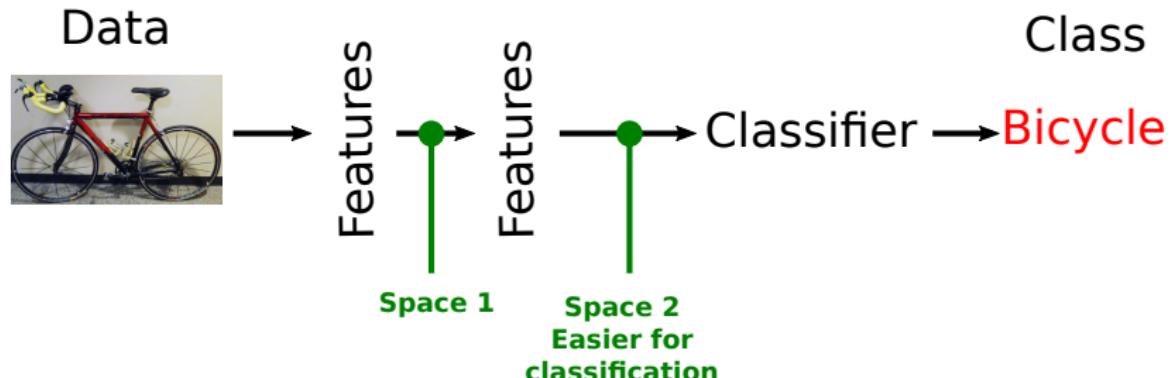
What can we do with neural networks ?

Frameworks

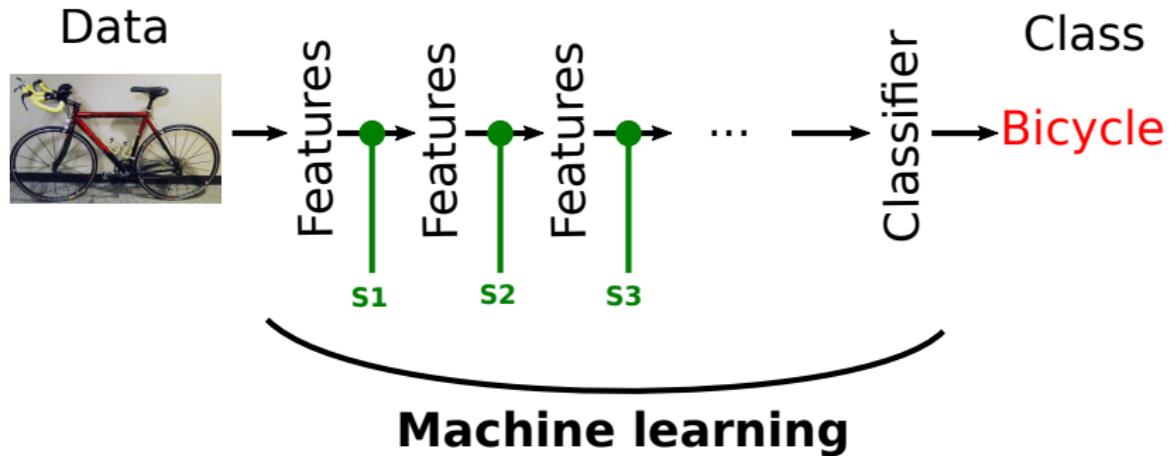
Deep learning concept



Deep learning concept



Deep learning concept



Concept

Deep learning is about learning representations.

Multi-layer perceptron

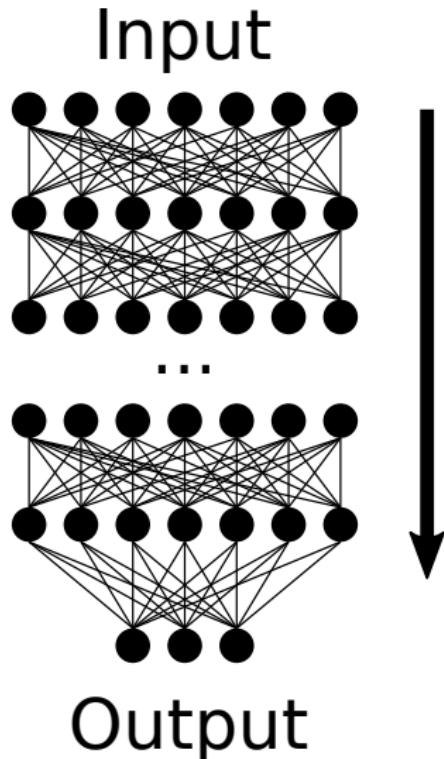
MLP becomes larger and deeper (before 1990)

- ▶ Difficult convergence
- ▶ Few data
- ▶ Very long training

→ Progressive loss of interest

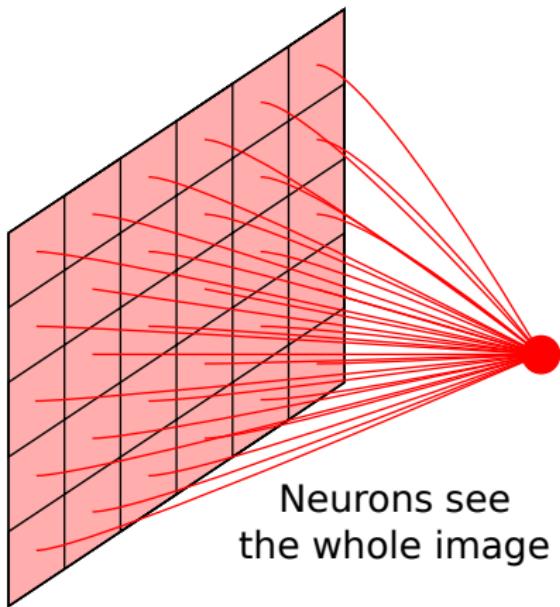
SVM

- ▶ Simple to use
- ▶ Convergence proof
- ▶ Fast



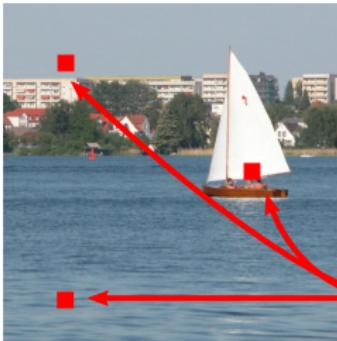
Using a linear layer ?

- ▶ Lot of weights ! (at least one per pixel !)



Using a linear layer ?

- ▶ Lot of weights ! (at least one per pixel !)
- ▶ Is it interesting to look at relations in the whole image ?



Look at
the whole
image

Idea ?

