

Deep Learning

Aymeric DIEULEVEUT

May 2020

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 !

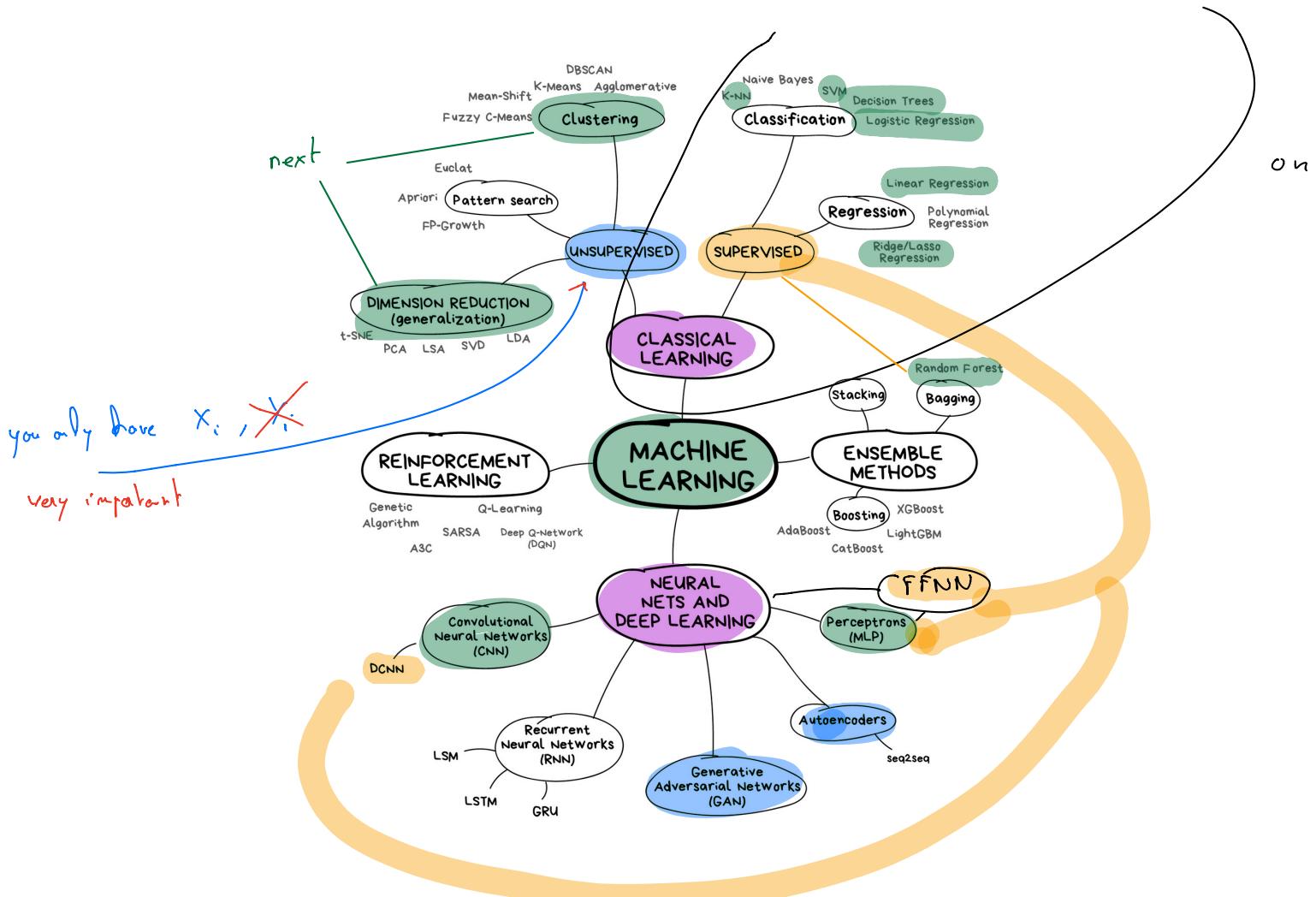
- **Some Applications**

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

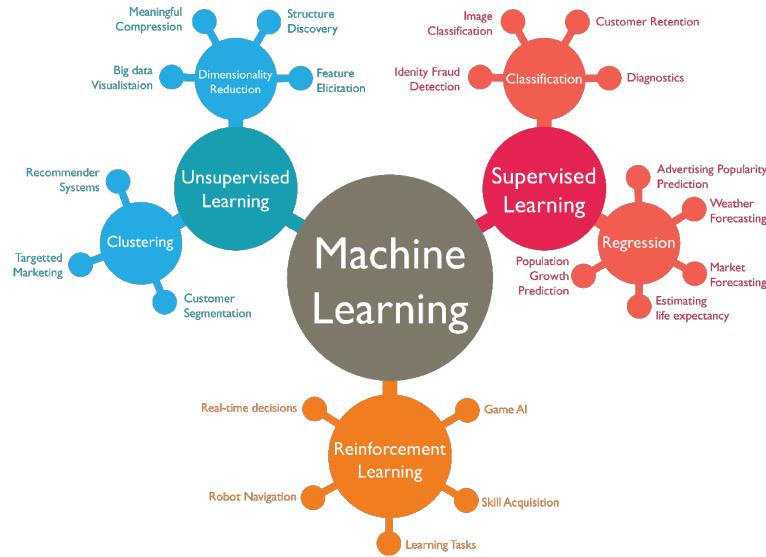
Goals: Understanding core concepts of Deep Learning.

- 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.

