

# Deep Learning

Aymeric DIEULEVEUT

May 2022

## 1 Goals

## 2 The first Neural Network: the Perceptron

## 3 Convolutional Neural Networks

## 4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# Outline

## 1 Goals

## 2 The first Neural Network: the Perceptron

## 3 Convolutional Neural Networks

## 4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# Deep Learning in one slide + Goals of the afternoon

- **How does it work:**

- ▶ Automatically learn representations of observations
- ▶ Learn highly non-linear models.

- **What does it require:**

- ▶ Large datasets with structure
- ▶ Computational power

- **Why now:**

- ▶ Combination of the 2 points above
- ▶ investment !

- **Some Applications**

- ▶ Image classification; object / face recognition
- ▶ Self driving cars
- ▶ Automatic Translation, Information extraction
- ▶ Caption Generation
- ▶ Ads, recommendation systems, etc.

**Goals:** Understanding core concepts of Deep Learning.

- When and how it works on paper (data, models, architecture)
- How to implement a simple neural network with Python.
- Overview of some of the main applications and challenges.

# Machine Learning

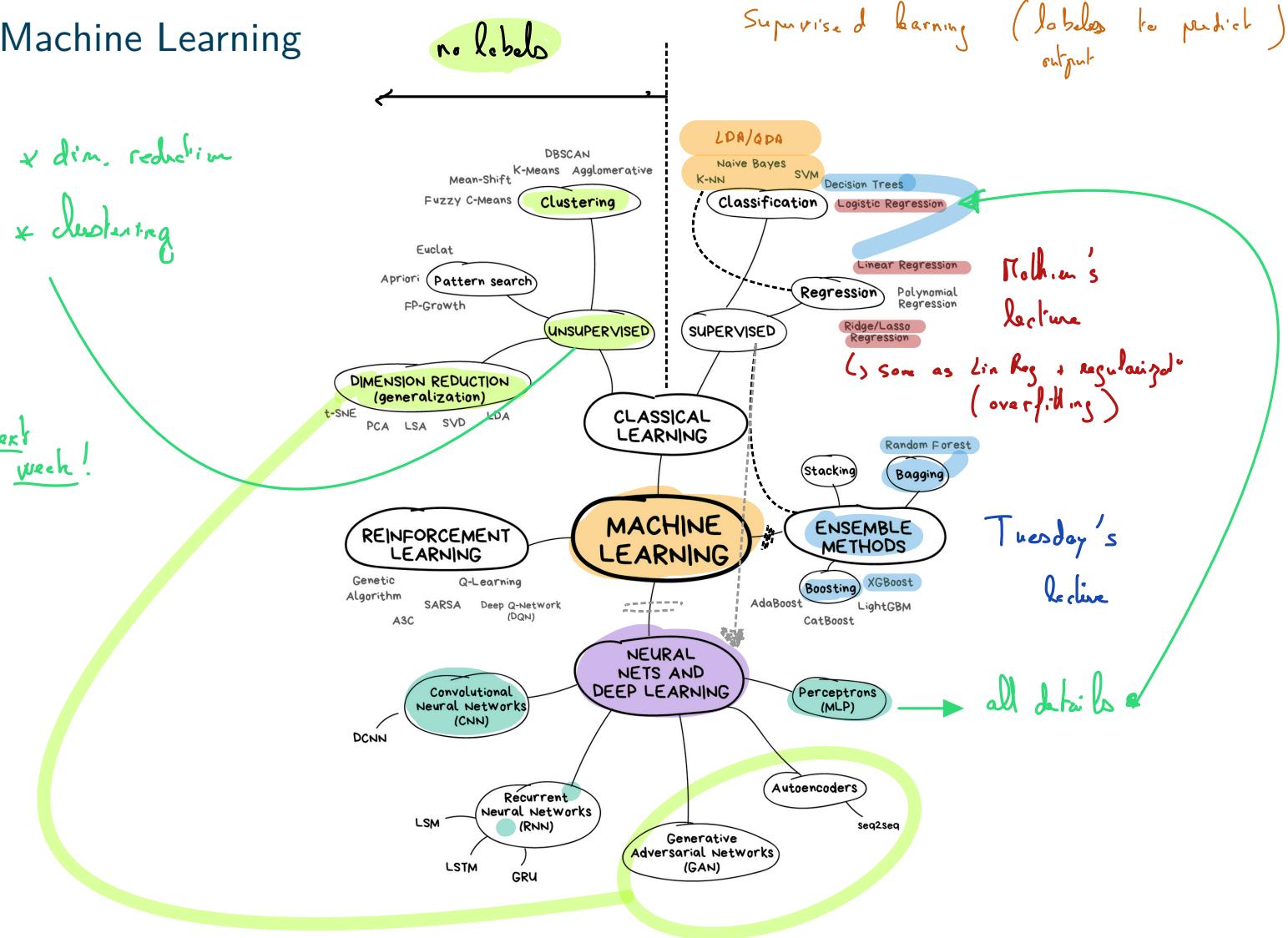
\* dim. reduction

\* clustering

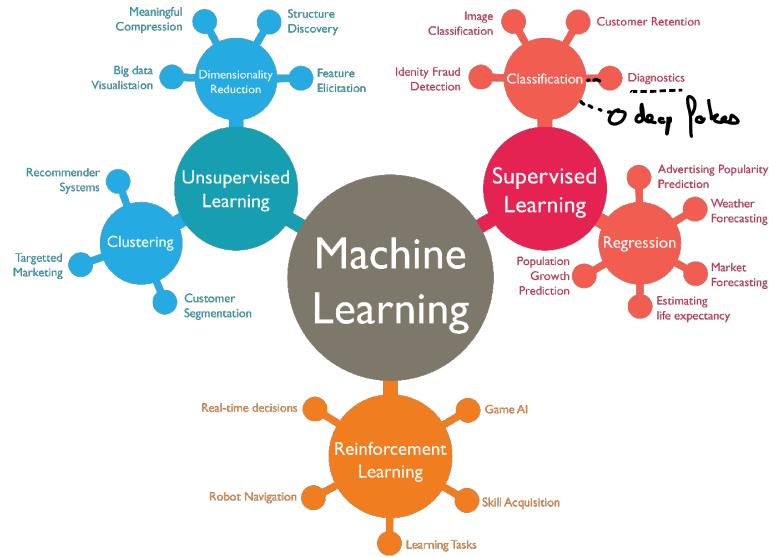
Next week!

no labels

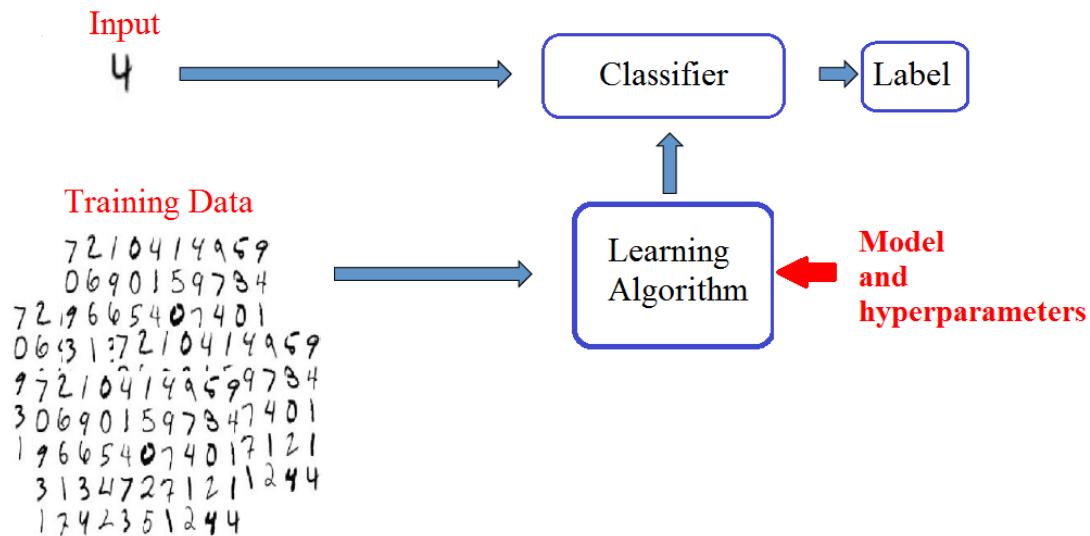
Supervised learning (labels to predict)



# Machine Learning



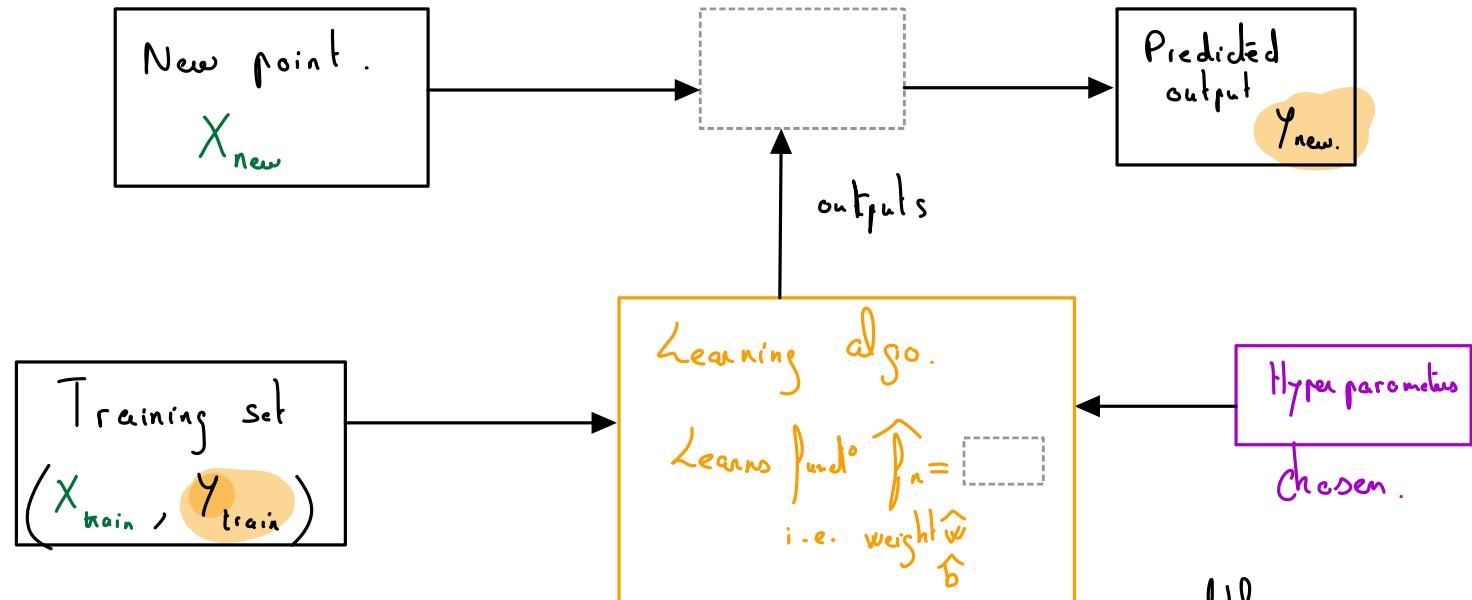
# Summary of previous lectures on Machine Learning



**Training Data:**  $D_n = (X_i, Y_i)_{i=1,\dots,n}, X_i \in \mathbb{R}^d, Y_i \in \{0, \dots, K\}$

**Question:** predict the label of a new one.

# Supervised Learning framework



$X \rightarrow [dashed box]$  predict<sup>10</sup> func<sup>10</sup>

$\hat{P}_n(x)$

Linear regression

Logistic regression

SVM

NN

Decis<sup>10</sup> lines

Random Forests

linear f<sup>10</sup>

"

highly non linear f<sup>10</sup>

DT

set  
up  
[dashed box]

global  
criteria

Max/Likelihood  
Loss  
Doss

Doss

∅  
Random!

Linear regression

Logistic regression

SVM

NN

Decision trees

Random Forests



Linear fit

"

"

highly non linear fit

DT

global  
criteria

Loss / likelihood

Loss

Loss

∅  
local cut!

empirical  
error

Loss 2: logistic loss

$$l(\omega, (x_i, y_i)) = \log(1 + e^{-(x^T \omega)(y_i)})$$

$y_i = 1$   
 $\downarrow$   
 $, \gg 0$

Predic.  
output

$$\boxed{1 \text{ if } x^T \omega \geq 0}$$

Criterium:  $\widehat{R}_n(\omega) = \frac{1}{n} \sum_{i=1}^n l(\omega, (x_i, y_i))$

training set      model to learn  
obs  $n^o_i$

Loss: 0.1 Loss → accuracy  
(,  $\prod (y_i \neq \hat{y}_i)$ ) → impossible to train with !!

improving the model often doesn't change accuracy

Summary :



I predict w. linear fcts

I optimize the loss

Logistic reg :

$$\text{predict} = \langle \mathbf{x}, \mathbf{w} \rangle$$



$$\frac{1}{n} \sum_{i=1}^n \log \left( 1 + e^{-\langle \mathbf{x}_i, \mathbf{w} \rangle} \right)$$

NN:

complex class of fcts:

$$\begin{cases} \mathbf{x} \\ \mathbf{w} \end{cases}$$

non linear.

I optimize the one loss!