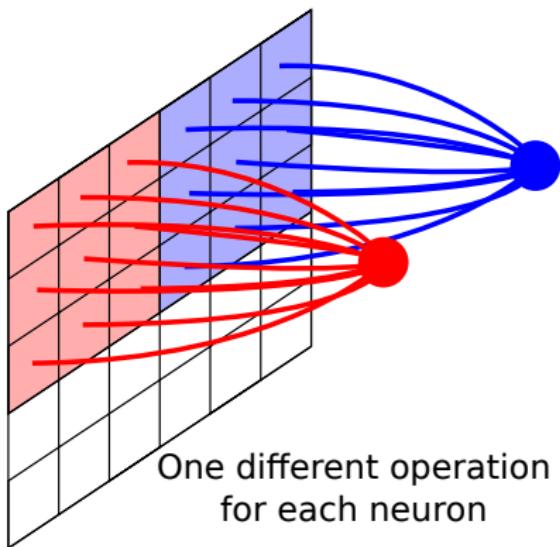
- ▶ Look at small neighborhoods (where the objects are)



Look at neighborhoods

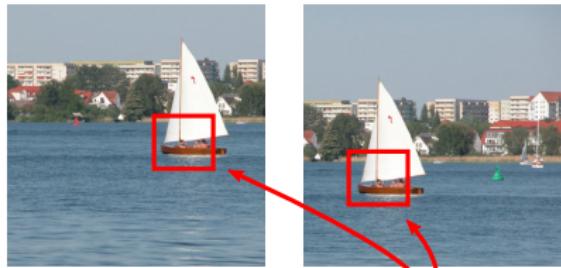
Idea ?

- ▶ Look at small neighborhoods (where the objects are)
- ▶ Create neurons that take a patch input



Problem ?

- ▶ Translation of the object must lead to same behaviour of the neurons



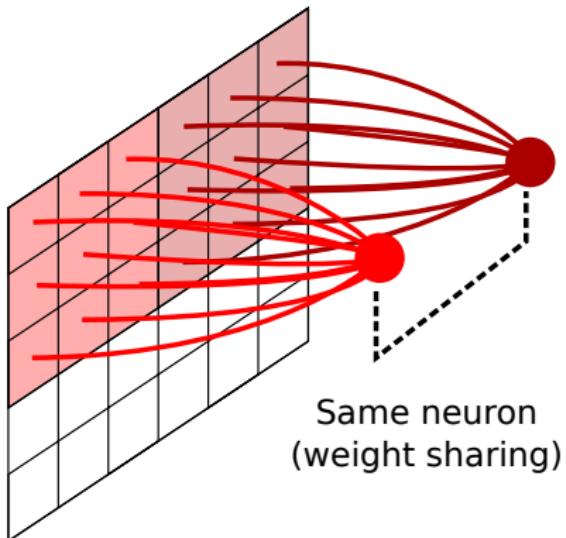
Same behaviour for the same pattern at different location

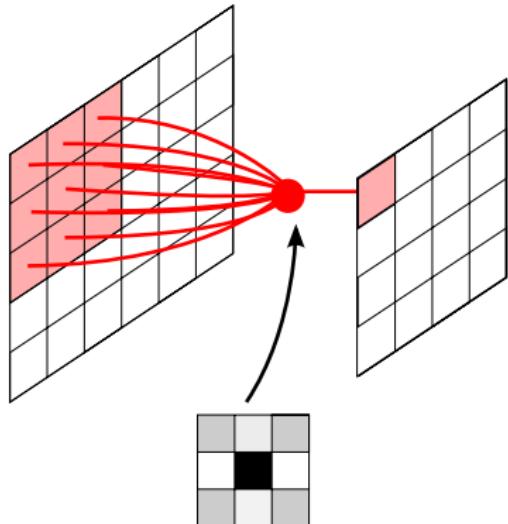
Problem ?

- ▶ Translation of the object must lead to same behaviour of the neurons

Solution ?

- ▶ Use the same neuron (i.e. all the neurons of the layer share weights)





Forward

Let (i, j) be the coordinates in the input map.

(k, l) be the size of the patch (size of the kernel, usually $k = l$)

Then:

$$y_{i,j} = \sum_k \sum_l w_{k,l} x_{i+k, j+l} + b$$

Convolution

Backward weight update

$$\frac{\partial y_{i,j}}{\partial w_{k,l}} = x_{i+k,j+l} \quad (1)$$

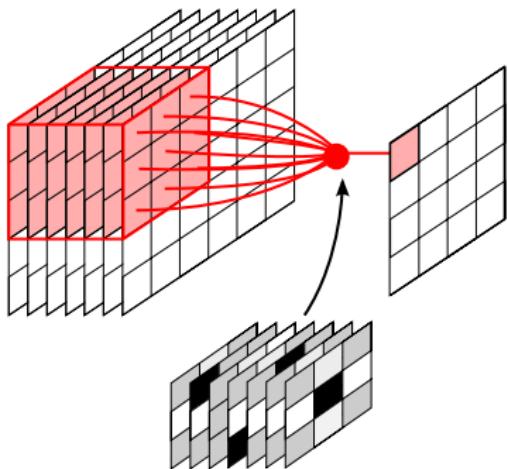
Let y be the output map and Δy be the gradient coming back:

$$\frac{\partial y}{\partial w_{k,l}} = \sum_i \sum_j x_{i+k,j+l} \quad (2)$$

Finally, the update rule:

$$w_{k,l} \leftarrow w_{k,l} - \alpha \frac{\partial y}{\partial w_{k,l}} \Delta y \quad (3)$$

$$\leftarrow \sum_i \sum_j x_{i+k,j+l} \Delta y_{i,j} \quad (4)$$



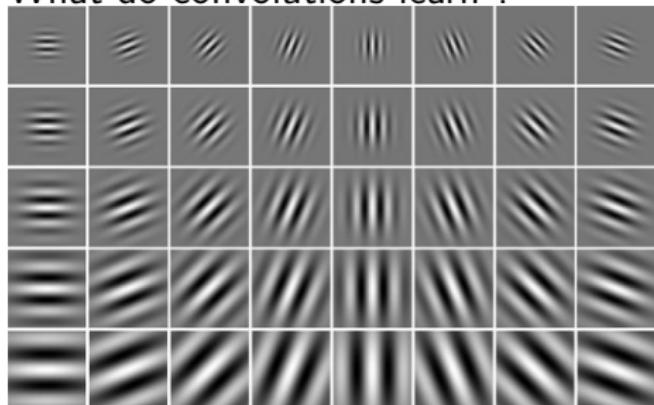
Forward

Same with term to term multiplication:

$$y_{i,j} = \sum_a \sum_b \mathbf{w}_{a,b} \mathbf{x}_{i+a,j+b}$$

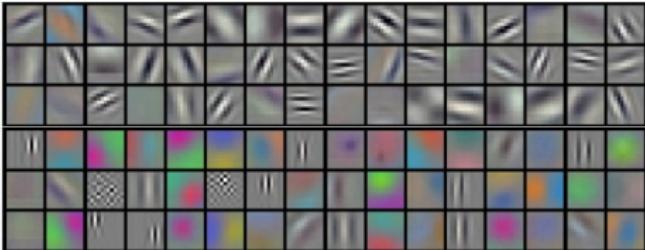
Convolution

What do convolutions learn ?