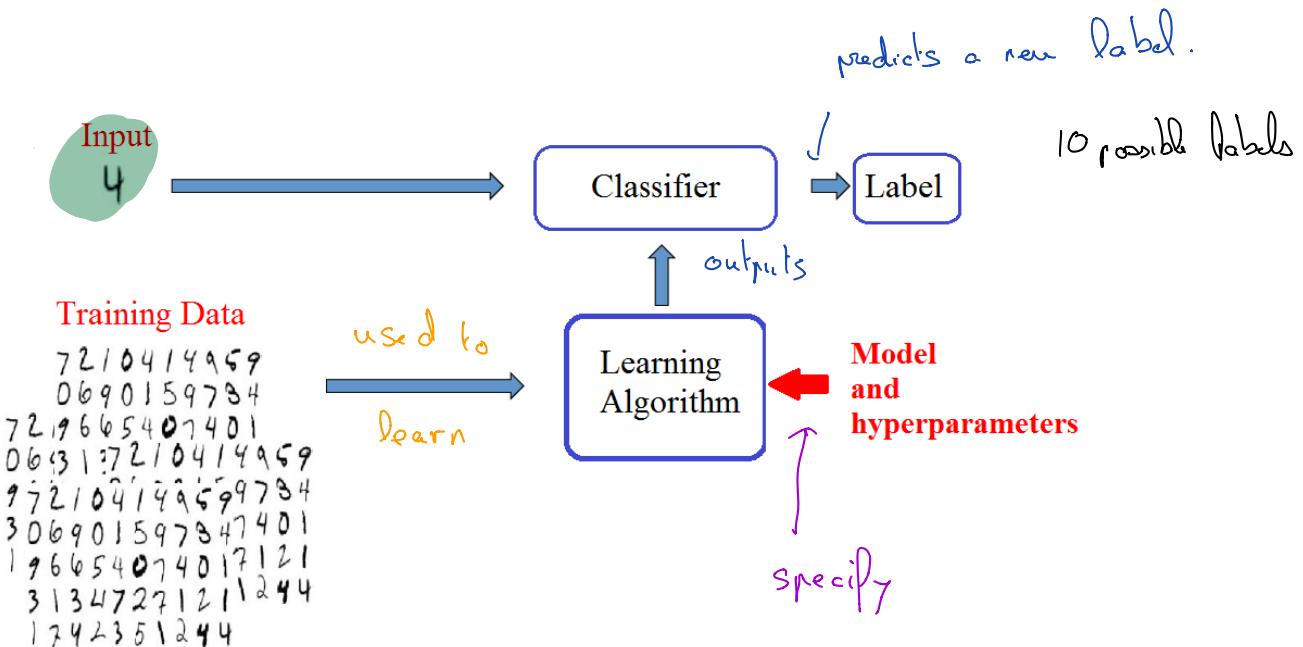
Machine Learning



Machine Learning



Summary of previous lectures on Machine Learning



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

multi-class K classes

Question: predict the label of a new one.

digits $K=70$

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. given X_{new}

2 approaches:

$$\text{Learn } f \circ f / f(x_{\text{new}}) \approx Y_{\text{new}}$$

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 ℓ \approx cost of predicted $f(X)$ instead of Y
- Goal:
- Approach:

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

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

ERM
empirical
risk

minimizing ℓ

choose ℓ

do ℓ

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 ℓ
- 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 !

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

Logistic Loss

From logistic regression to Multi Layer Perceptron 3

$(x_1, \dots, x_d) \in \mathbb{R}^d$

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

Point $x^1 \dots x^n$: observation $x^i \in \mathbb{R}^d$

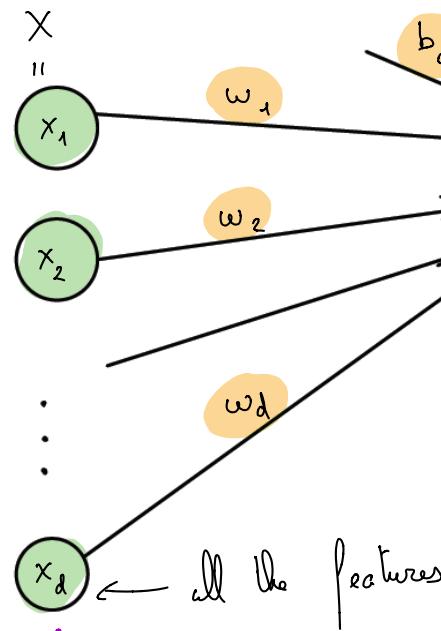
$$x^i = (x_1^i \dots x_d^i)$$

Learn w

Predict

$$\sigma(w^T X) \in [0; 1]$$

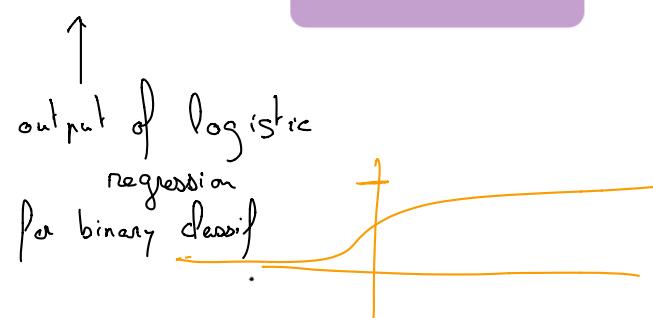
Logistic Regression.