Logistic regression:

$$\langle \mathbf{w}, \mathbf{x} \rangle \rightarrow \sigma(\langle \mathbf{w}, \mathbf{x} \rangle)$$

# Outline

1 Goals

2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# From logistic regression to Multi Layer Perceptron 1

Supervised ML problem:

- ① Data  $D_n = (X_i, Y_i)_{i=1,\dots,n}, X_i \in \mathbb{R}^d, Y_i \in \{0, \dots, K\}$
- ② Goal: Predict a the label for a new point.

2 approaches:

## Generative approach

- Define a model on your data (e.g., logit model).
- Estimate parameter (likelihood maximization).

## Discriminative approach

- No statistical model !
- Define a loss function  $\ell$
- Goal:

$$\min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(X), Y)]$$

- Approach:

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)]$$

data is iid.

# From logistic regression to Multi Layer Perceptron 1

Supervised ML problem:

- ① Data  $D_n = (X_i, Y_i)_{i=1,\dots,n}, X_i \in \mathbb{R}^d, Y_i \in \{0, \dots, K\}$
- ② Goal: Predict a the label for a new point.

2 approaches:

## Generative approach

- Define a model on your data (e.g., logit model).
- Estimate parameter (likelihood maximization).

## Discriminative approach

- No statistical model !
- Define a loss function  $\ell$
- Goal:

$$\min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(X), Y)]$$

- Approach:

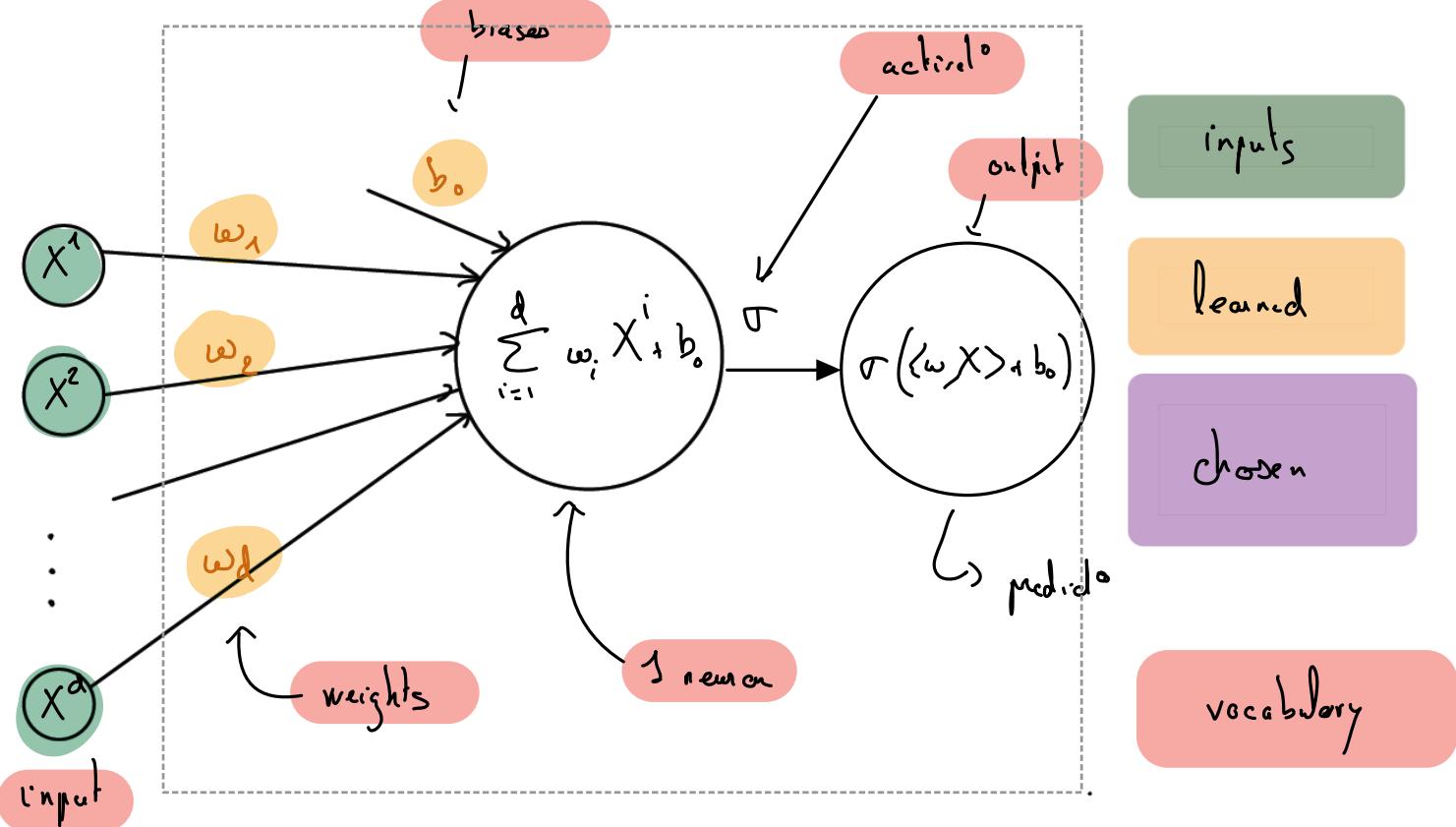
$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)]$$

Logistic regression can be seen both ways !

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-Y_i \cdot w^T X_i))$$

# From logistic regression to Multi Layer Perceptron 3

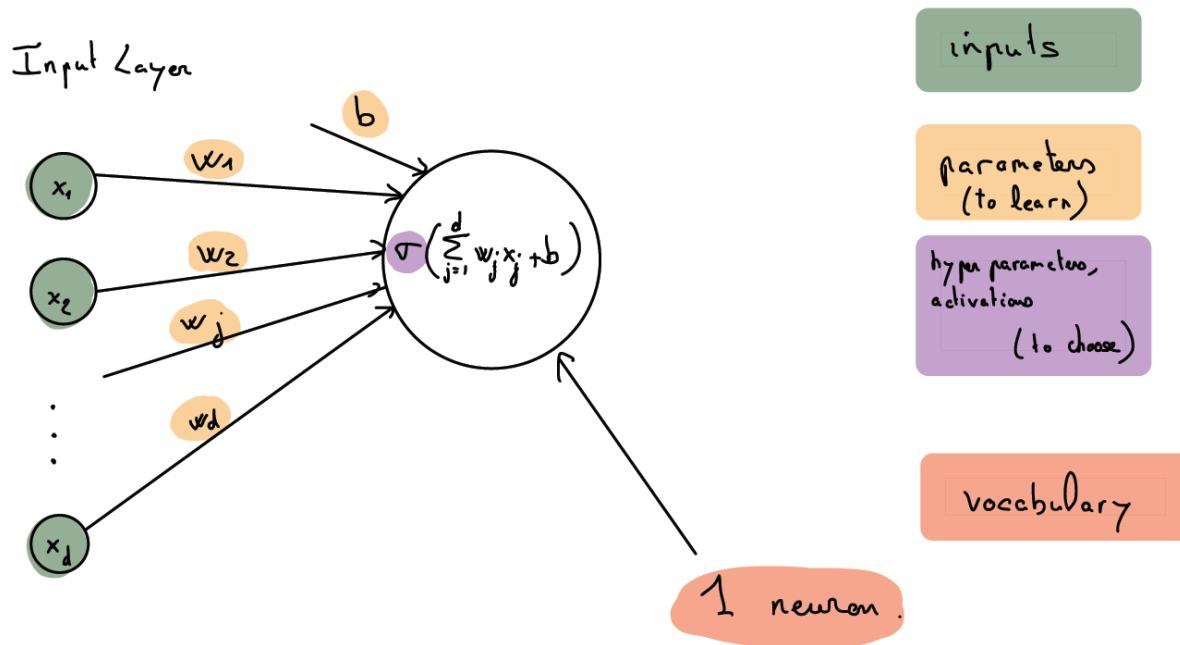
- Class of functions for logistic regression:  $\mathcal{F} = \{X \mapsto w^T X, w \in \mathbb{R}^d\}$ .
- Prediction for a new point is  $1_{\sigma(X_i) > 1/2}$
- Can be seen as predicting a probability  $\sigma(X_i)$  that the output is 1



This is a neural network, with one neuron !

# From logistic regression to Multi Layer Perceptron 3

- Class of functions for logistic regression:  $\mathcal{F} = \{X \mapsto w^T X, w \in \mathbb{R}^d\}$ .
- Prediction for a new point is  $1_{\sigma(X_i) > 1/2}$
- Can be seen as predicting a probability  $\sigma(X_i)$  that the output is 1



This is a neural network, with one neuron !

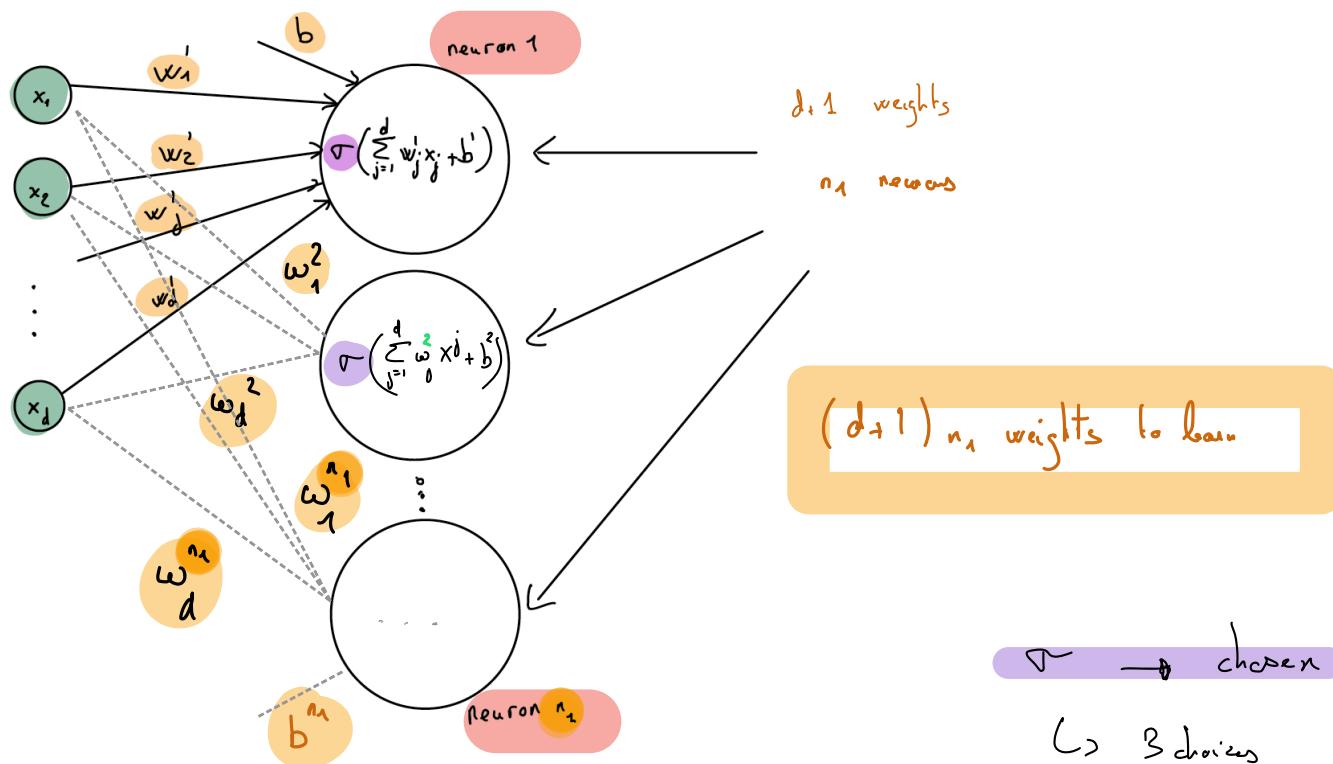
# From logistic regression to Multi Layer Perceptron 4