Gabor filters.

First layer of AlexNet.



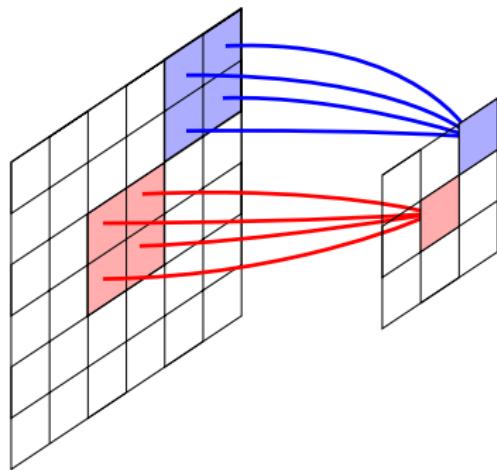
With the previous convolution, the output dimension is the same as the output dimension.

For classification: only one label, need for **dimension reduction**.

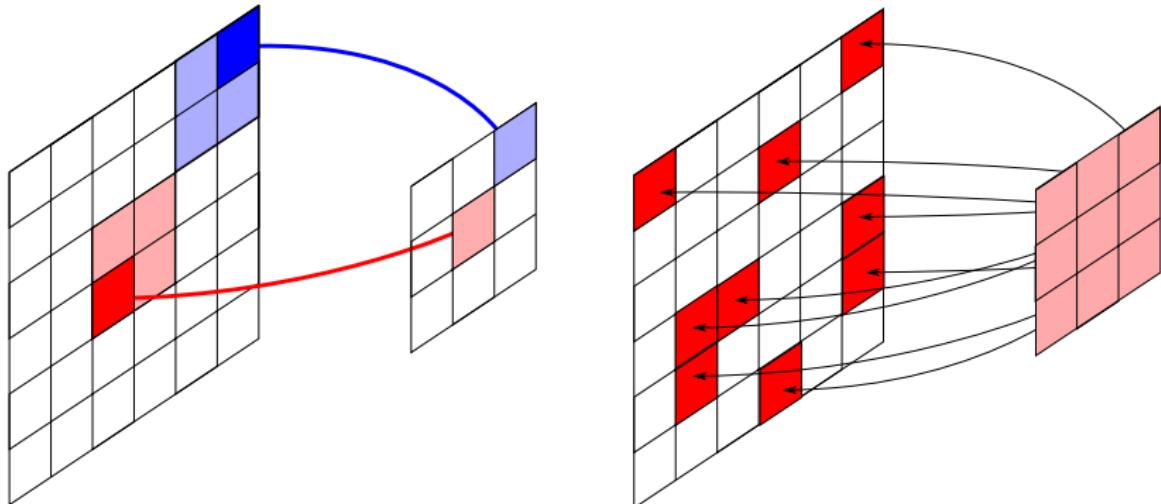
- ▶ convolution stride: do not look at all the pixels of the input (one every two, one every three...)
- ▶ **Max Pooling**

Max Pooling

- ▶ dimension reduction
- ▶ relative translation invariability



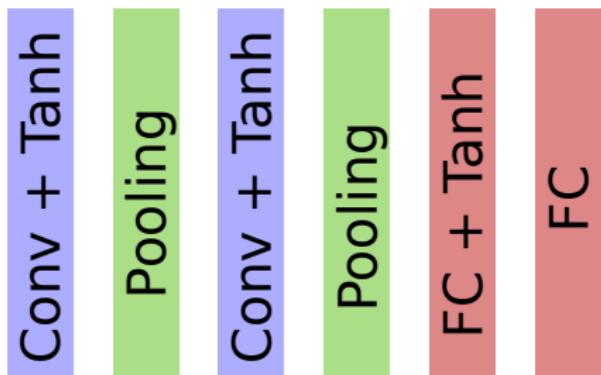
Max Pooling



Forward: max signal

Backward: Gradient transmission to max signal origin, zero otherwise

LeNet (1990)
Images 28x28



Very good results on digits recognition !

Issues

Issues

- ▶ Learning speed
- ▶ Exploding or vanishing gradients
- ▶ Overfitting
- ▶ Local minima

Limitations

- ▶ Architecture
- ▶ Initialization
- ▶ Computing power
- ▶ Data
- ▶ Optimization

Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

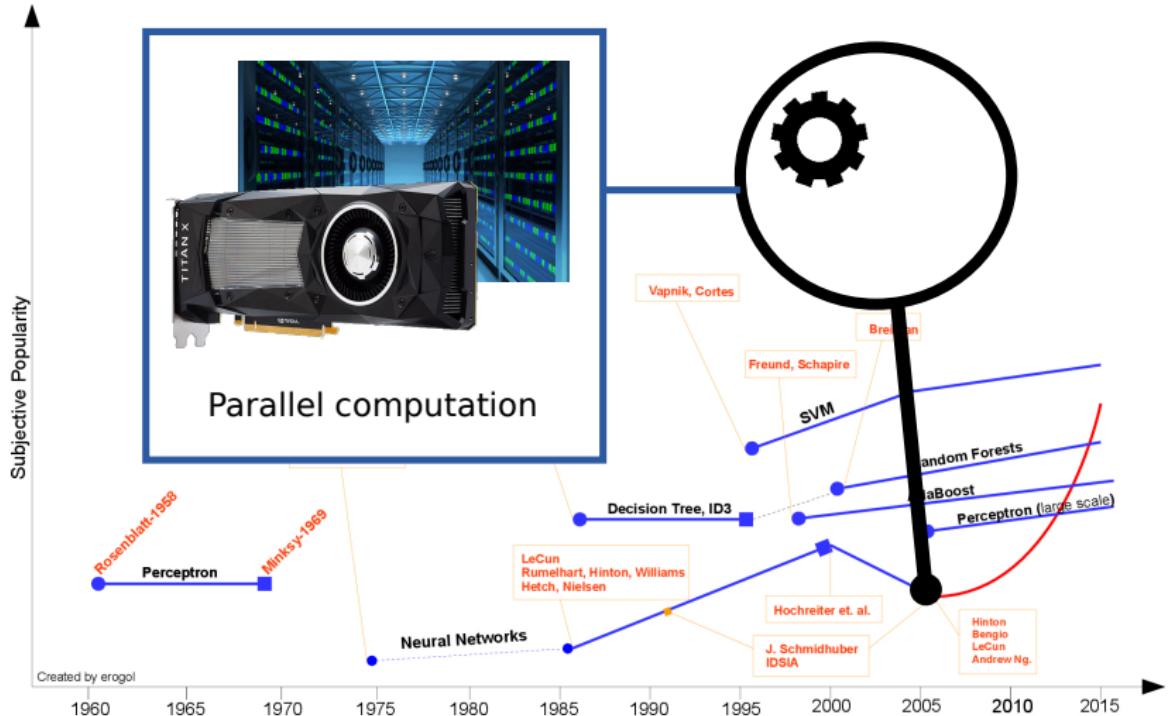
2000's: better, faster, stronger

Do not forget the classics

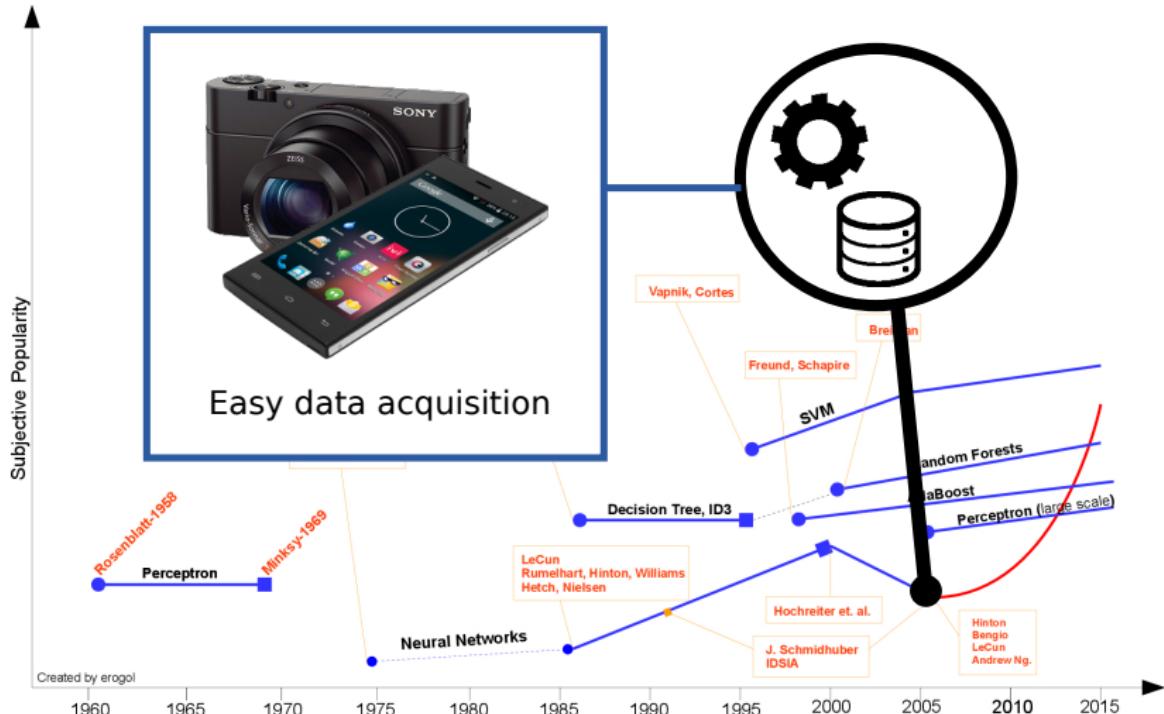
What can we do with neural networks ?