inputs

to learn

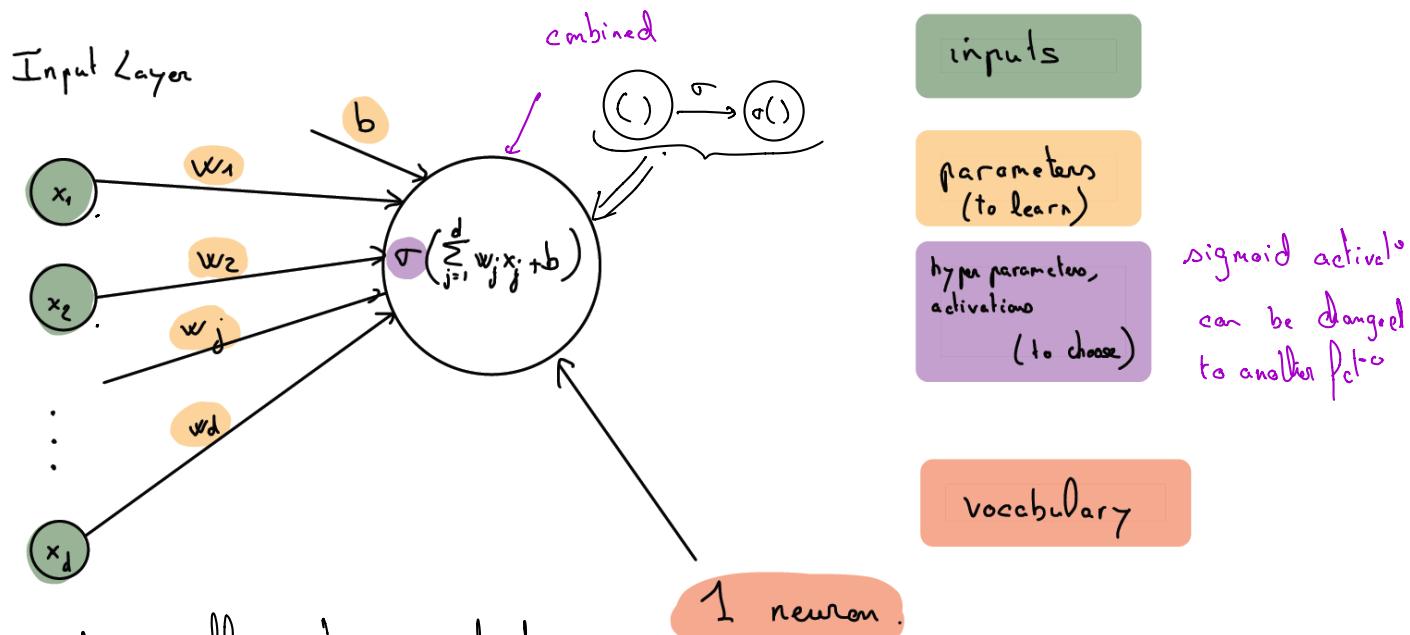
to specify: ϕ



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



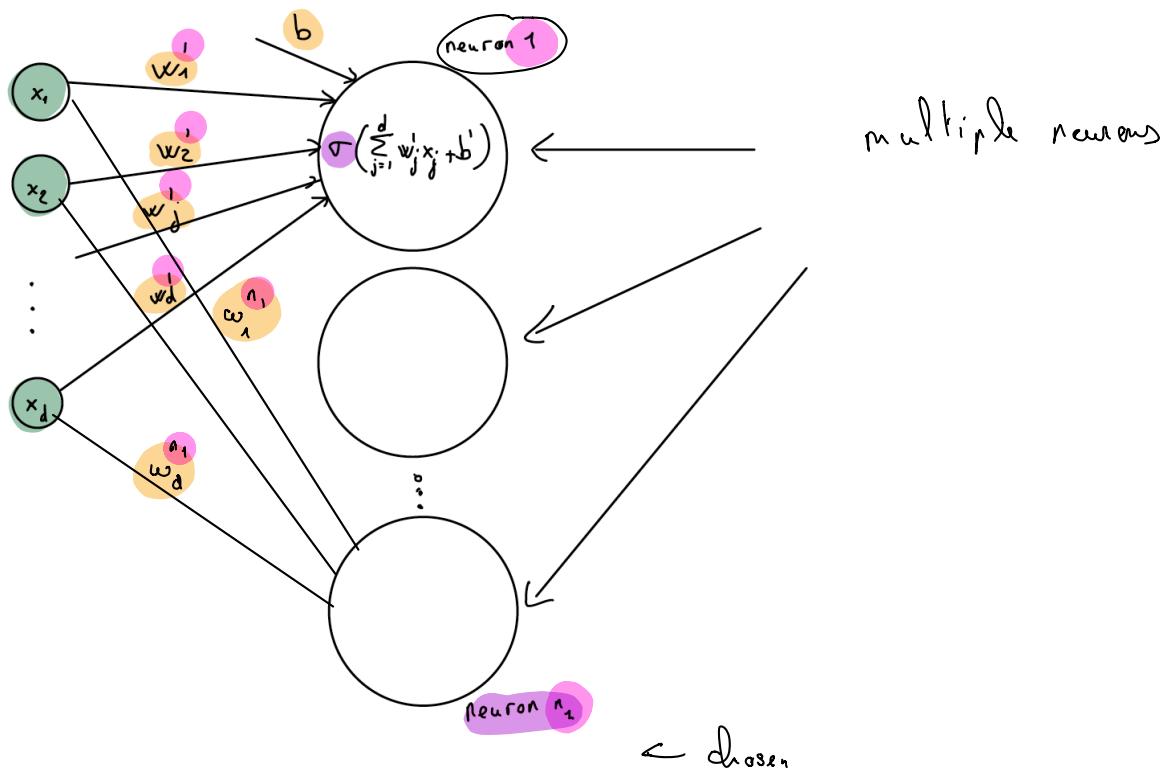
Fully connected : all inputs connected to the first neuron

This is a neural network, with one neuron !

From logistic regression to Multi Layer Perceptron 4

Extend to

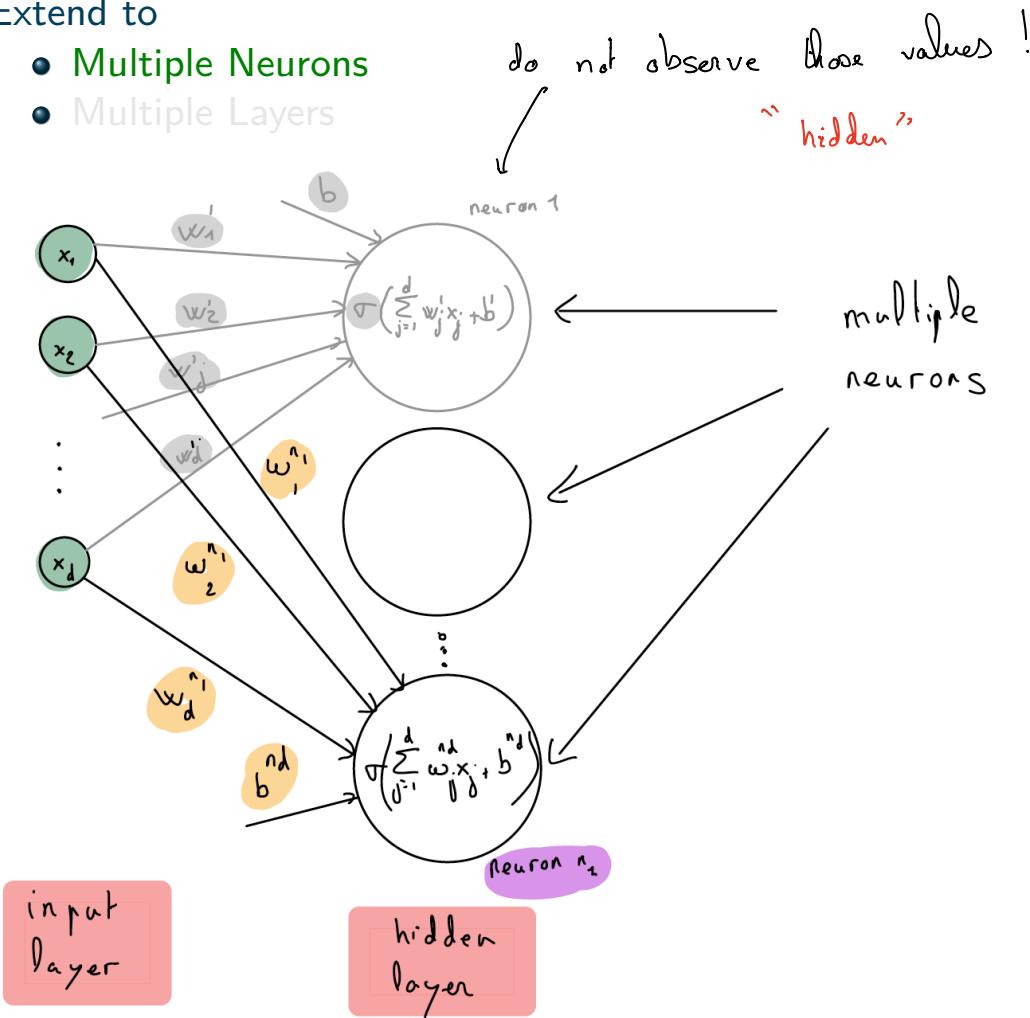
- Multiple Neurons \equiv
- Multiple Layers