Extend to

- Multiple Neurons
- Multiple Layers

First Layer

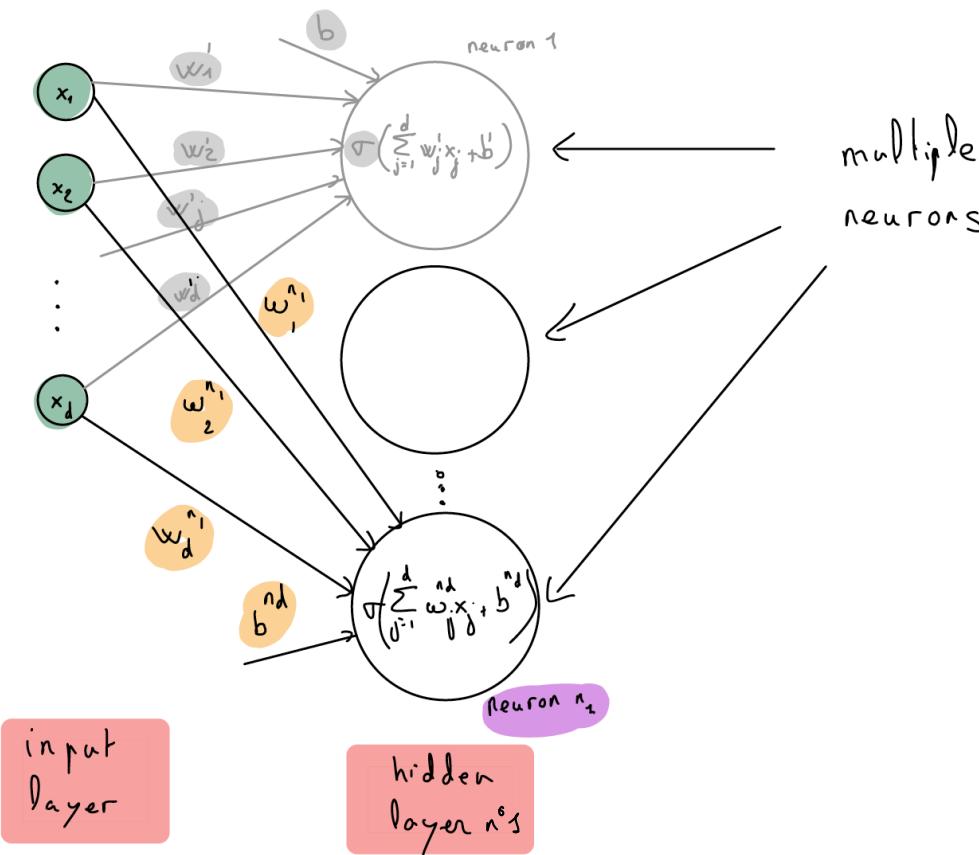
Learned parameters



# From logistic regression to Multi Layer Perceptron 4

Extend to

- Multiple Neurons
- Multiple Layers



input  
layer

hidden  
layer  $n^s$

# From logistic regression to Multi Layer Perceptron 5

Extend to

- Multiple Neurons
- Multiple Layers

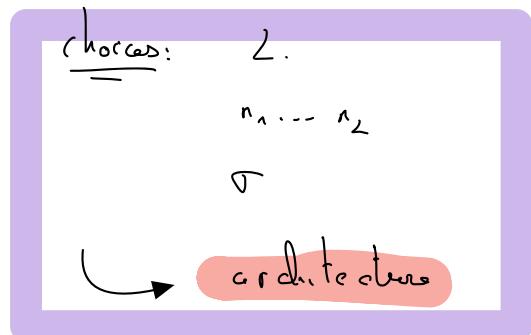
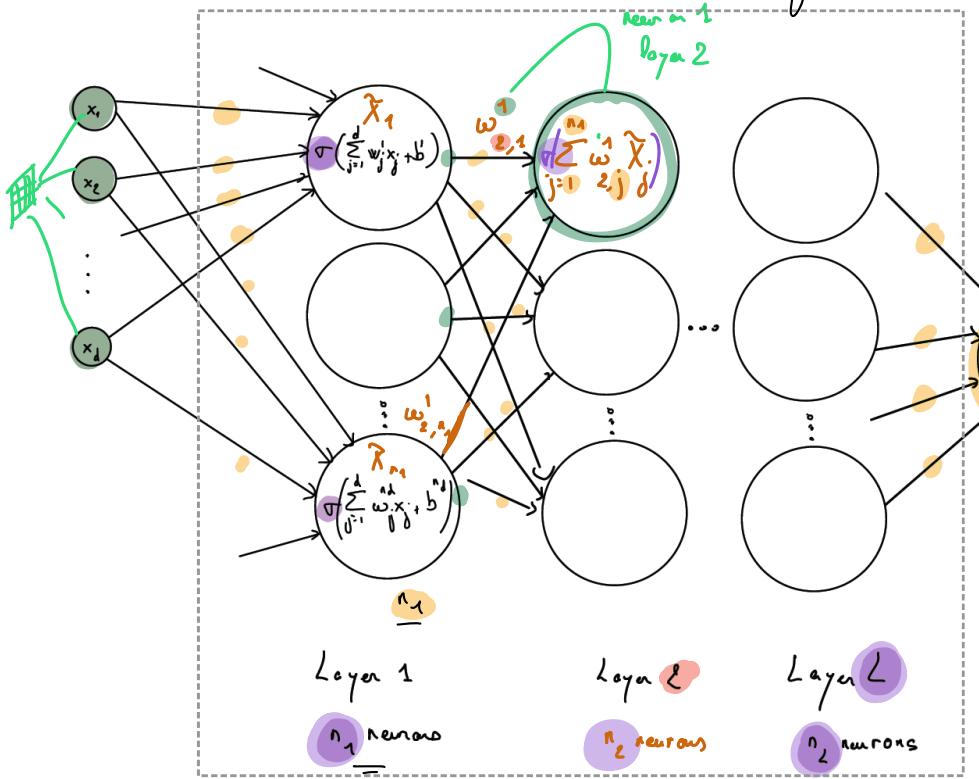


Diagram illustrating the cost function for training a Multi-Layer Perceptron. The cost function is defined as:

$$\text{min } \frac{1}{n} \sum_{i=1}^n \left( P_{\omega}(\vec{x}_i) - y_i \right)^2$$

This represents the mean squared error between the predicted output  $P_{\omega}(\vec{x}_i)$  and the target value  $y_i$ .

Diagram illustrating the parameters of a Multi-Layer Perceptron. The parameters are represented as a vector of weights and biases:

$$P(\omega_1, \dots, \omega_L, n_1, \dots, n_L)$$

# From logistic regression to Multi Layer Perceptron 5

Neural network:

- Goal  $\min_{f \in \mathcal{F}_{MLP}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)]$

- Class of non linear functions.

$$\mathcal{F}_{MLP} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(\dots \sigma(b_1 + W_1 x)))\},$$

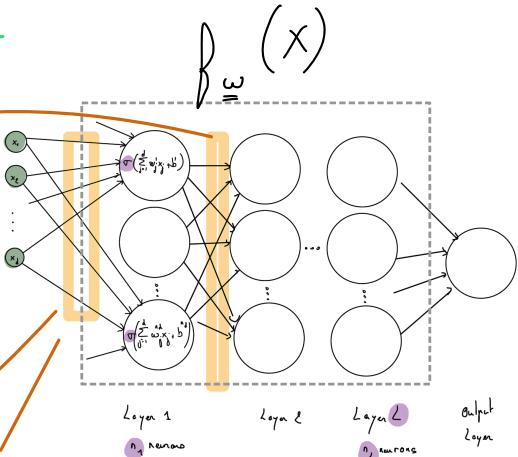
$\curvearrowleft$  set of functions for MLP  
parametrized by  $(w_{\dots}), (b_{\dots})$

$$W \in \mathbb{R}^{n_1 \times d}, b \in \mathbb{R}^{n_1}$$

$$(n_1, d)$$

$$(n_1 \times d)$$

Summary: NN generalize regression by looking for candidates in a much larger class of functions than linear ones.



What needs to be learned

$$l = 1 \dots L$$

$$w_{j,h}$$

$$j = 1 \dots n_{l-1}$$

$$h = 1 \dots n_l$$

$$(b)$$

What needs to be chosen

$$\sigma$$

$$L$$

$$(n_l)_{l=1..L}$$

Vocabulary

- neurons

- activation:  $\sigma$ : creates non linearity

- layers

- input / output

- loss func

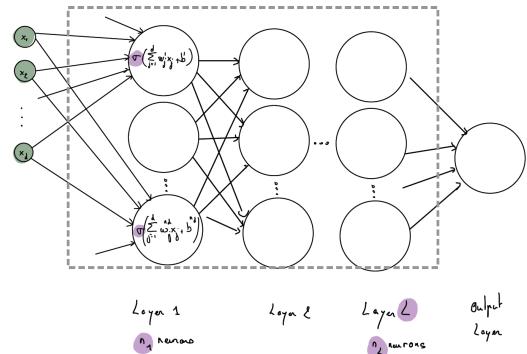
# From logistic regression to Multi Layer Perceptron 5

Neural network:

- Goal  $\min_{f \in \mathcal{F}_{MLP}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)]$

- Class of non linear functions.

$$\mathcal{F}_{MLP} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(\dots(b_1 + W_1 x))), \\ W \in \mathbb{R}^{c \times c}, b \in \mathbb{R}^c\}$$



Summary: NN generalize regression by looking for candidates in a much larger class of functions than linear ones.

What needs to be learned

Parameters  $W, b$

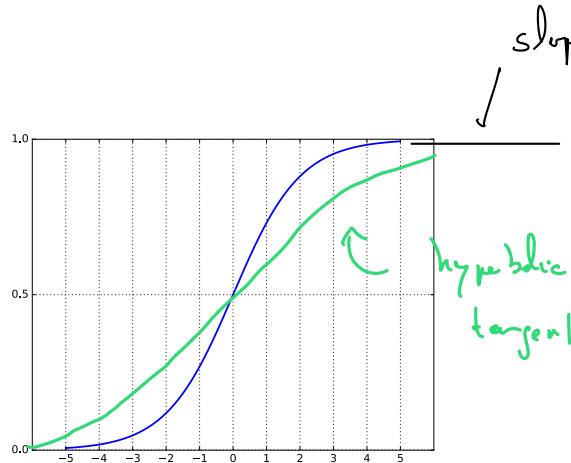
What needs to be chosen

- Activation  $\sigma$
- Number of layers  $L$
- Number of neurons per hidden layer  
 $n_i, i = 1, \dots, L$ .

Vocabulary

- Neuron
- Activation
- Hidden Layer
- Width, Depth
- Fully connected layer

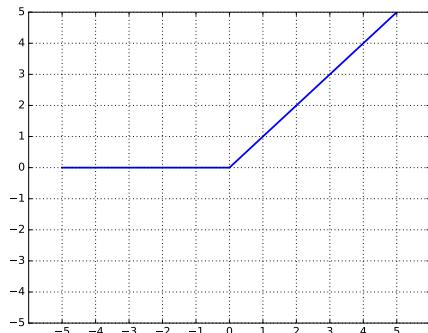
## 2 possible activations: Sigmoid and Rectified Linear Unit



### Sigmoid function

- $x \mapsto \frac{\exp(x)}{1+\exp(x)}$
- Problems:

- ➊ Saturated function: gradient killer -> need for rescaling data



### Rectified Linear Unit (ReLU)

- $x \mapsto \max(0, x)$
- Pros & cons:
  - ➊ Not a saturated function. Kills negative values.
  - ➋ Empirically, convergence is faster than sigmoid/tanh.
  - ➌ Plus: biologically plausible

# How to deal with multi-class output: softmax activation

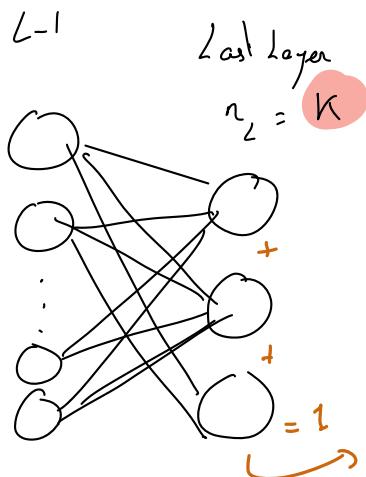
*← particular activate  
for last layer in MCC*

When we do multi-class classification, i.e.,  $Y_i \in 1, \dots, K$ , we output  $K$  different values:

- Each of them corresponds to the probability of belonging to the class  $j$ ,  $1 \leq j \leq K$
- To obtain probabilities (adding up to 1):

- ① We have  $K$  neurons on the last layer
- ② And use a Softmax activation to renormalize.

Softmax output unit, used to predict  $\{1, \dots, K\}$ :



$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$

*comparison*

$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \rightarrow \text{Vers.}$   
 $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow \text{Virg}$   
 $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \rightarrow \text{Sepicolo}$

softmax activation

$$\begin{pmatrix} -10 \\ 10 \\ 20 \end{pmatrix} \xrightarrow{\text{soft}} \begin{pmatrix} \frac{e^{-10}}{a} \\ \frac{e^{10}}{a} \\ \frac{e^{20}}{a} \end{pmatrix}$$

$a = e^{-10} + e^{10} + e^{20}$

Up to that point, we have seen :

- What a Neural Network was
- Why it is expected to learn better.

Next Question: how to implement it ?

# Python - Keras

## 1. Hardware:

- CPU
- GPU 
- TPU

## 2. Software (Python packages):

- Pytorch/Tensorflow
- → **Keras** : TF high level API. Ideal for applications.