Frameworks

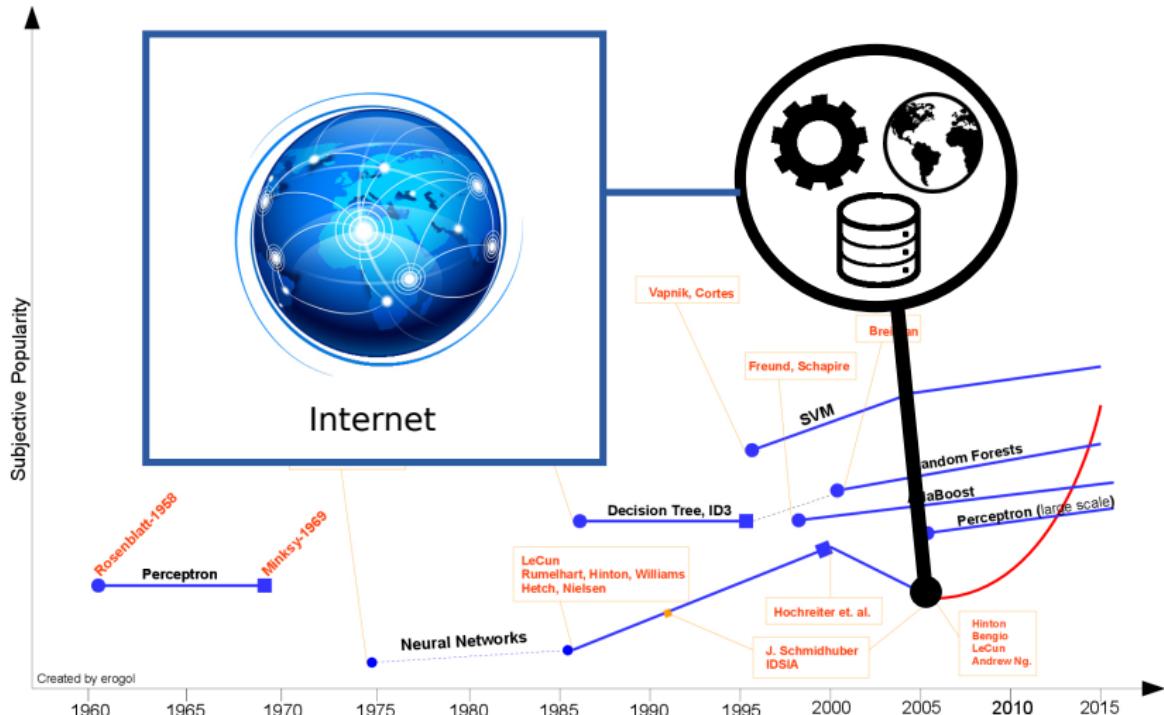
Massively data driven approaches



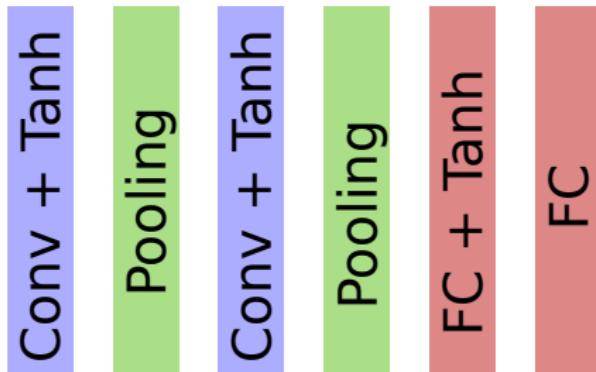
Massively data driven approaches



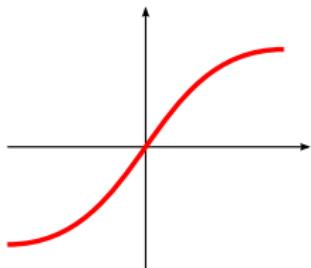
Massively data driven approaches



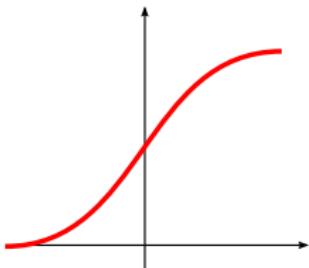
LeNet (1990)
Images 28x28



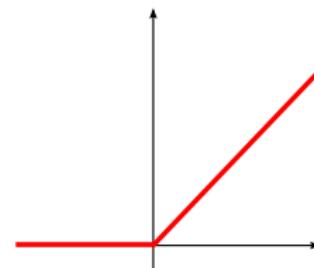
After each linear transformation (linear layer or convolution layer):
a non linear layer (activation) otherwise **linear + linear = linear**
(impossible to learn non linear decision).



Tangente
hypébolique



Sigmoïde

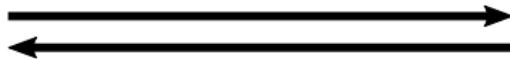
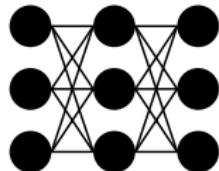


Rectified Linear
Unit (ReLU)

Rectified linear unit

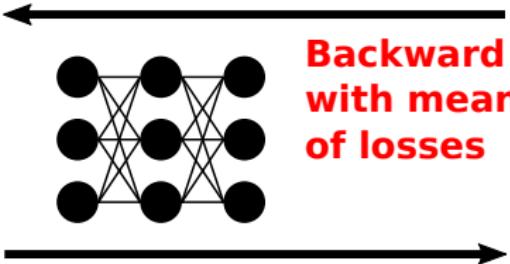
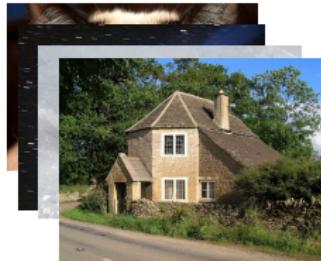
1. Faster gradient computation
2. Similar convergence

Mini-batches

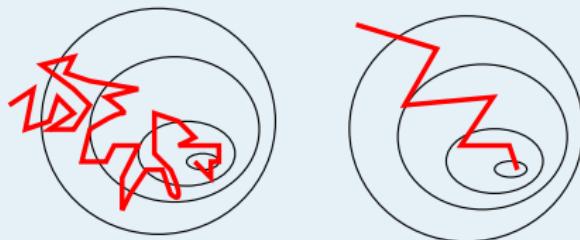


Loss image 1

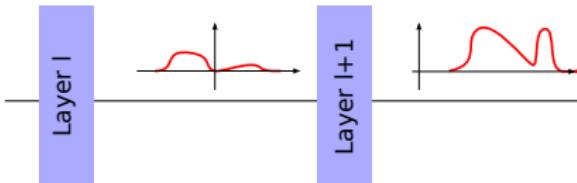
Mini-batches



Gradient smoothing



Smoother gradient converges faster.



Changes in the signal dynamic make the model more difficult to optimize: exponential or vanishing gradients.

Objective: control the signal distribution:

$$y^{I*} = \frac{y^I - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \beta \quad (5)$$

γ and β are learnt, μ and σ are computed (mean and standard deviation).

Learning is faster (iteration number) but slower (statistics computation).

Weight initialization

Weights have great influence on convergence speed. They are randomly initialized.

- ▶ too small weights: vanishing signal
- ▶ too high: exploding signal

Objective

Conservation of signal properties.

$$\text{Var}(Y) = \text{Var}(X)$$

Weight initialization

Xavier initialization

$X \in \mathbb{R}^n$, weights W and output $Y \in \mathbb{R}$

$$Y = W_1 X_1 + W_2 X_2 + \cdots + W_n X_n$$

X_i and W_i independent:

$$\text{Var}(W_i X_i) = E[X_i]^2 \text{Var}(W_i) + \text{Var}(X_i) \text{Var}(W_i) + \text{Var}(X_i) E[W_i]^2$$

$E[X_i] = 0$ and $E[W_i] = 0$:

$$\text{Var}(W_i X_i) = \text{Var}(X_i) \text{Var}(W_i)$$

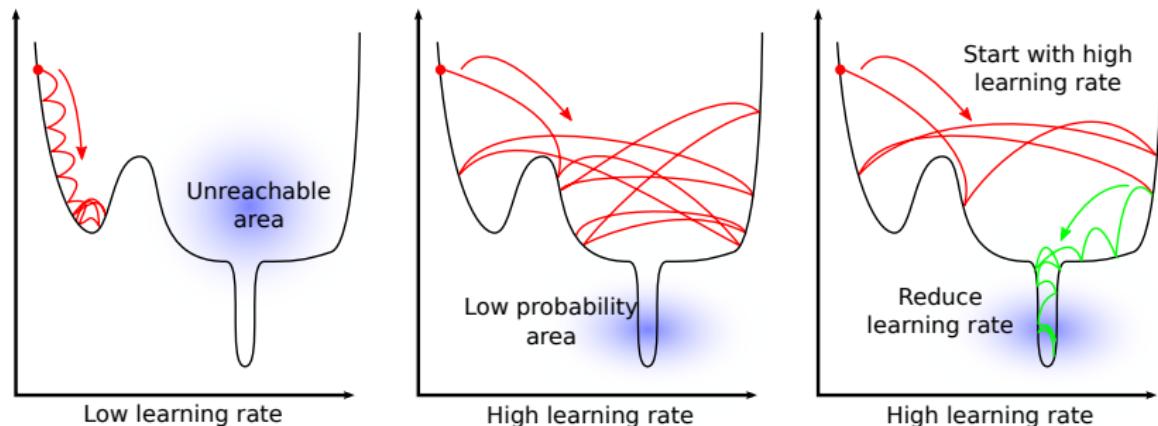
finally $\text{Var}(Y) = n \text{Var}(X_i) \text{Var}(W_i)$ and we chose

$$\text{Var}(W_i) = \frac{1}{n}$$

Optimization - Stochastic Gradient Descent

See neural network class

Update rule: $w_{t+1} = w_t + \alpha \Delta w$ (learning rate α)



See neural network class

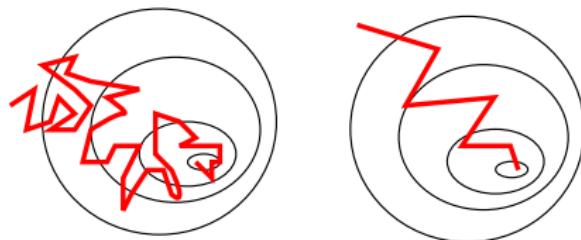
Update rule: $w_{t+1} = w_t + \alpha \Delta w$ (learning rate α)

Learning rate variation

- ▶ **Step decrease**
- ▶ Exponential decrease
- ▶ ...