From logistic regression to Multi Layer Perceptron 4

Extend to

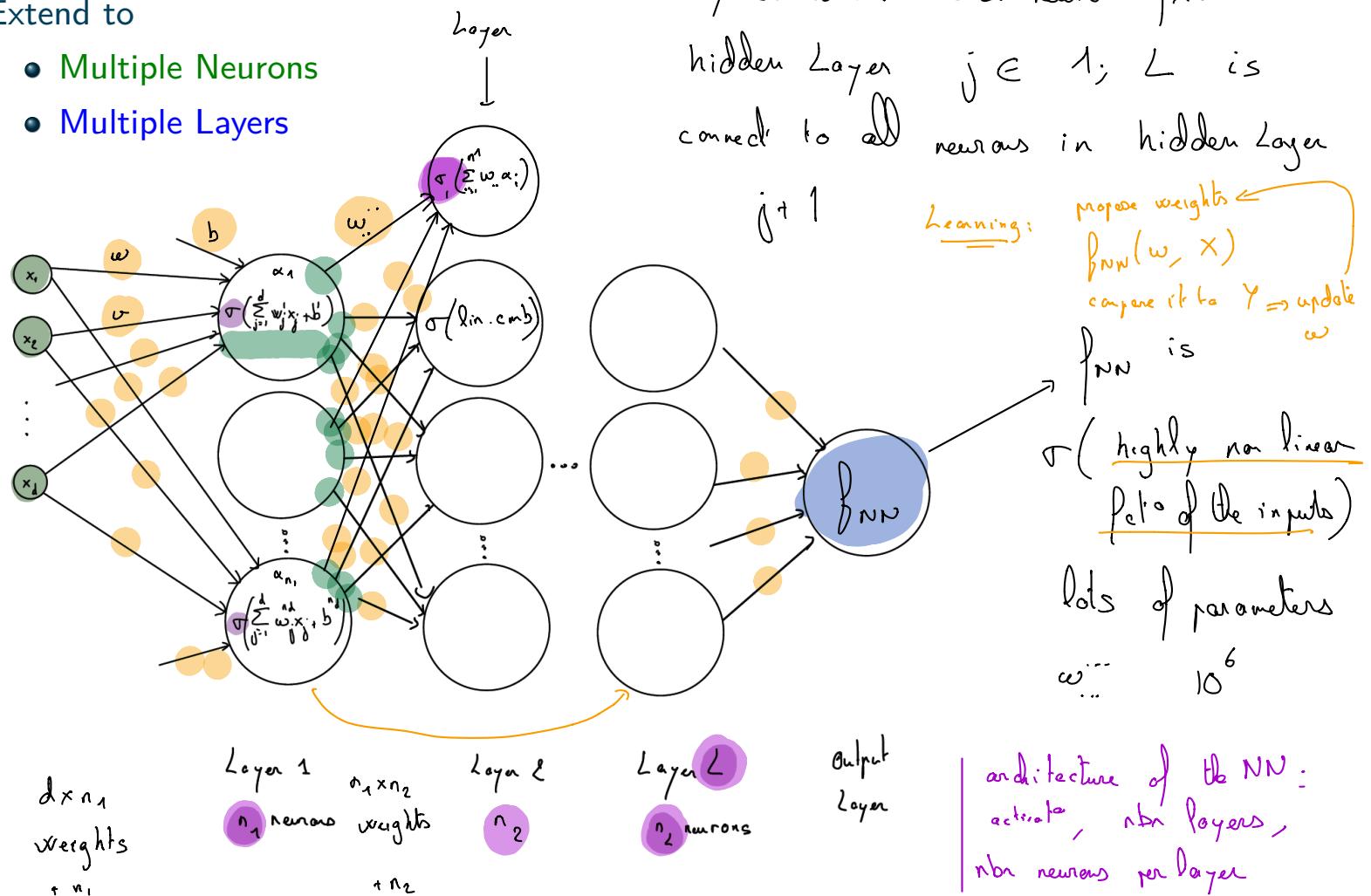
- Multiple Neurons
- Multiple Layers



From logistic regression to Multi Layer Perceptron 5

Extend to

- Multiple Neurons
- Multiple Layers



From logistic regression to Multi Layer Perceptron 5

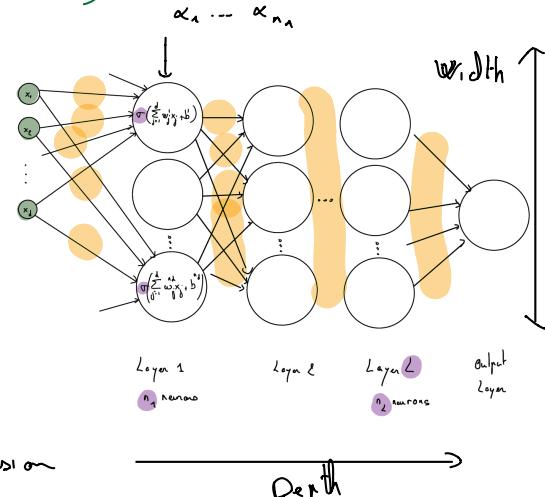
Neural network:

- 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.

$$\mathcal{F}_{MLP} = \{f(X; W, b) = \sigma(b_0 + \underbrace{W_n \sigma(\dots)}_{W \in \mathbb{R}^{\dots}, b \in \mathbb{R}^{\dots}) (b_1 + \underbrace{W_1 x})\},$$

much much bigger class of f(x) than for logistic regression

average loss in training set



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

What needs to be learned

What needs to be chosen

Vocabulary

Depth : L :

Deep : L is big

Width : n_1, \dots, n_L

From logistic regression to Multi Layer Perceptron 5

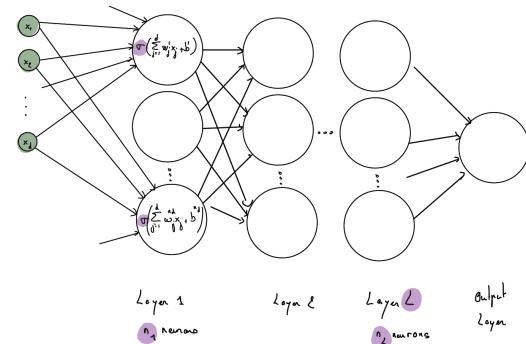
Neural network:

- 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.

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

$$W \in \mathbb{R}^{n \times n} \quad \# \text{ param} \approx 10^6$$



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

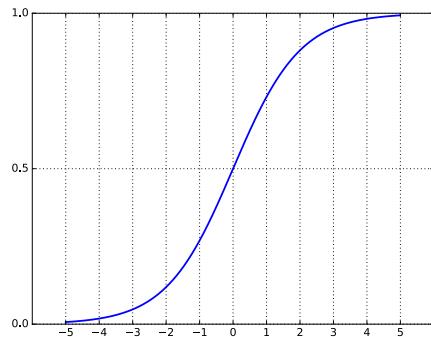
What needs to be chosen

- Activation σ
- 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

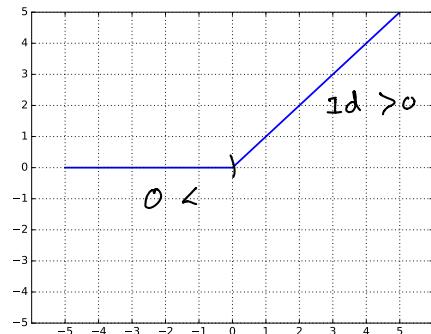
2 possible activations: Sigmoid and Rectified Linear Unit



Sigmoid function

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

- ➊ Saturated function: gradient killer \rightarrow need for rescaling data



Rectified Linear Unit (ReLU)