## Keras - A few lines !<sup>1</sup>

## What do we need to specify?

- \* do we need to specify?
    - \* data
    - \* class: NN. →
      - ↳ (neural)
      - activations per layer
    - \* off algo

## What do we need to do next?

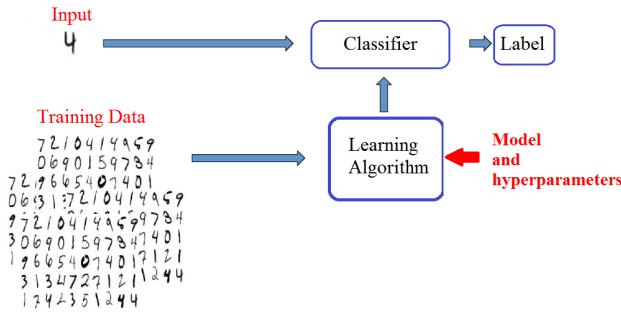
- \* train . fit
  - \* predict on the test  
repeat last error

## 1 Notebook Python

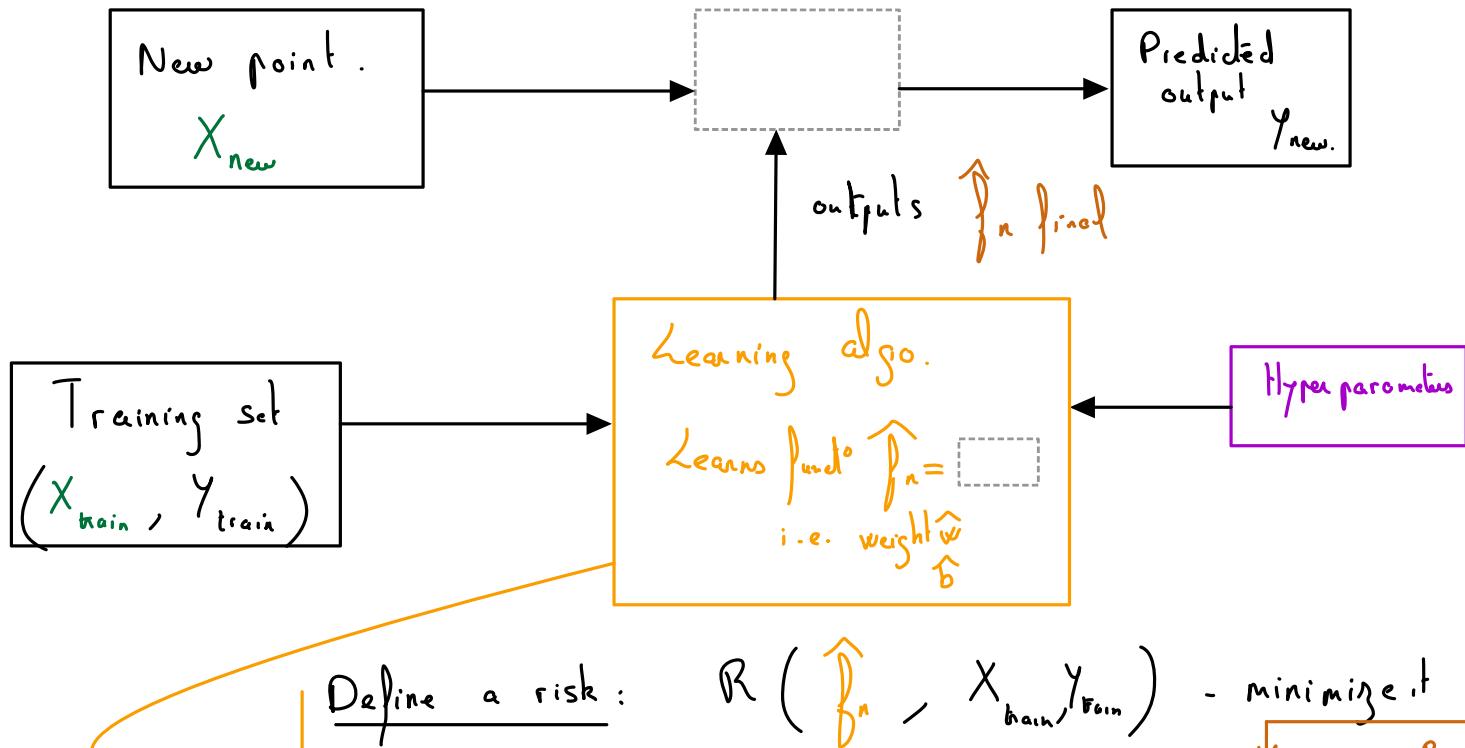
## Keras - A few lines !<sup>1</sup>

## What do we need to specify?

## What do we need to do next?



## 1 Notebook Python

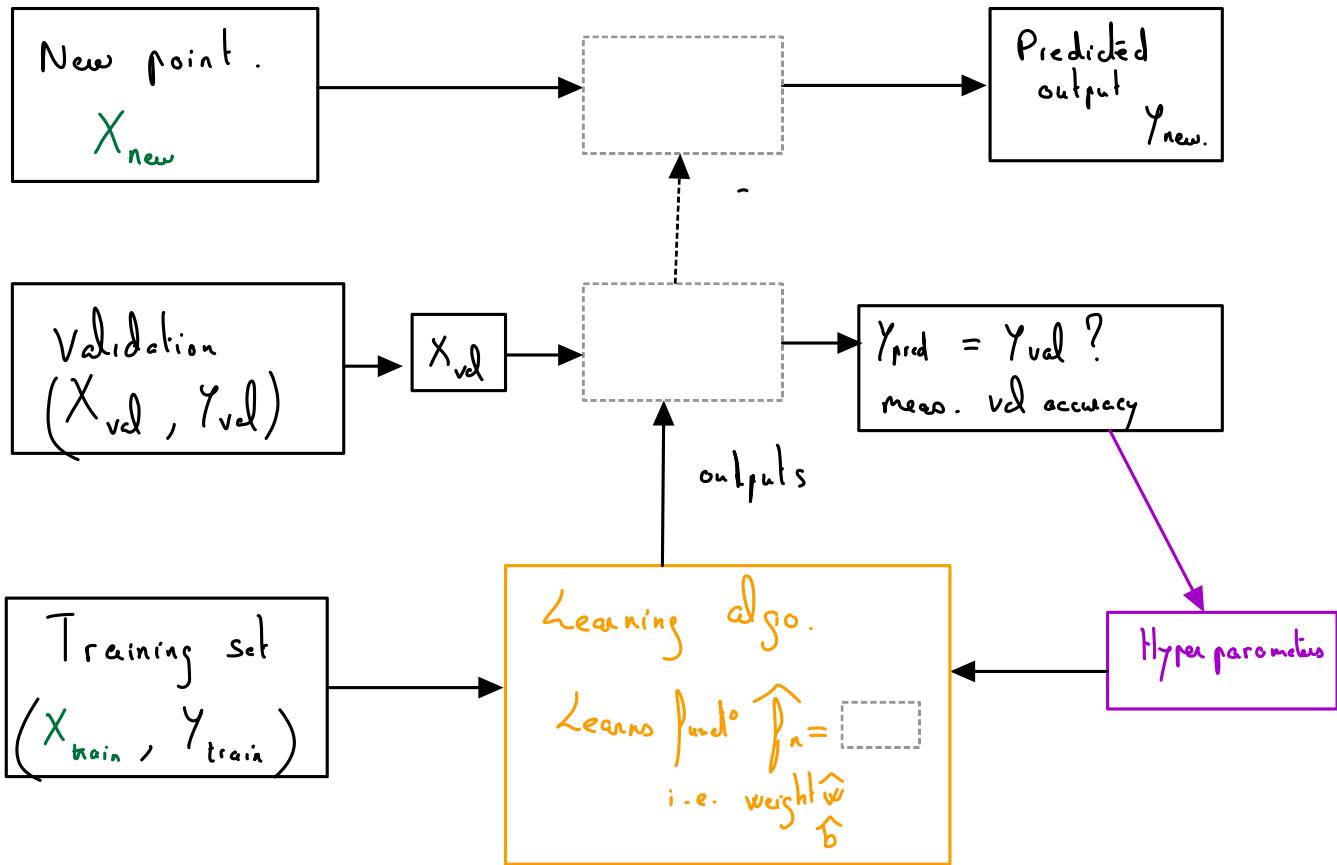


Learning: by GD.

Define a risk:  $R(\hat{f}_n, X_{train}, Y_{train})$  - minimize it

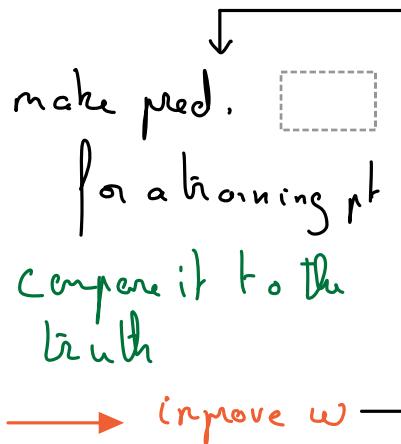
Iteratively improve  $\hat{f}_n$  within the class of allowed funcs. :

take a training point  $X^p$   
 compute  $\hat{f}_n(X^p) = \hat{Y}^p$   
 $X^p \rightarrow [ ] \rightarrow \hat{Y}^p$   
 compare to  $Y^p$   
 improve the weights  $\times 10^4$



# Learning Process

init choose random  $w$ :  
each weight is different



Some Learning process

# Logistic regression

D L.

make pred.   
for a training pt  
compare it to the  
truth  
→ (improve  $w$ )

} gradient descent

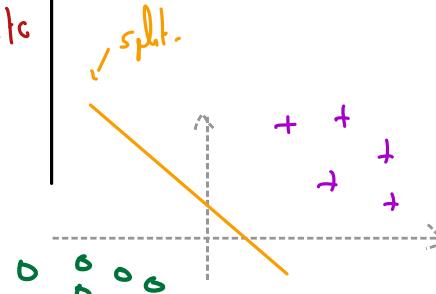
& weights (11)

much and more  
weights

MLP.



Different  
Weights  
and predict  
boxed



not linear.  
Depend on?

# Intuition

Up to that point, we have seen :

- What a Neural Network was.
- How to implement it in Python.

Next Question: why and when does it work ?

# Intuition

Up to that point, we have seen :

- What a Neural Network was.
- How to implement it in Python.

Next Question: why and when does it work ?

- ① Approximation theorem
- ② Optimization

# Approximation Theorem

## Goal

$$\min_{f \in \mathcal{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)].$$

As said before, a MLP can output non-linear functions... But how powerful is it ?

# Approximation Theorem

## Goal

$$\min_{f \in \mathcal{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)].$$

As said before, a MLP can output non-linear functions... But how powerful is it ?

Continuous Neural Networks with one single hidden layer and any bounded and non constant activation function can approximate any function in  $L^p$ , provided a sufficient number of hidden units.

- ⇒ Very powerful in terms of approximations.
- Not very surprising: non linear function with millions of parameters !

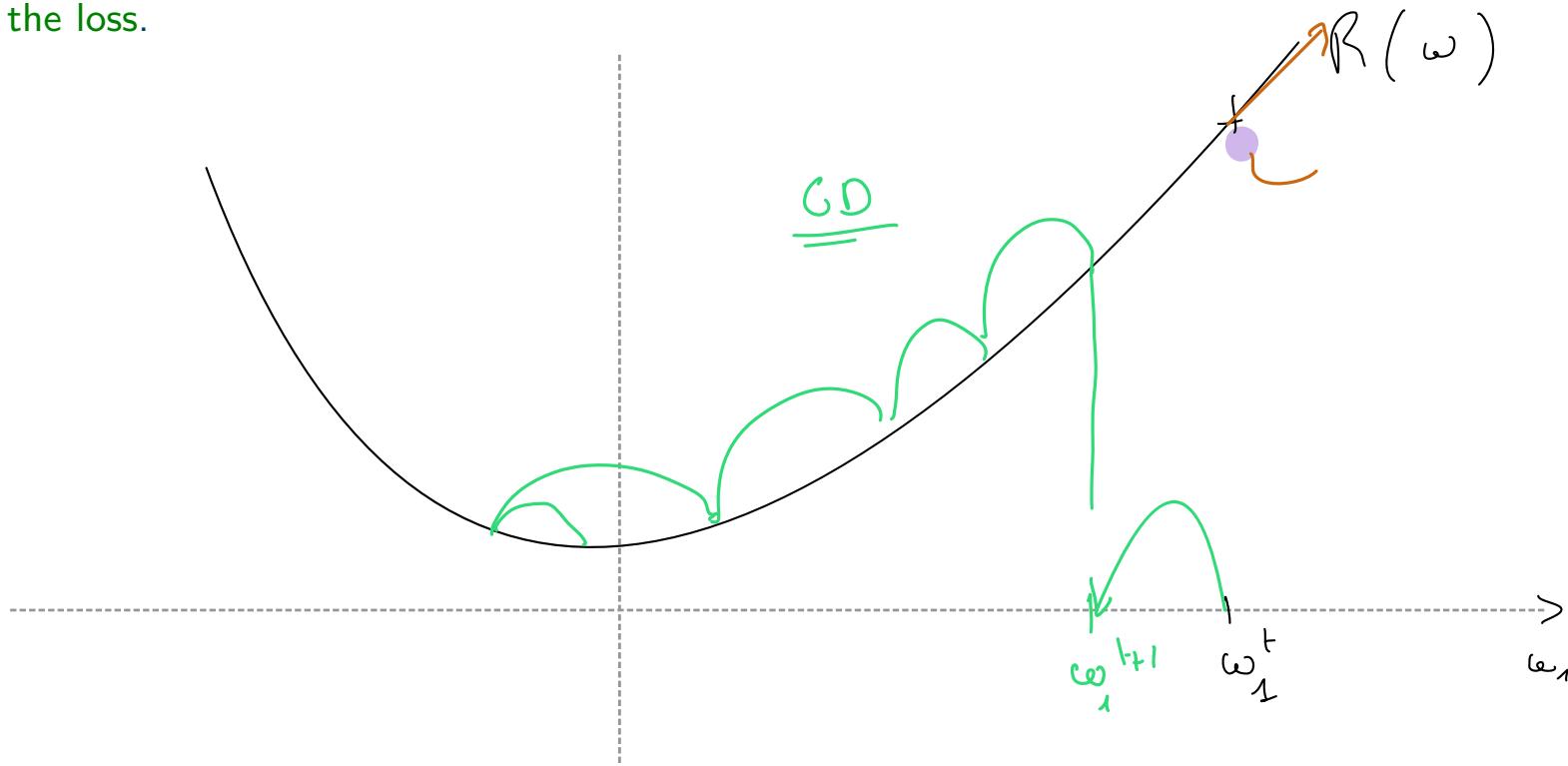
# Optimization

Goal

$$\min_{f \in \mathcal{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)].$$

How to minimize a function?