Optimization - SGD with Momentum

Same idea as mini batch: smooth gradient in the good direction



Momentum

Use previous gradient to ponderate the direction of the new gradient.

$$v_t = \gamma v_{t-1} + \alpha \Delta w$$

$$w_t = w_{t-1} - v_t$$

γ is the momentum.

- ▶ **Adagrad**

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} \Delta w_i$$

with $G_{t,i}$ sum of gradients of w_i

- ▶ **Adam (adaptive momentum estimation)**

$$m_t = \frac{\beta_1}{1 - \beta_1} m_{t-1} + \Delta w_t$$

$$v_t = \frac{\beta_2}{1 - \beta_2} v_{t-1} + \Delta w_t^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} m_t$$

Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

2000's: better, faster, stronger

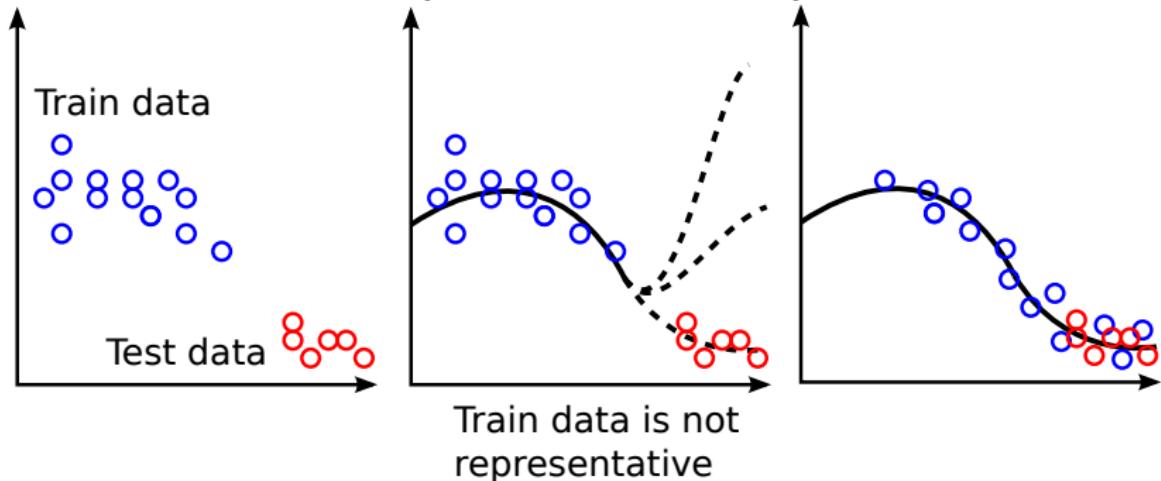
Do not forget the classics

What can we do with neural networks ?

Frameworks

- ▶ Representative
- ▶ Data augmentation
- ▶ Data normalization

Train data must be representative of the problem



Data normalization

- ▶ Compute mean μ and standard deviation σ on the train set.
- ▶ Normalize input I (train and test):

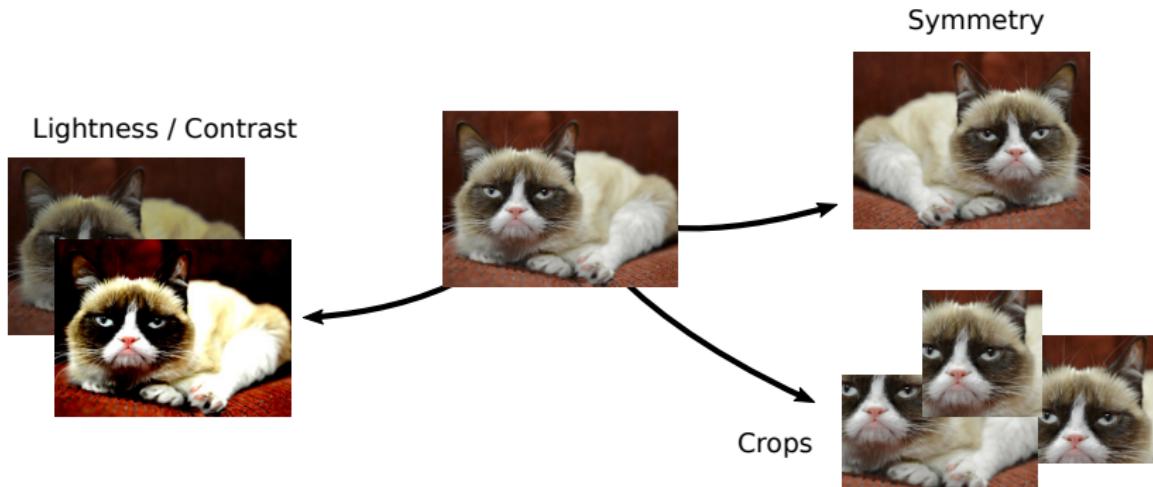
$$\hat{I} = \frac{I - \mu}{\sigma}$$

Data

Data augmentation

Random variations of input parameters (images: lightness, contrast ...)

- ▶ train on a more representative set
- ▶ avoid learning on unwanted features



Problems

- ▶ Small amount of data
- ▶ Low computational power

Solutions ?

- ▶ Use classical approaches (Perceptron, SVM, ...)

Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

2000's: better, faster, stronger

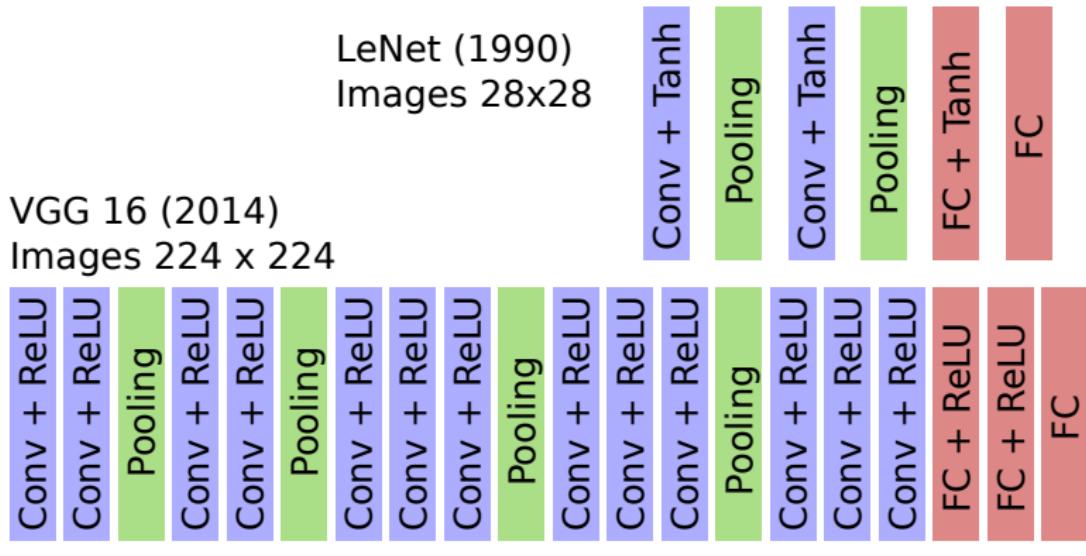
Do not forget the classics

What can we do with neural networks ?

Frameworks

Image classification - LeNet vs VGG

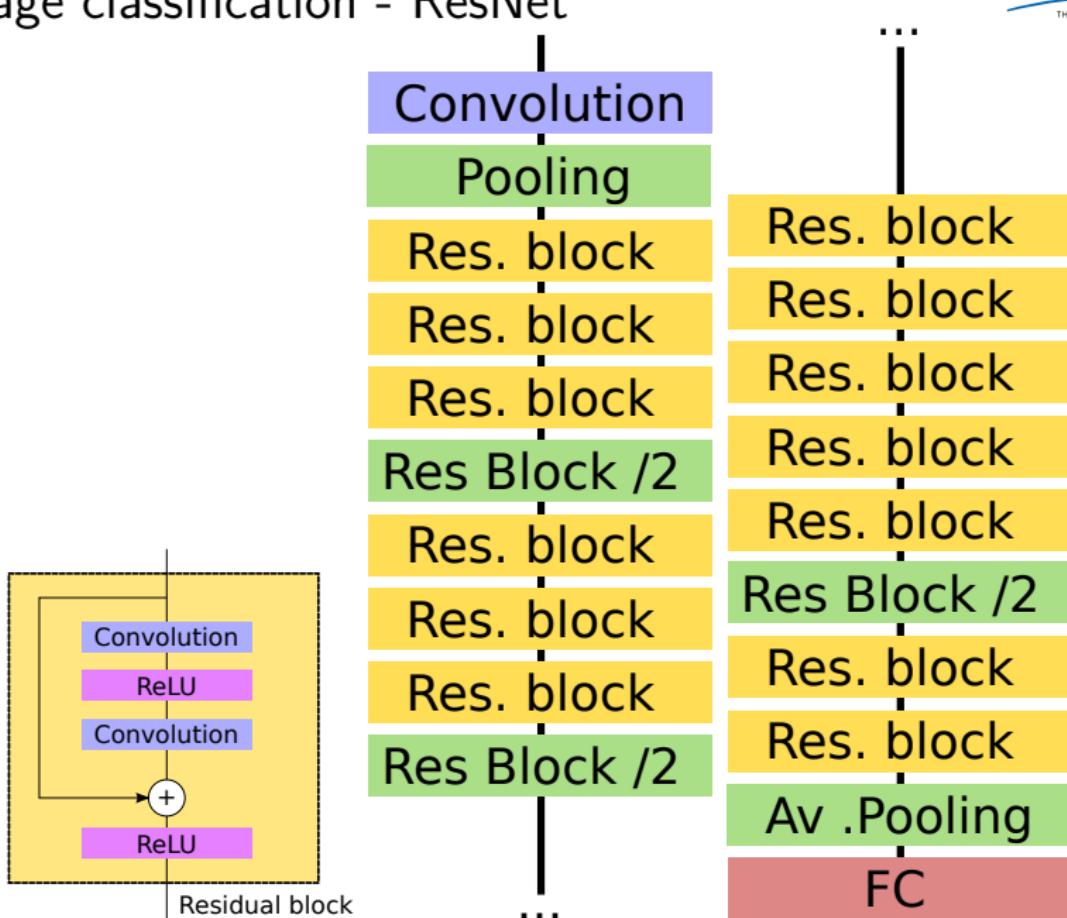
VGG 16 (2014)
Images 224 x 224



Feature extraction

Classification

Image classification - ResNet



Semantic segmentation

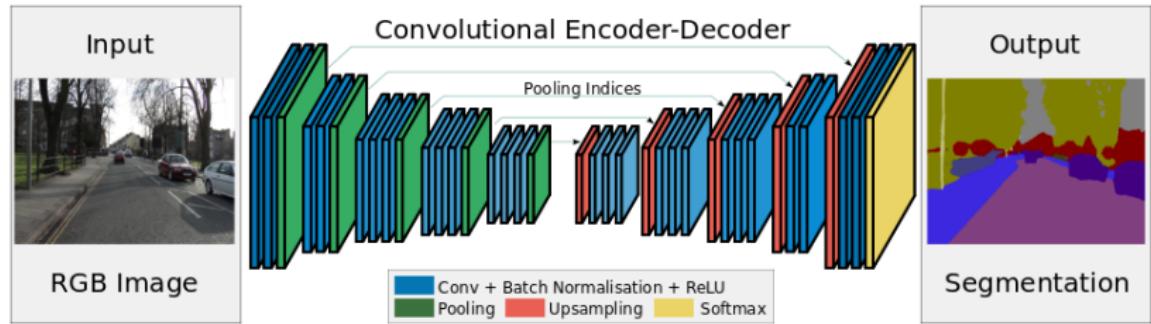
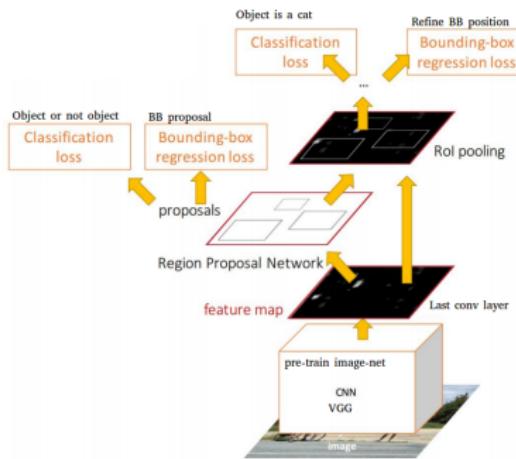
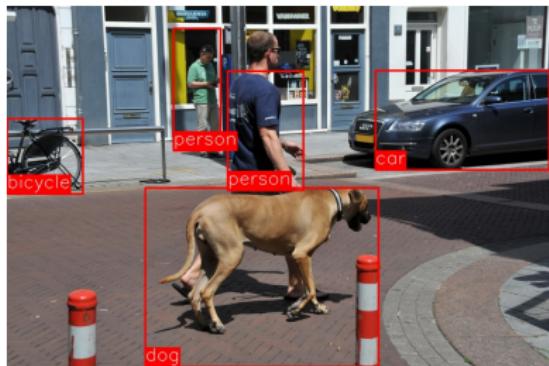