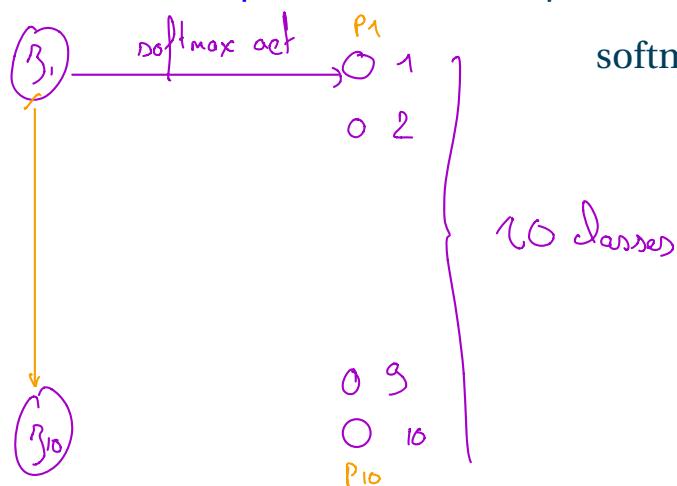
- $x \mapsto \max(0, x)$
 - Pros & cons:
 - ➊ Not a saturated function. Kills negative values.
 - ➋ Empirically, convergence is faster than sigmoid/tanh.
 - ➌ Plus: biologically plausible
- ← used
more often

How to deal with multi-class output: softmax activation

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 $\{1, \dots, K\}$:

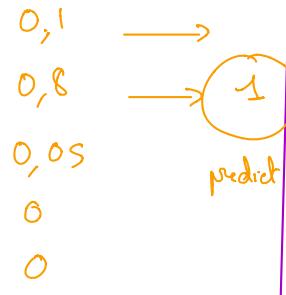


$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$

Logistic $\{0, 1\}$

$w^\top X$: nbr

$\sigma(w^\top X)$: probability
in $\{0, 1\}$



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 ?

1. Hardware:

- CPU :
- GPU : graphical proc. units
- TPU



Keras - A few lines !¹

1 hidden layer FC NN

What do we need to specify?

\hookrightarrow layers , $n_1 \dots n_L$ neurons/layer

activations

n. 128 neurons in HL1 transform input

What do we need to do next?

① Step 3: specify the model

② Compile the model: * optimizer

* Loss

③ fit the model: how many iterative algo steps to make = epochs

Start 3:30

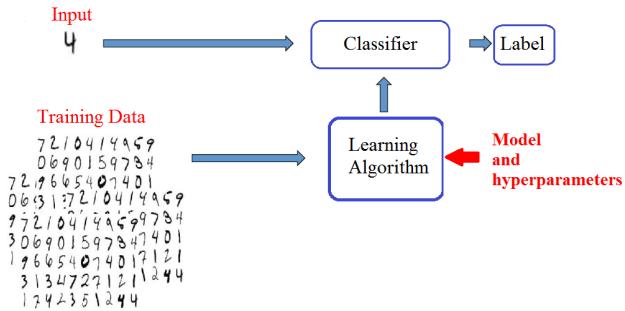
```
model_ffnn = Sequential()  
model_ffnn.add(Flatten(input_shape=input_shape))  
model_ffnn.add(Dense(128, activation='relu'))  
model_ffnn.add(Dense(num_classes, activation='softmax'))
```

1 Notebook Python

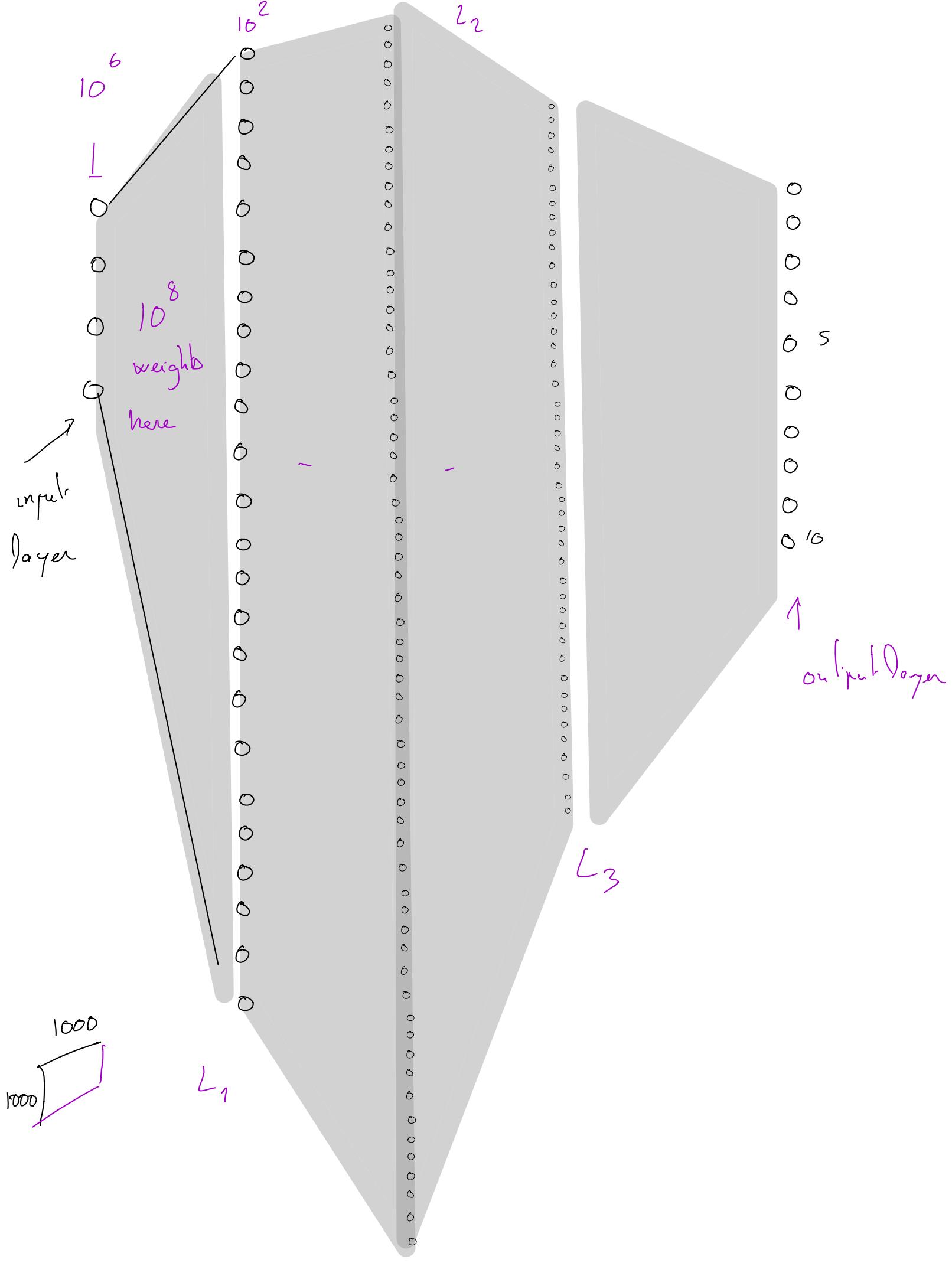
Keras - A few lines !¹

What do we need to specify?

What do we need to do next?



¹Notebook Python



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

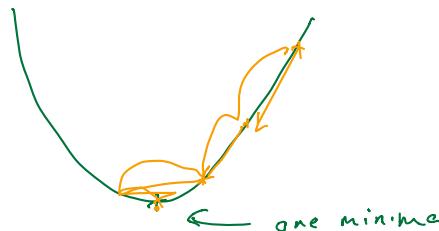
Goal

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

How to minimize a function?