Gradient Methods. = cheapest way to update (learn) the weights iteratively to minimize the loss.



# Optimization

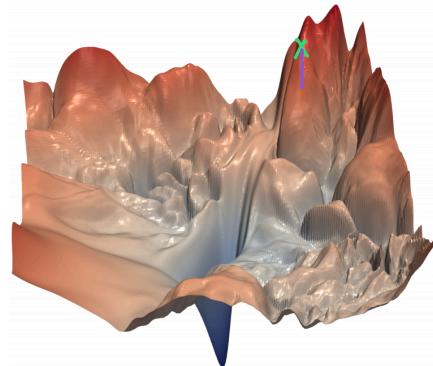
Goal

$$\min_{f \in \mathcal{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^n [\ell(f(X_i), Y_i)].$$

Gradient Methods. = cheapest way to update (learn) the weights iteratively to minimize the loss.

Two “miracles”:

- ① Autodifferentiation: even though the function is very complex and high dimensional, it is possible to compute gradients !
- ② Optimization seems to work ok, though the function is non convex - we do not end up in too bad local minima. (specialized optim algos: Adam, AdaDelta, etc.)



x dimension helps

x miracle happens.

These 2 miracles make it possible to learn a good regressor/classifier with NN and to benefit from the powerful approximation properties

# Conclusion

Neural networks :

- ① learn very complex non linear functions in high dimension
- ② can approximate nearly any function
- ③ can be optimized even though highly non convex.

Are thus expected to work:

- When depth or width increases.
- Thus with very large datasets
- Requiring a lot of computational power.

⇒ **Explains why they were so successful since 2010**

# Conclusion

Neural networks :

- ① learn very complex non linear functions in high dimension
- ② can approximate nearly any function
- ③ can be optimized even though highly non convex.

Are thus expected to work:

- When depth or width increases.
- Thus with very large datasets
- Requiring a lot of computational power.

⇒ **Explains why they were so successful since 2010**

*What my layer wants me to say:* we haven't talked about many points !!

- Initialization
- Regularization

# Conclusion

Neural networks :

- ① learn very complex non linear functions in high dimension
- ② can approximate nearly any function
- ③ can be optimized even though highly non convex.

Are thus expected to work:

- When depth or width increases.
- Thus with very large datasets
- Requiring a lot of computational power.

⇒ **Explains why they were so successful since 2010**

*What my layer wants me to say:* we haven't talked about many points !!

- Initialization
- Regularization

Next:

- Brief history
- How to use the structure ?

# A brief history of NN

- 1943: Neural networks
- 1957: Perceptron
- 1974-86: Backpropagation, RBM, RNN
- 1989-98: CNN, MNIST
- 2009: ImageNet
- 2012: AlexNet,
- 2014: GANss
- 2016: AlphaGo
- 2018: BERT

The Computational power made the major change (+ investment and creativity).

# Directions

- When does it work ?

**Large or huge datasets. Structured tasks.**

↳ **Images and text.**

⚠ do not try to apply DL everywhere

Each application requires a special architecture:

- Images : **Convolutional Neural Network (CNN).**
- Text : **Recurrent Neural Networks (RNN).**



# Outline

1 Goals

2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# CNN

We consider an **image classification task**: we want to recognize which object is on an image.

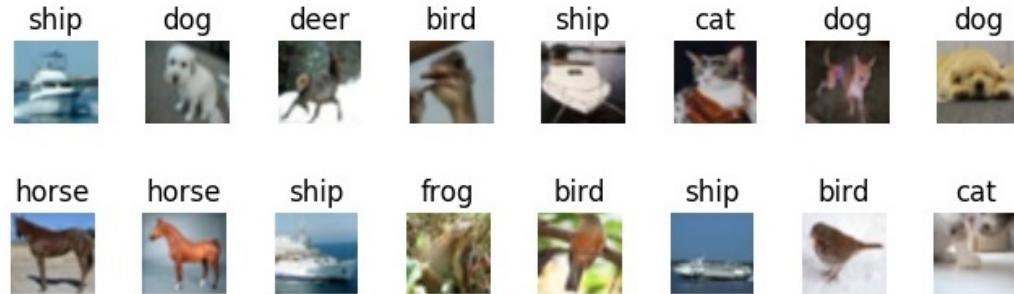


Figure: CIFAR dataset

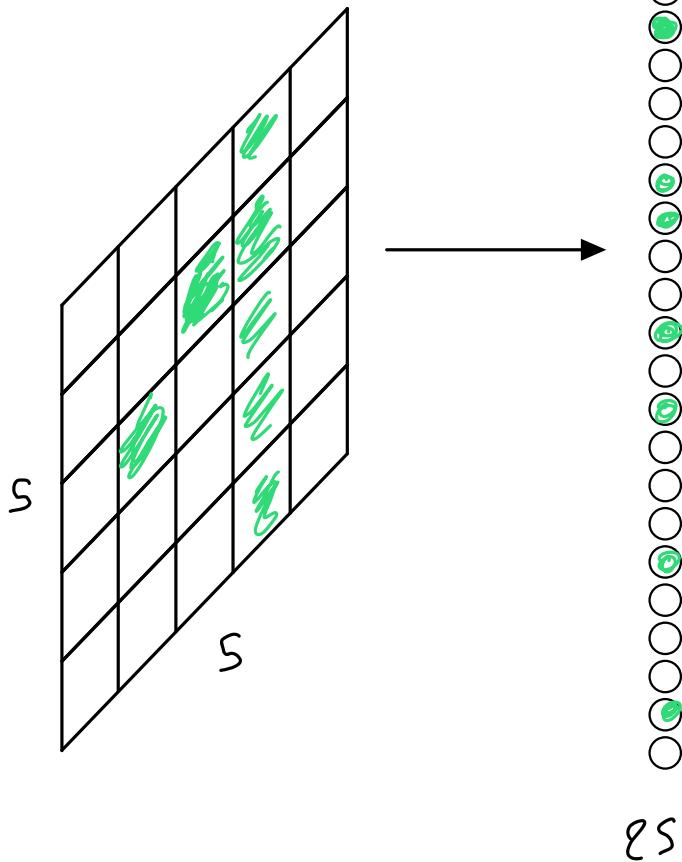
The input is an image: about  $10^3$  to  $10^7$  pixels: these are **our inputs**.

**Problems:** The multi-layer perceptron:

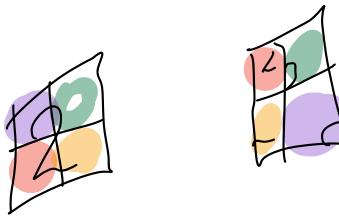
- Does not take into account local information. It would work similarly on any permutation of the pixels !
- Has, just on the first layer, as many parameters as there are inputs !

## Convolutional Layer

## Pb of flattern layer

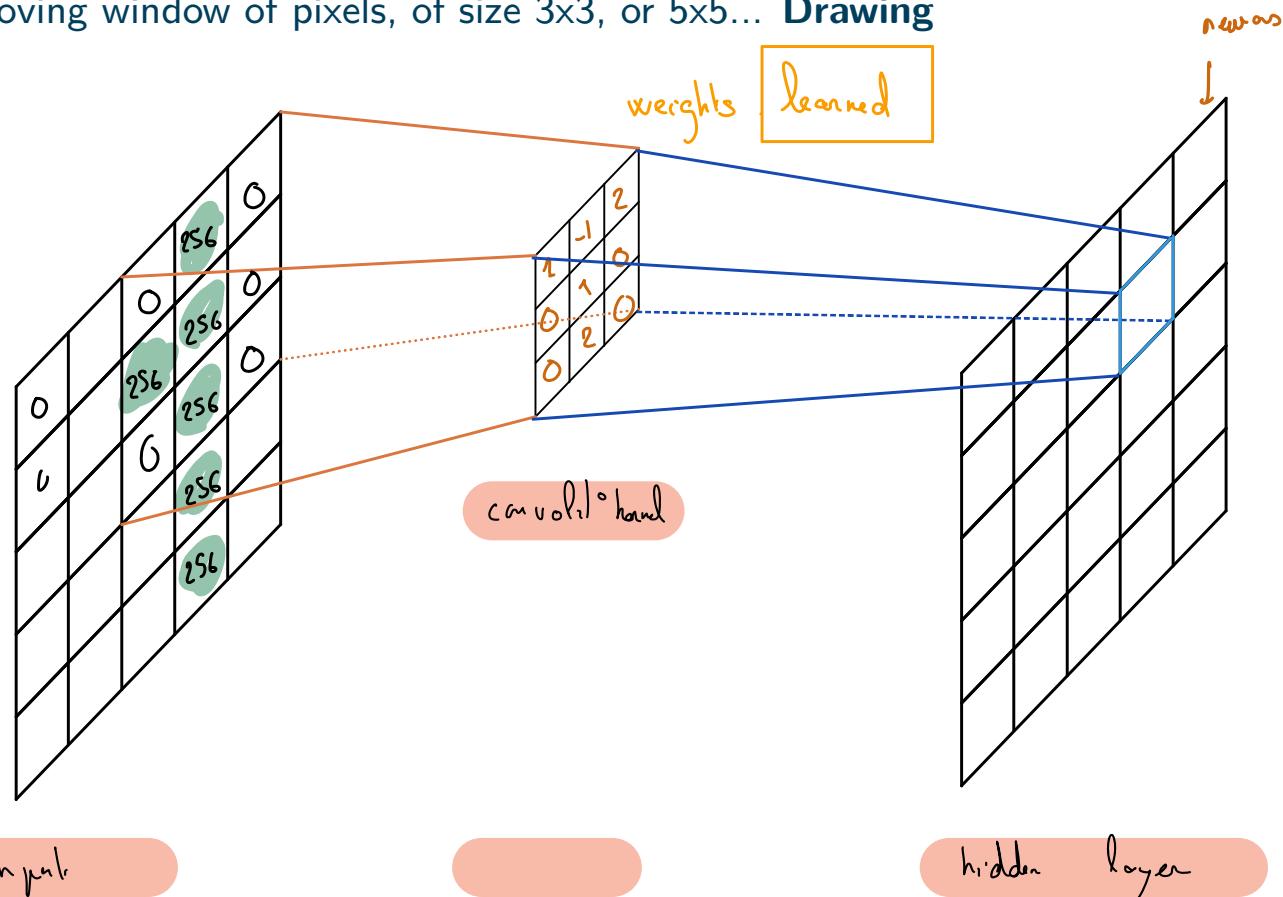


a fully connected NN  
would be able to predict  
just as well if it got as  
input  
 $x^1$  sep1-l  
 $x^2$  sep1-w  
 $x^3$  petal length  
 $x^4$  petal - w  
a permuted  
of the inputs



# Convolutional Layer

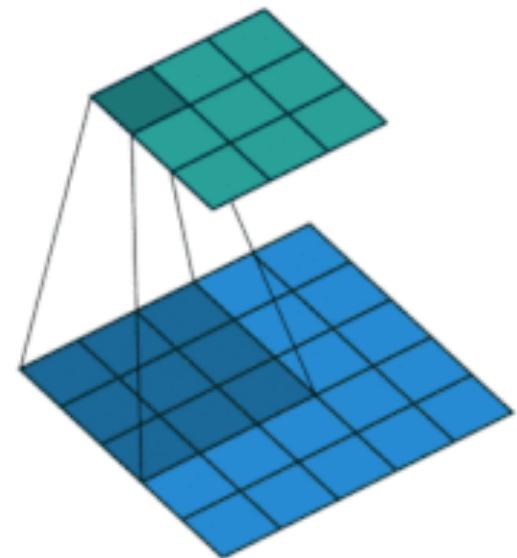
Instead of making a linear combination of all the pixels, we consider a weighted average of a moving window of pixels, of size 3x3, or 5x5... **Drawing**



# Convolutional Layer

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

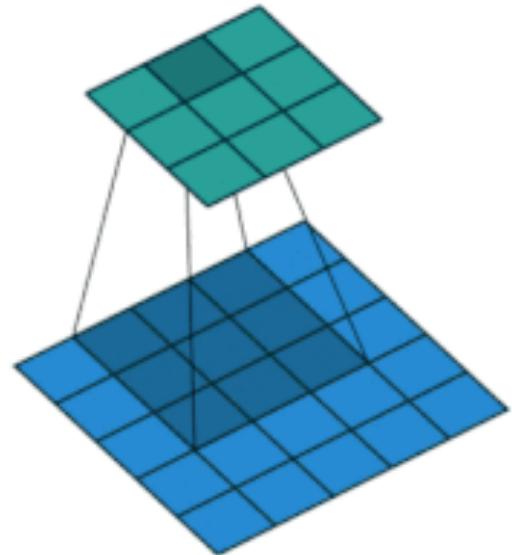
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	$3_0$	$2_1$	$1_2$	0
0	$0_2$	$1_2$	$3_0$	1
3	$1_0$	$2_1$	$2_2$	3
2	0	0	2	2
2	0	0	0	1