Image : <http://mi.eng.cam.ac.uk/projects/segnet/>

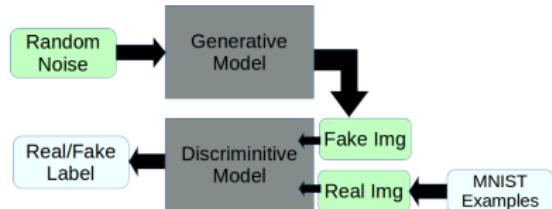
Detection



Data generation

Generative Adversarial Networks

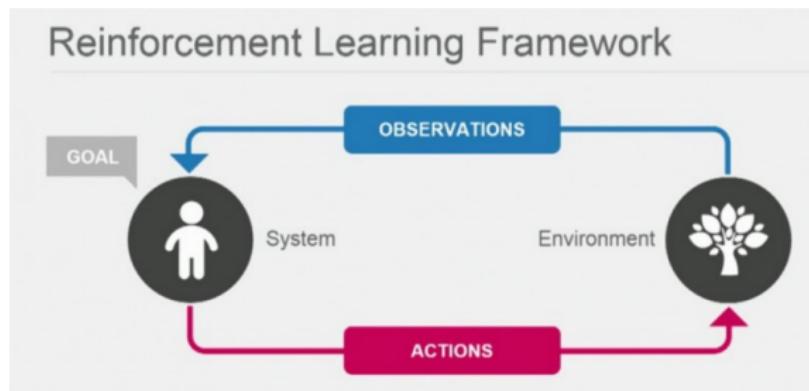
2 networks: one for data generation, one trying to discriminate real and generated data.



Reinforcement learning

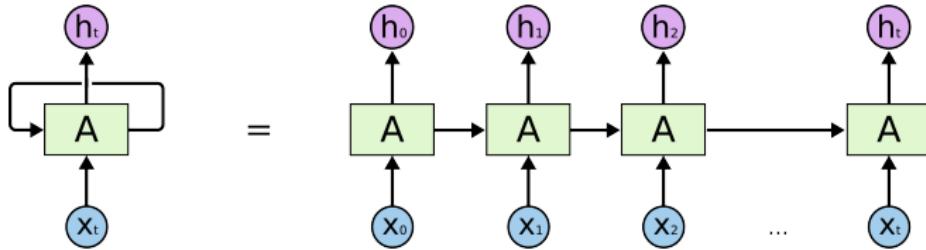
Try and error learning

Reward and Penalty.



1

¹Image [http://visteon.bg/2017/03/02/
machine-learning-algorithms-autonomous-cars/](http://visteon.bg/2017/03/02/machine-learning-algorithms-autonomous-cars/)



Dealing with sequences

Apply the same operation on each data and use a memory to propagate information in time.

Autoencoders

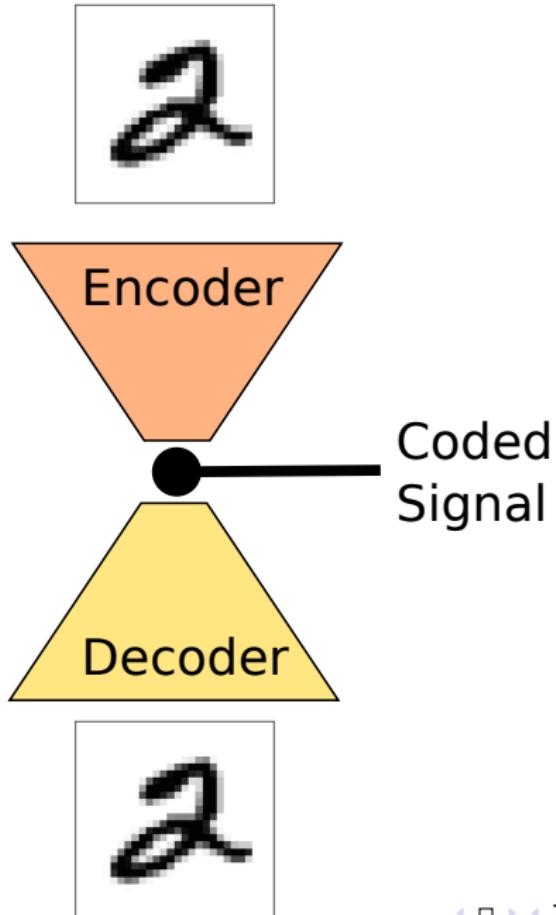


Table of contents

Neural networks classes

From 90's to Deep learning

Deep learning

2000's: better, faster, stronger

Do not forget the classics

What can we do with neural networks ?

Frameworks



Tensorflow and Keras

Principaux frameworks Python

Pytorch and Tensorflow: CPU and GPU, good maintenance, tutorials.

Keras run on top of Tensorflow (and Theano, CNTK...). It provides a high level API.

Conclusion

- ▶ Fast overview
- ▶ Few applications
- ▶ Practical work: create a CNN for image classification

Projects

Starting next week: projects.

- ▶ group of 2
- ▶ 3 subjects

For next week

- ▶ form working pair
- ▶ choose subject

Project 1: dehazing



URL

<https://competitions.codalab.org/competitions/21163>

Project 2: medical images segmentation



(a)



(b)



(c)



(d)

URL <https://chaos.grand-challenge.org>

Project 3: traffic sign recognition

- ▶ Classification
- ▶ 40+ classes
- ▶ 50000 images



URL <http://benchmark.ini.rub.de/?section=gtsrb&subsection=news>

Code (notebook) + **Report** (can be in the notebook)

Evaluation criteria

- ▶ Project execution (code)
- ▶ Justification of algorithmic choices
- ▶ Methodology
- ▶ Results analysis

Next class: **1h evaluation**

- ▶ All notes and course material
- ▶ **Except for Neural networks and deep learning**
 - knowledge and understanding test