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

going in the direction where it decreases the most.



Optimization

Goal

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

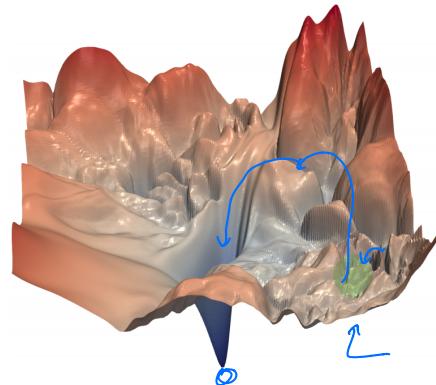
on \hat{y} on y

Loss ($\ell(x)$)
 $\sigma(b + w_1 x_1 + w_2 x_2 + \dots)$

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.)



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: GANs
- 2016: AlphaGo
- 2018: BERT

7
recently
7

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)**.

↳ works for sequences

Outline

1 Goals

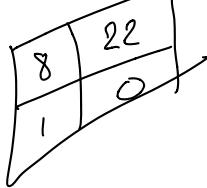
2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

4 Applications

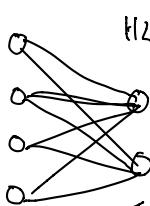
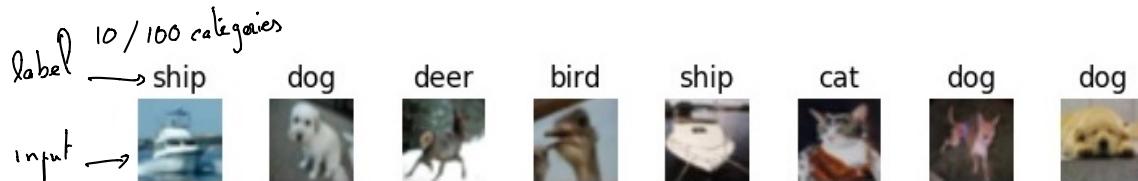
- Image classification
- Pose, action detection
- Object detection and segmentation
- Scene labeling - Semantic segmentation
- Object tracking - videos

CNN



$$\left\{ \begin{array}{l} \omega_1 8 + \omega_2 22 + \omega_3 1 + \omega_4 0, \quad w \in \mathbb{R}^4 \\ \omega_1 22 + \omega_2 1 + \omega_3 8 + \omega_4 0, \quad w \in \mathbb{R}^4 \end{array} \right.$$

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



$$f_{NN} = \text{same} \quad \text{if I had permuted the pixels in the original image}$$

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

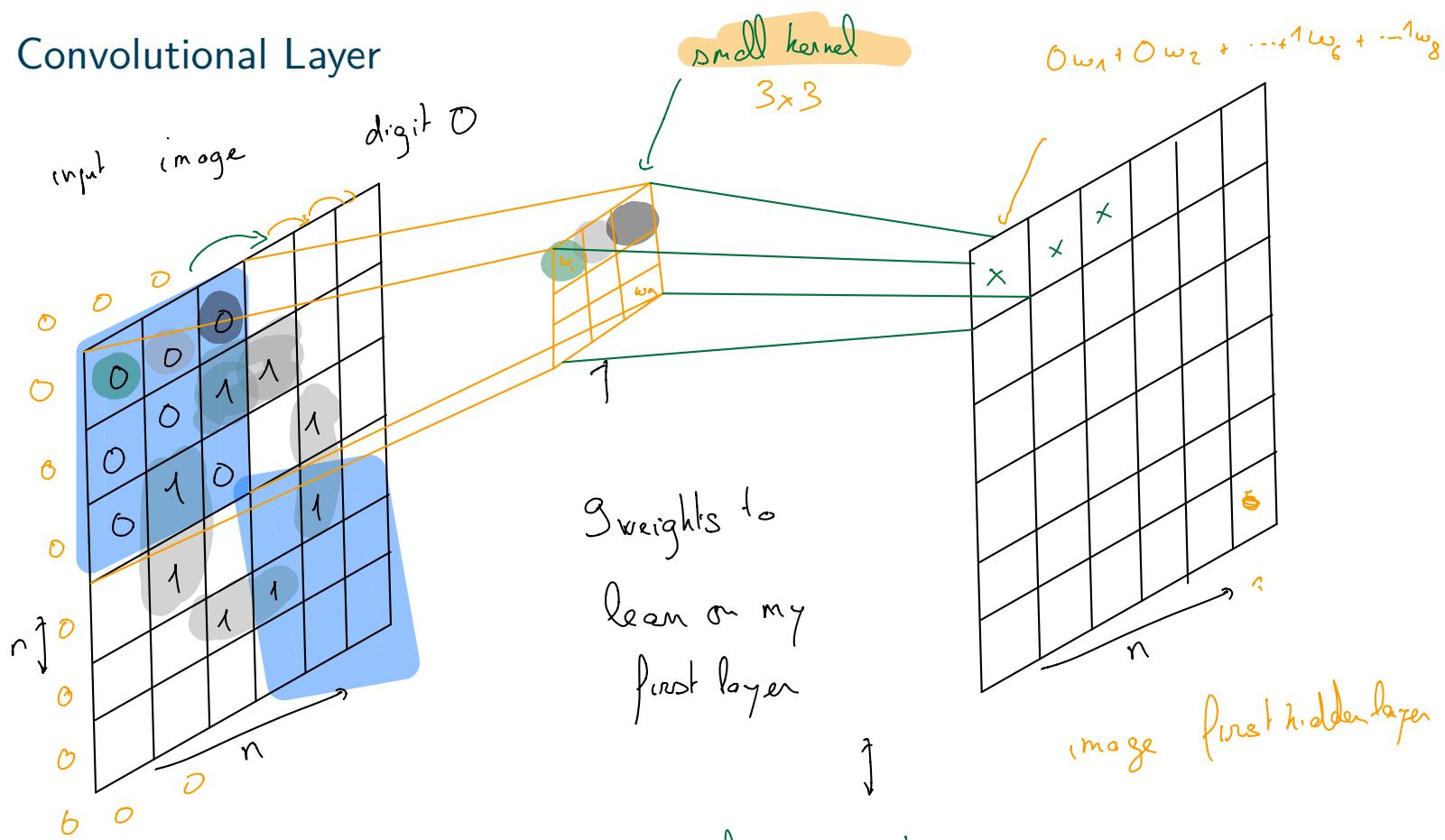
Problems: The multi-layer perceptron:

- 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

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

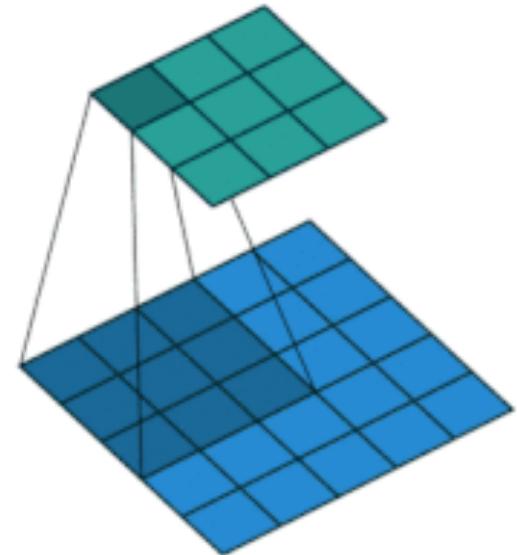


- * reduce the nb of weights
 - * use spatial information

Convolutional Layer

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

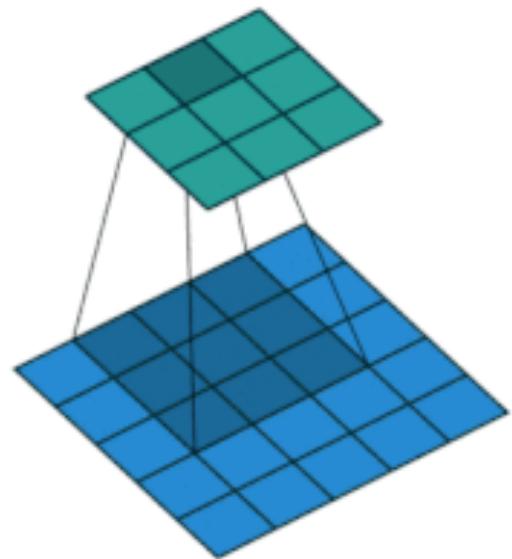
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



Convolutional Layer

3	3_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

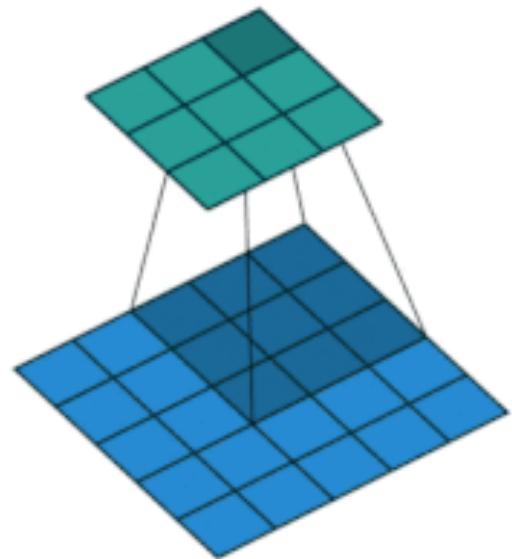
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



Convolutional Layer

3	3	2 ₀	1 ₁	0 ₂
0	0	1 ₂	3 ₂	1 ₀
3	1	2 ₀	2 ₁	3 ₂
2	0	0	2	2
2	0	0	0	1

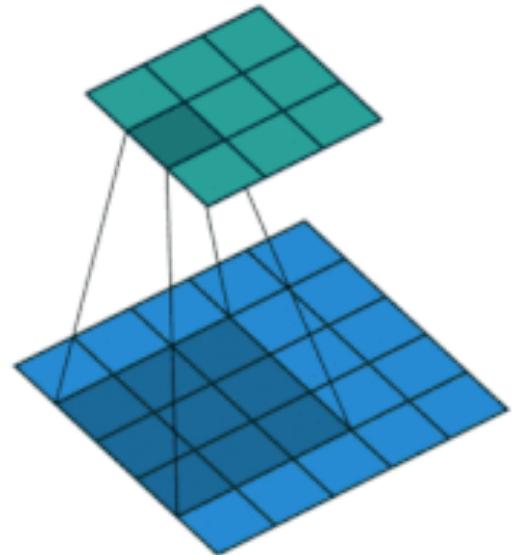
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



Convolutional Layer

3	3	2	1	0
0 ₀	0 ₁	1 ₂	3	1
3 ₂	1 ₂	2 ₀	2	3
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

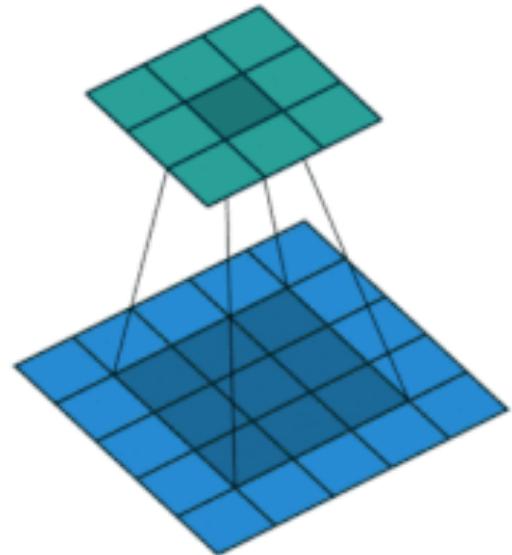
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



Convolutional Layer

3	3	2	1	0
0	0 ₀	1 ₁	3 ₂	1
3	1 ₂	2 ₂	2 ₀	3
2	0 ₀	0 ₁	2 ₂	2
2	0	0	0	1

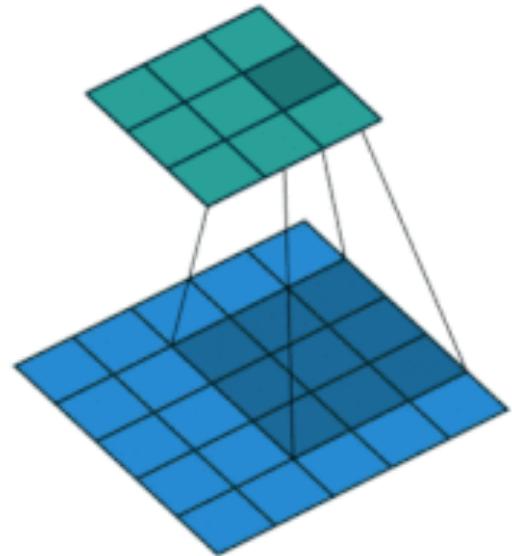
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



Convolutional Layer

3	3	2	1	0
0	0	1_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

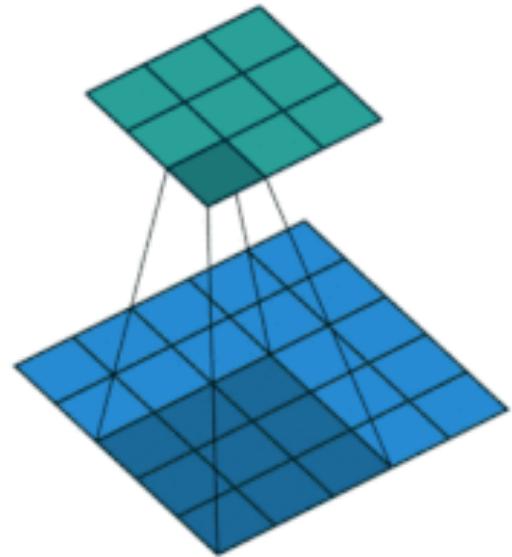
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