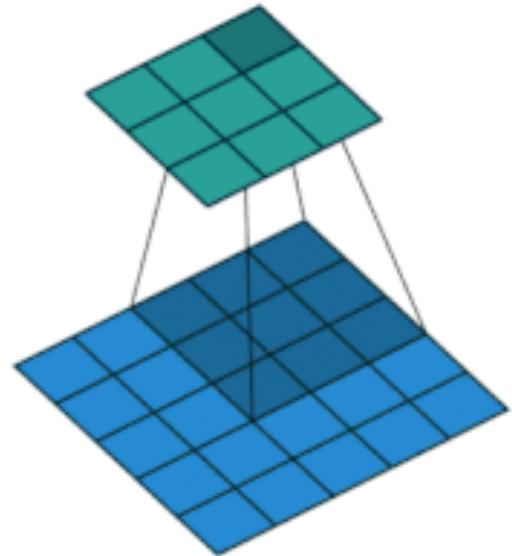
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2 <sub>0</sub>	1 <sub>1</sub>	0 <sub>2</sub>
0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>0</sub>
3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>
2	0	0	2	2
2	0	0	0	1

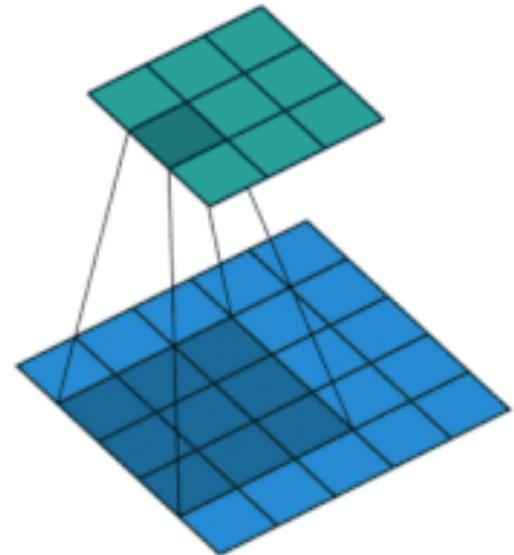
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2	1	0
0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>2</sub>	3	1
3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>0</sub>	2	3
2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	2	2
2	0	0	0	1

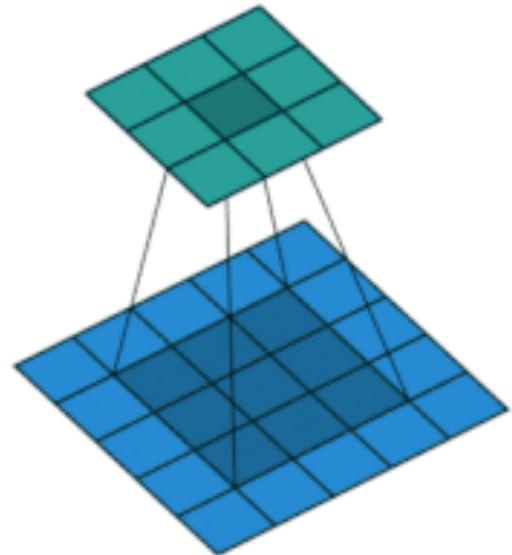
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2	1	0
0	0 <sub>0</sub>	1 <sub>1</sub>	3 <sub>2</sub>	1
3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>	3
2	0 <sub>0</sub>	0 <sub>1</sub>	2 <sub>2</sub>	2
2	0	0	0	1

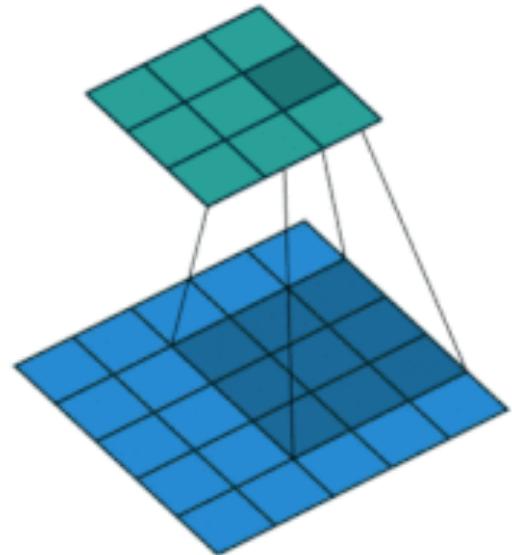
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2	1	0
0	0	$1_0$	$3_1$	$1_2$
3	1	$2_2$	$2_2$	$3_0$
2	0	$0_0$	$2_1$	$2_2$
2	0	0	0	1

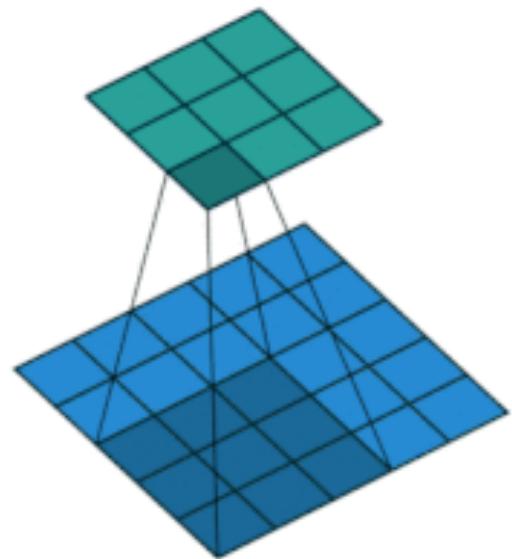
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2	1	0
0	0	1	3	1
$3_0$	$1_1$	$2_2$	2	3
$2_2$	$0_2$	$0_0$	2	2
$2_0$	$0_1$	$0_2$	0	1

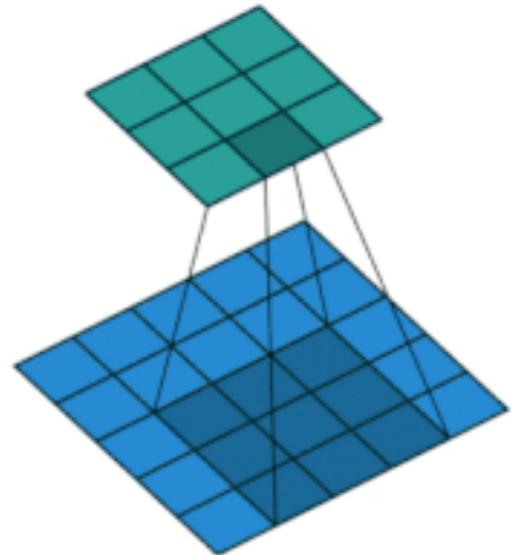
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2	1	0
0	0	1	3	1
3	$1_0$	$2_1$	$2_2$	3
2	$0_2$	$0_2$	$2_0$	2
2	$0_0$	$0_1$	$0_2$	1

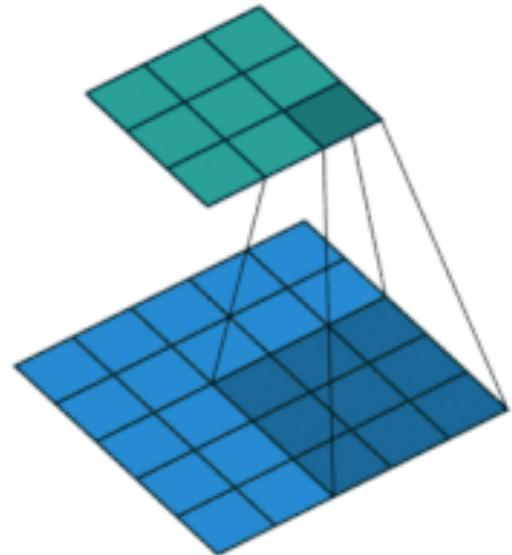
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

3	3	2	1	0
0	0	1	3	1
3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>
2	0	0 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>
2	0	0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>2</sub>

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



# Convolutional Layer

# Convolutional Layer

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

+

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

$$164 + 1 = -25$$

Bias = 1

-25					...
					...
					...
					...
...	...	...	...	...	...

# Convolutional Layer

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...	...	...	...	...	...	...	...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



310

+

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...	...	...	...	...	...	...	...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-170

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...	...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



325

+

+ 1 = 466



-25	466			...
				...
				...
				...
...	...	...	...	...

# Convolutional Layer

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...	...	...	...	...	...	...	...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



314

+

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...	...	...	...	...	...	...	...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-175

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...	...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



326

+

+ 1 = 466

Bias = 1

Output

-25	466	466	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

# Convolutional Layer

0	0	0	0	0	0	0	...
0	156	155	156	158	158	...	
0	153	154	157	159	159	...	
0	149	151	155	158	159	...	
0	146	146	149	153	158	...	
0	145	143	143	148	158	...	
...	...	...	...	...	...	...	...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



318

+

0	0	0	0	0	0	0	...
0	167	166	167	169	169	...	
0	164	165	168	170	170	...	
0	160	162	166	169	170	...	
0	156	156	159	163	168	...	
0	155	153	153	158	168	...	
...	...	...	...	...	...	...	...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-173

0	0	0	0	0	0	0	...
0	163	162	163	165	165	...	
0	160	161	164	166	166	...	
0	156	158	162	165	166	...	
0	155	155	158	162	167	...	
0	154	152	152	157	167	...	
...	...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



329

+

$$329 + 1 = 475$$

Bias = 1  
↑

-25	466	466	475	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

Output

# Convolutional Layer

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



298

+

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-491

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



487

+

+ 1 = 295

Bias = 1

-25	466	466	475	...
295				...
				...
				...
...	...	...	...	...

# Convolutional Layer

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



148

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-8

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



646

+

+ 1 = 787

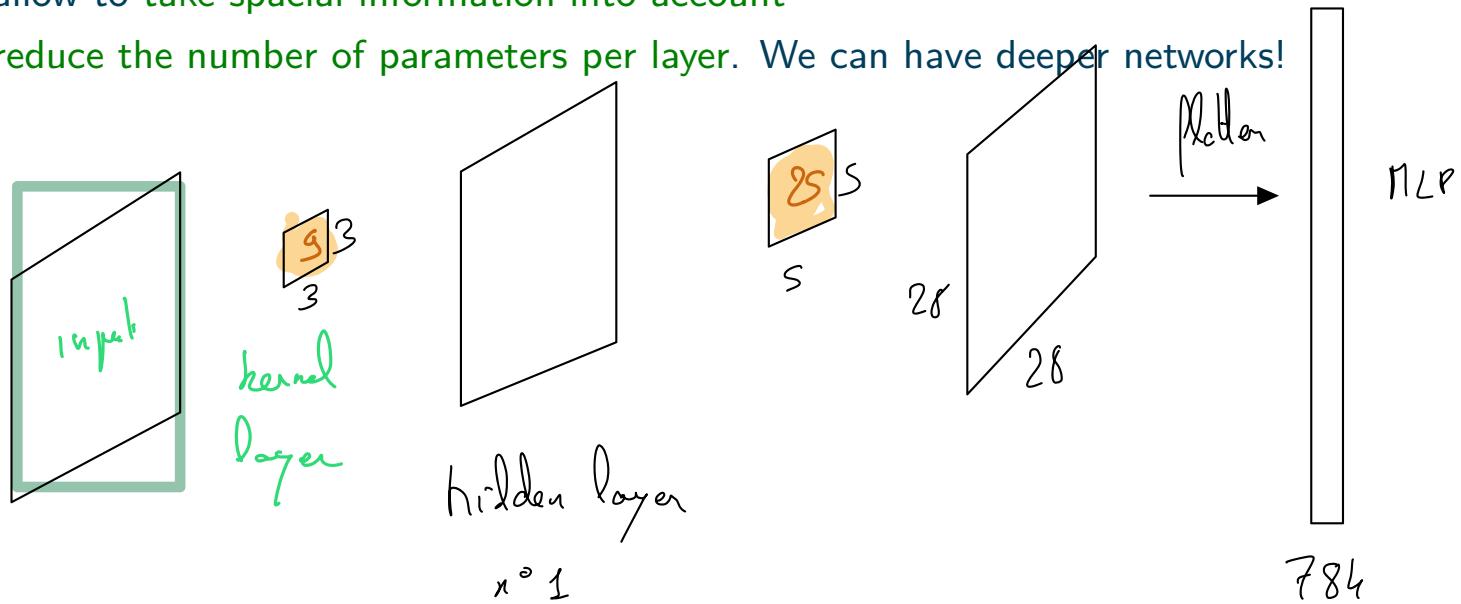
Bias = 1

-25	466	466	475	...
295	787			...
				...
				...
...	...	...	...	...

# Summary

CNN:

- allow to take spacial information into account
- reduce the number of parameters per layer. We can have deeper networks!



# Summary

## CNN:

- allow to take spacial information into account
- reduce the number of parameters per layer. We can have deeper networks!

## LeNet 1998:

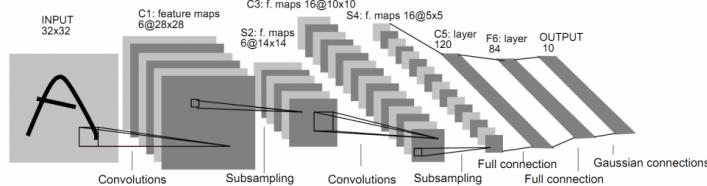
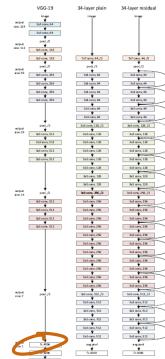


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

8 layers, 60k parameters.

## Resnet 2016



~ 40 Layers, 140M parameters

# Outline

1 Goals

2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# Outline

1 Goals

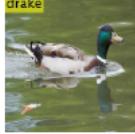
2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# Image classification - CNN Tree

Category	Confusion Set							
tench								
indigo bunting								
red-breasted merganser								
echidna								
shopping basket								

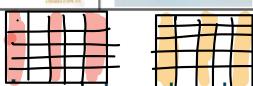


Figure: Confusion set outputs by AlexNet softmax prediction on validation set of ILSVRC 2015.

# Image classification - CNN Tree

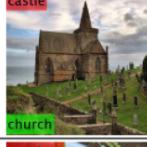
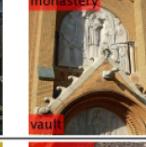
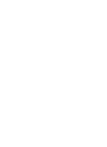
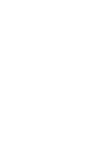
Category	Example Validation Images					
barracouta	 coho  barracouta	 coho  barracouta	 tench  barracouta	 hammerhead  tiger shark	 great white shark  great white shark	 coho  barracouta
church	 castle  church	 beacon  castle	 altar  church	 monastery  vault	 castle  castle	 altar  vault
spaghetti squash	 spaghetti squash  spaghetti squash	 lemon  spaghetti squash	 lemon  spaghetti squash	 ice cream  spaghetti squash	 acorn squash  butternut squash	 spaghetti squash  spaghetti squash
espresso	 cup  espresso	 soup bowl  coffee mug	 soup bowl  plate	 plate  espresso	 bakery  bakery	 coffeepot  espresso maker
trolleybus	 school bus  trolleybus	 police van  trolleybus	 streetcar  trolleybus	 school bus  passenger car	 school bus  passenger car	 streetcar  trolleybus

Figure: Top label is given by basic AlexNet CNN while bottom one is given by CNNTree (green color corresponds to a correct prediction)

# Outline

1 Goals

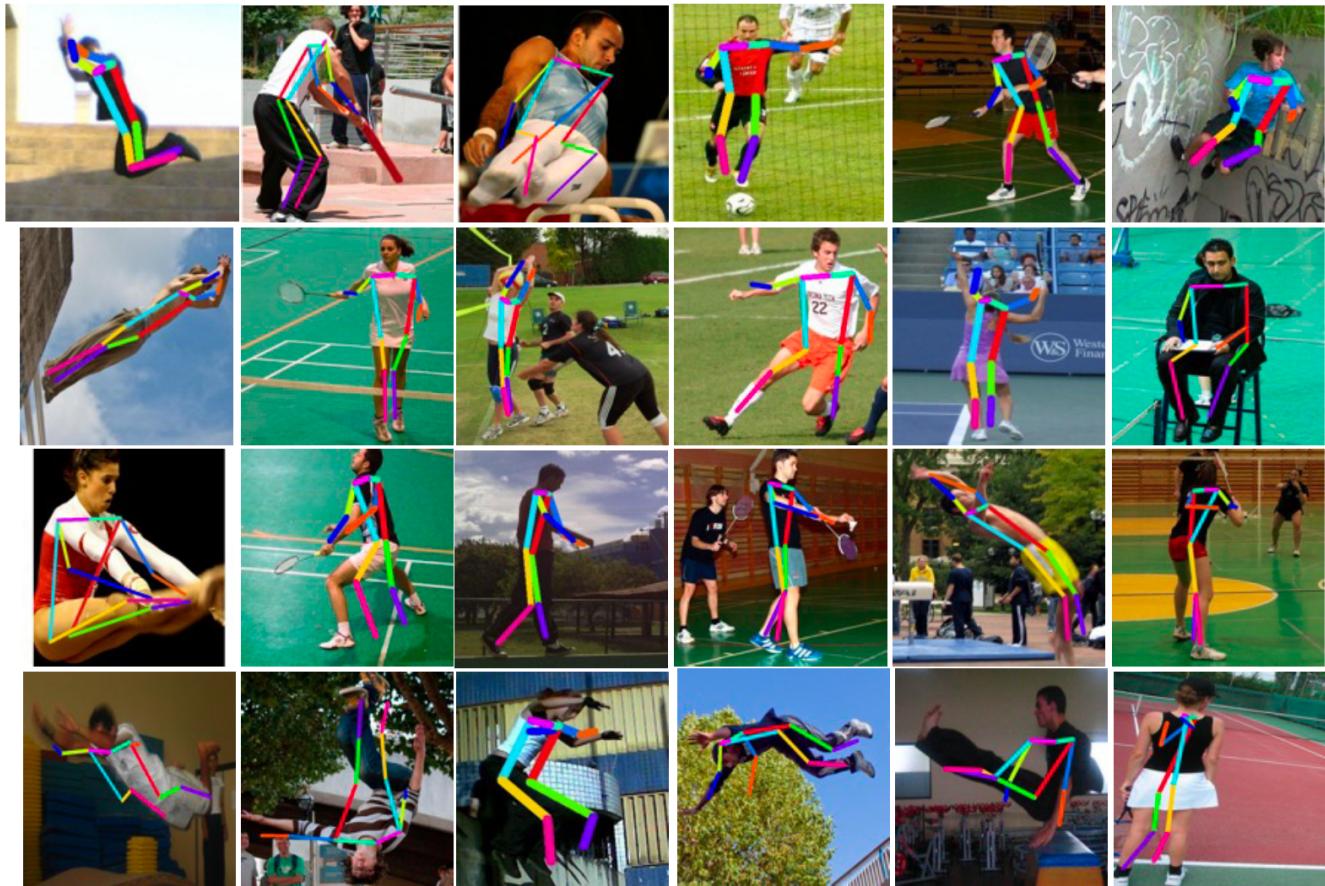
2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

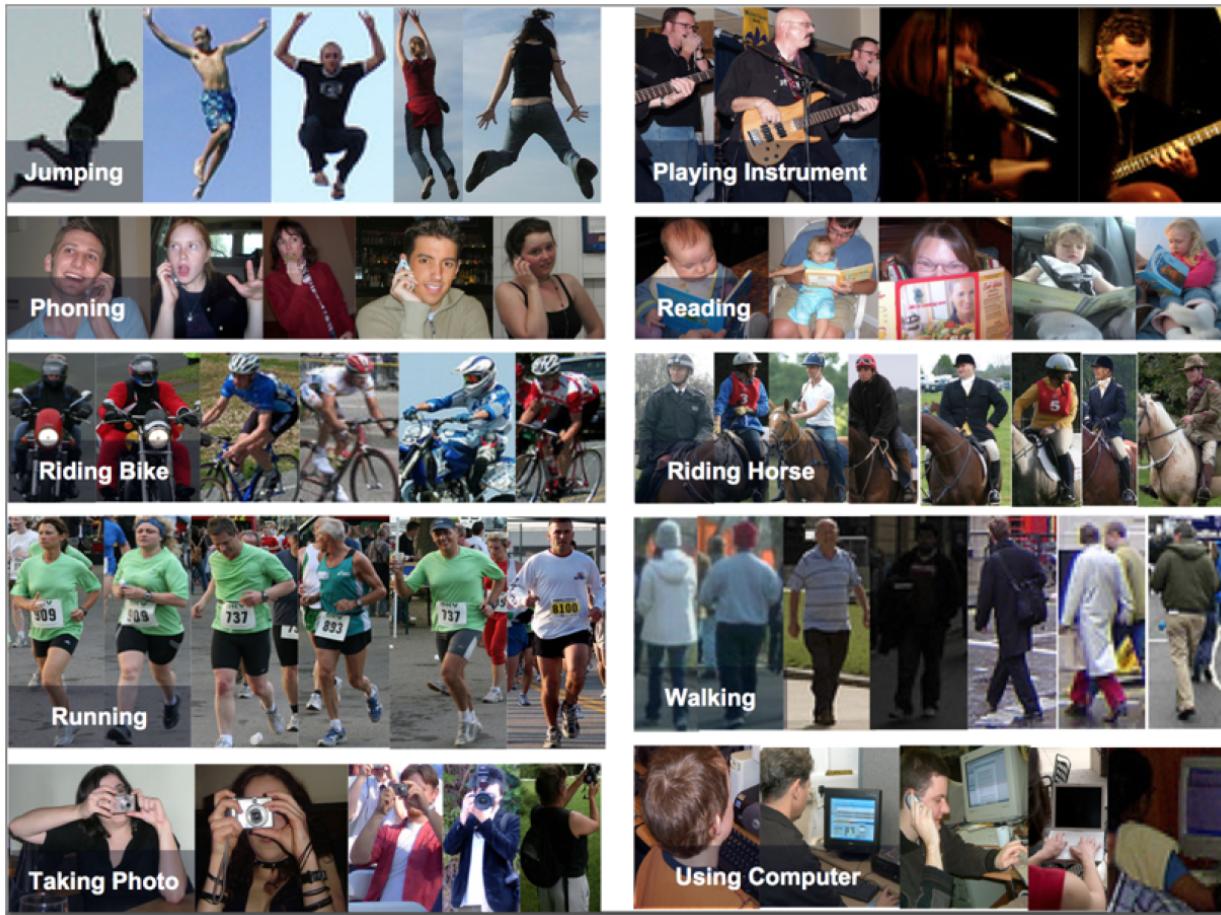
4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

# Pose estimation - Deeppose



# Action recognition



# Outline

1 Goals

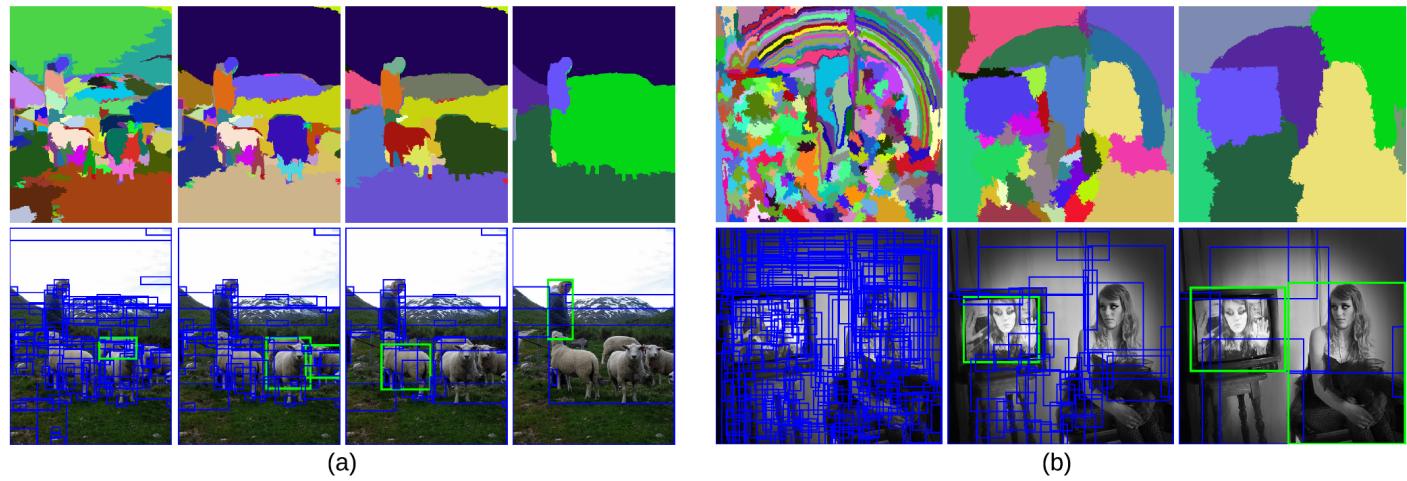
2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation**
- Scene labeling - Semantic segmentation
- Object tracking - videos

# Object detection and segmentation



# Object detection - YOLO, SDD

More recently, YOLO (You Only Look Once) and SSD (Single Shot Detector) allow single pipeline detection that directly predicts class labels.

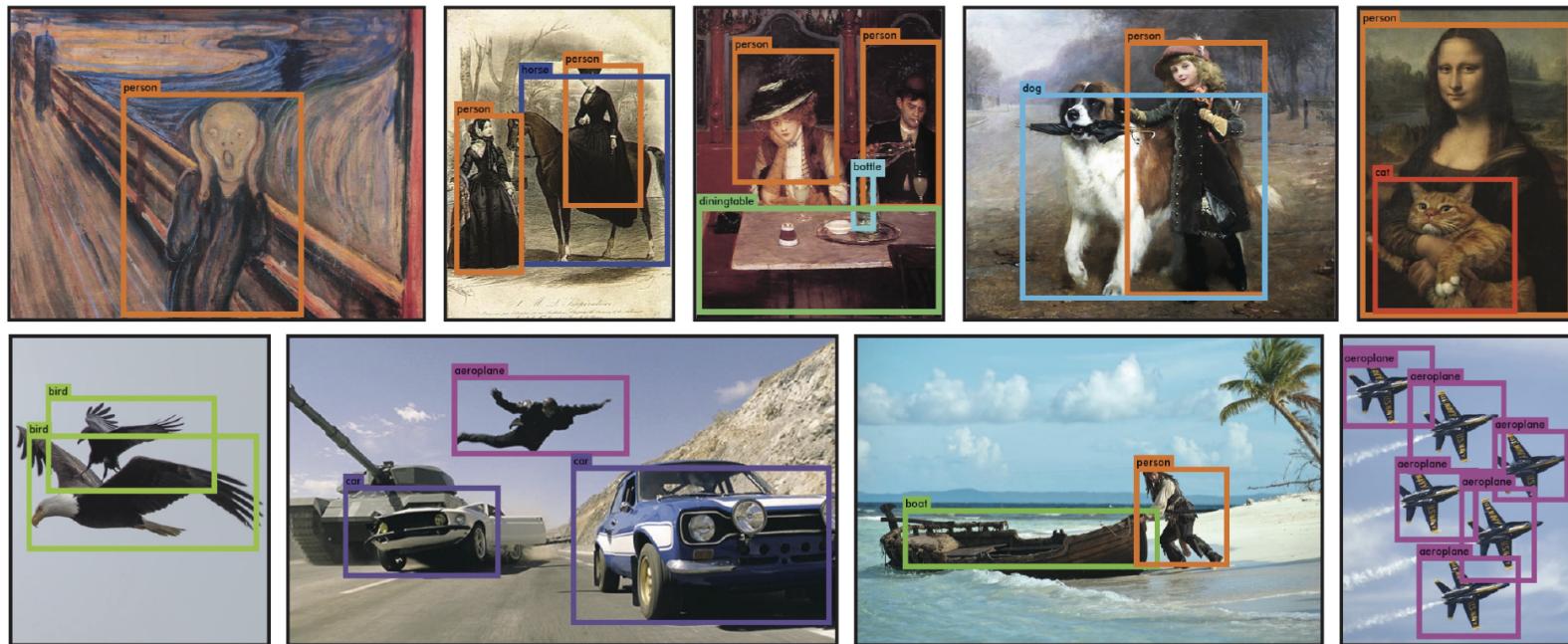
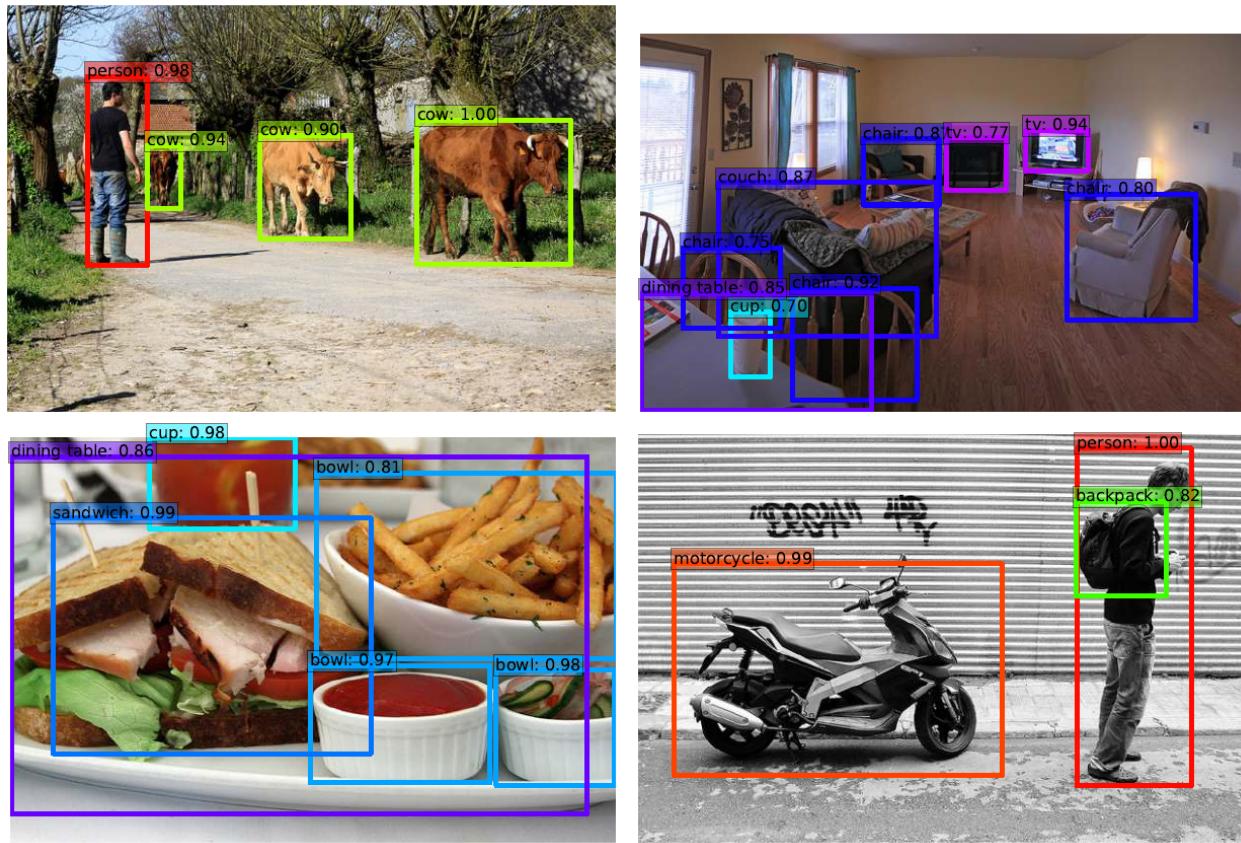


Figure: YOLO results



# Outline

1 Goals

2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation**
- Object tracking - videos

# Scene labeling - DAG-RNN

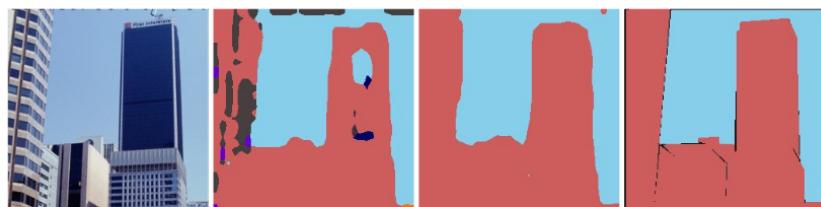


Input Image

CNN

DAG-RNN

Ground Truth



Input Image

CNN

DAG-RNN

Ground Truth

# Outline

1 Goals

2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- **Object tracking - videos**

# Object tracking

Object Detection in a video:

- YOLO <https://pjreddie.com/darknet/yolo/>
- YOLO - James Bond <https://www.youtube.com/watch?v=V0C3huqHrss>

# Deep Learning - Many many other applications !

- ① Image and video (super-resolution, 3D, Image captioning), medical applications...
- ② Self Driving cars (scene segmentation, etc.)
- ③ In NLP (language, text), automatic translation, voice recognition, text generation, next word completion ... (Virtual assistants)
- ④ Recommendation systems...
- ⑤ For time series, complex datasets too...

But also beyond the supervised settings: using adversarial networks (GANS) for generation of images, faces, voice, etc.

Click on the person who is real.



<http://www.whichfaceisreal.com/>

# Limits and problems

Limits of Deep Learning:

# Limits and problems

Limits of Deep Learning:

- Interpretability

# Limits and problems

## Limits of Deep Learning:

- Interpretability
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!

# Limits and problems

## Limits of Deep Learning:

- Interpretability
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory !

# Limits and problems

## Limits of Deep Learning:

- Interpretability
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory !
- Ethical and practical concerns !

# Deep Learning: failures

CNN are not always “robust” to adversarial attacks !

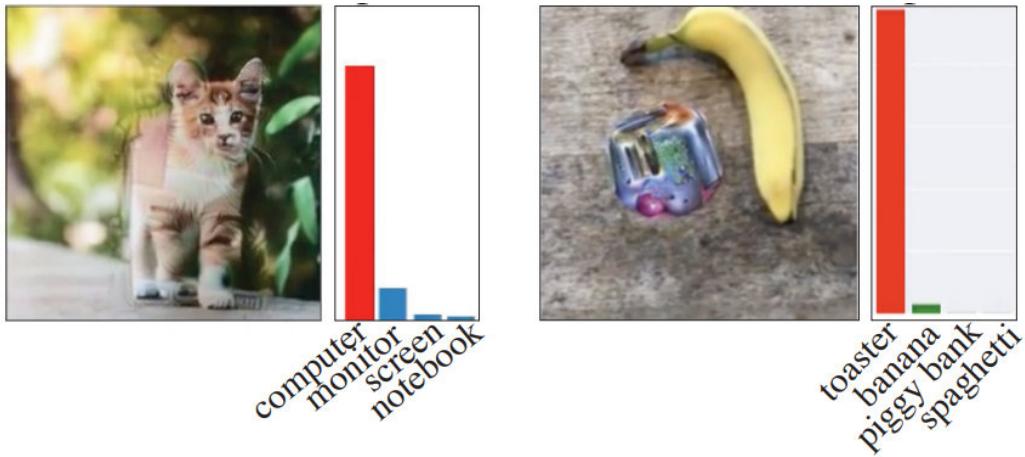
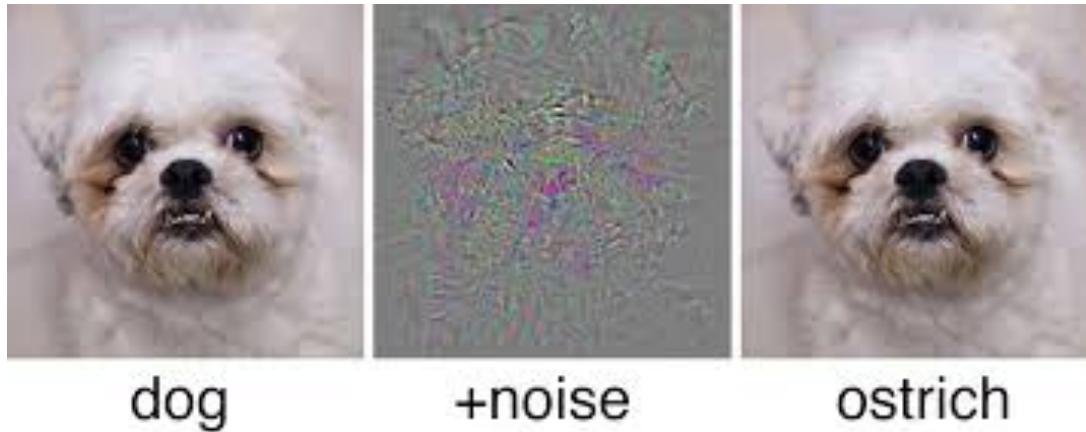


Figure: What happened here ??

## Deep Learning: failures

CNN are not always “robust” to adversarial attacks !



Adding a small well chosen noise can completely fool a CNN !

⇒ Can we trust deep networks on cars, medical applications, planes...?

# Deep Learning: failures

There can be biases in Learning sets:



# Deep Learning: failures

There can be biases in Learning sets, or AI can be manipulated...

- ① Chatbot becomes racist
- ② Google apologises for Photos app's racist blunder

# Deep Learning: questionable applications

Are some applications just bad (they all already exist :( ) ?

- Deep Fakes: insert someone in a video.
- Underground exploration: finding new fossil resources.
- Voice imitation: what if I cannot check who is calling me.
- Military applications + automatic weapons.
- Mass surveillance.

# Deep Learning: questionable applications

Are some applications just bad (they all already exist :( ) ?

- Deep Fakes: insert someone in a video.
- Underground exploration: finding new fossil resources.
- Voice imitation: what if I cannot check who is calling me.
- Military applications + automatic weapons.
- Mass surveillance.

# Deep Learning: questionable applications

A challenging problem: chain of responsibility. The same tools are used for positive and negative applications, e.g.:

- mass surveillance
- tumor recognition

are both based on image recognition...

Good news ! the research community is very aware of the situation.

Some references:

- Asilomar AI Principles [https://www.oecd.org/going-digital/  
ai-intelligent-machines-smart-policies/conference-agenda/  
ai-intelligent-machines-smart-policies-oheigearthaigh.pdf](https://www.oecd.org/going-digital/ai-intelligent-machines-smart-policies/conference-agenda/ai-intelligent-machines-smart-policies-oheigearthaigh.pdf)  
<https://futureoflife.org/ai-principles/>
- European guidelines <https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai>
- AI for Good <https://ai4good.org/>

# Conclusion: Deep Learning in one slide

- **How does it work:**
  - ▶ Automatically learn representations of observations
  - ▶ Learn highly non-linear models.
- **What does it require:**
  - ▶ Large datasets with structure
  - ▶ Computational power
- **Why now:**
  - ▶ Combination of the 2 points above
  - ▶ investment !
- **Some Applications**
  - ▶ Image classification; object / face recognition
  - ▶ Self driving cars
  - ▶ Automatic Translation, Information extraction
  - ▶ Caption Generation
  - ▶ Ads, recommendation systems, etc.

**Goals:** Understanding core concepts of Deep Learning.

- When and how it works on paper (data, models, architecture)
- How to implement a simple neural network with Python.
- Overview of some of the main applications and challenges.