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

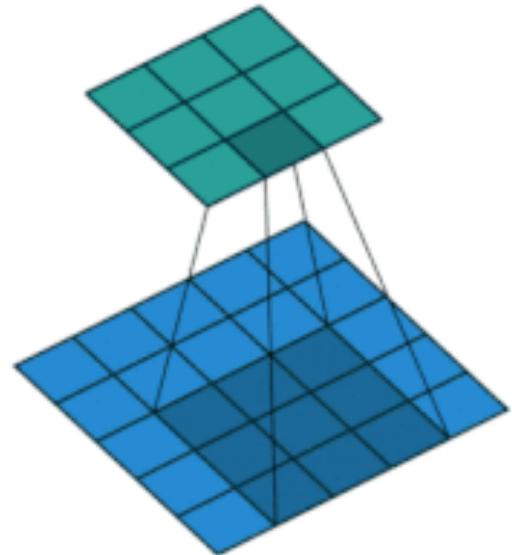
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



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

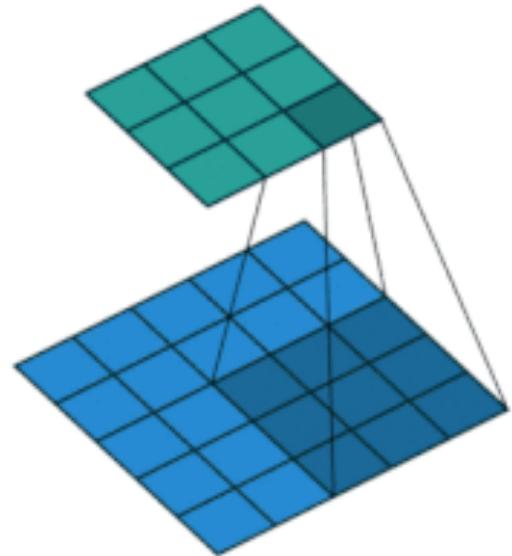
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



Convolutional Layer

3	3	2	1	0
0	0	1	3	1
3	1	2 ₀	2 ₁	3 ₂
2	0	0 ₂	2 ₂	2 ₀
2	0	0 ₀	0 ₁	1 ₂

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)

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)

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)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

308

+

-498

+

164 + 1 = -25

Bias = 1

Output

-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

Bias = 1

Output

-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



-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

+

+ 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

1

487

+

487

Bias = 1

Output

-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:

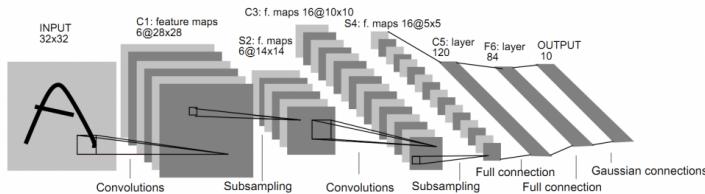


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

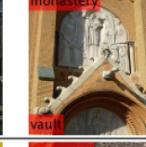
Category	Example Validation Images					
barracouta	 coho barracouta	 coho coho	 tench barracouta	 hammerhead tiger shark	 great white shark great white shark	 coho barracouta
church	 castle church	 beacon castle	 altar church	 monastery vault	 castle castle	 altar vault
spaghetti squash	 zucchini spaghetti squash	 lemon spaghetti squash	 lemon	 ice cream spaghetti squash	 acorn squash butternut squash	 acorn 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	 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

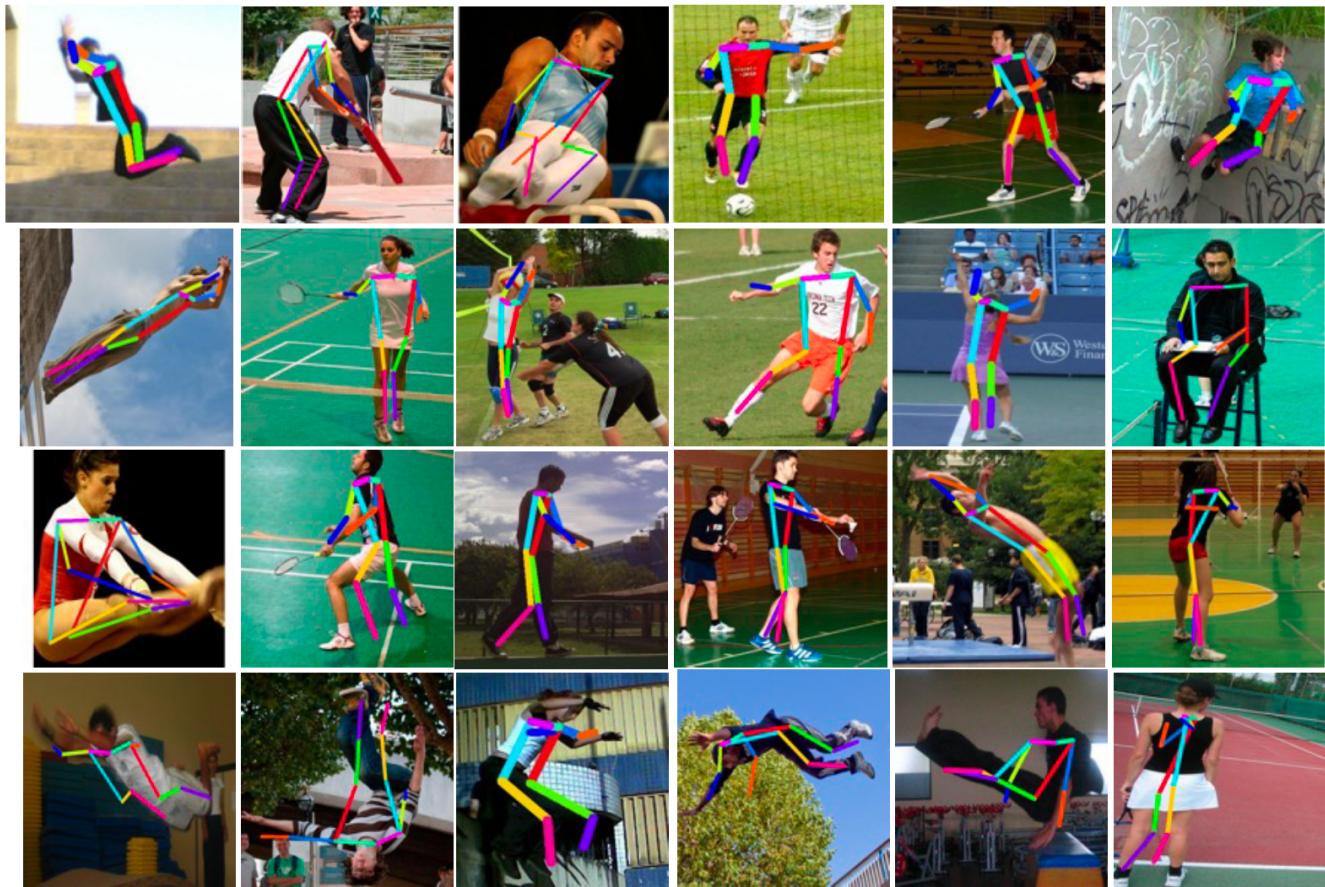
2 The first Neural Network: the Perceptron

3 Convolutional Neural Networks

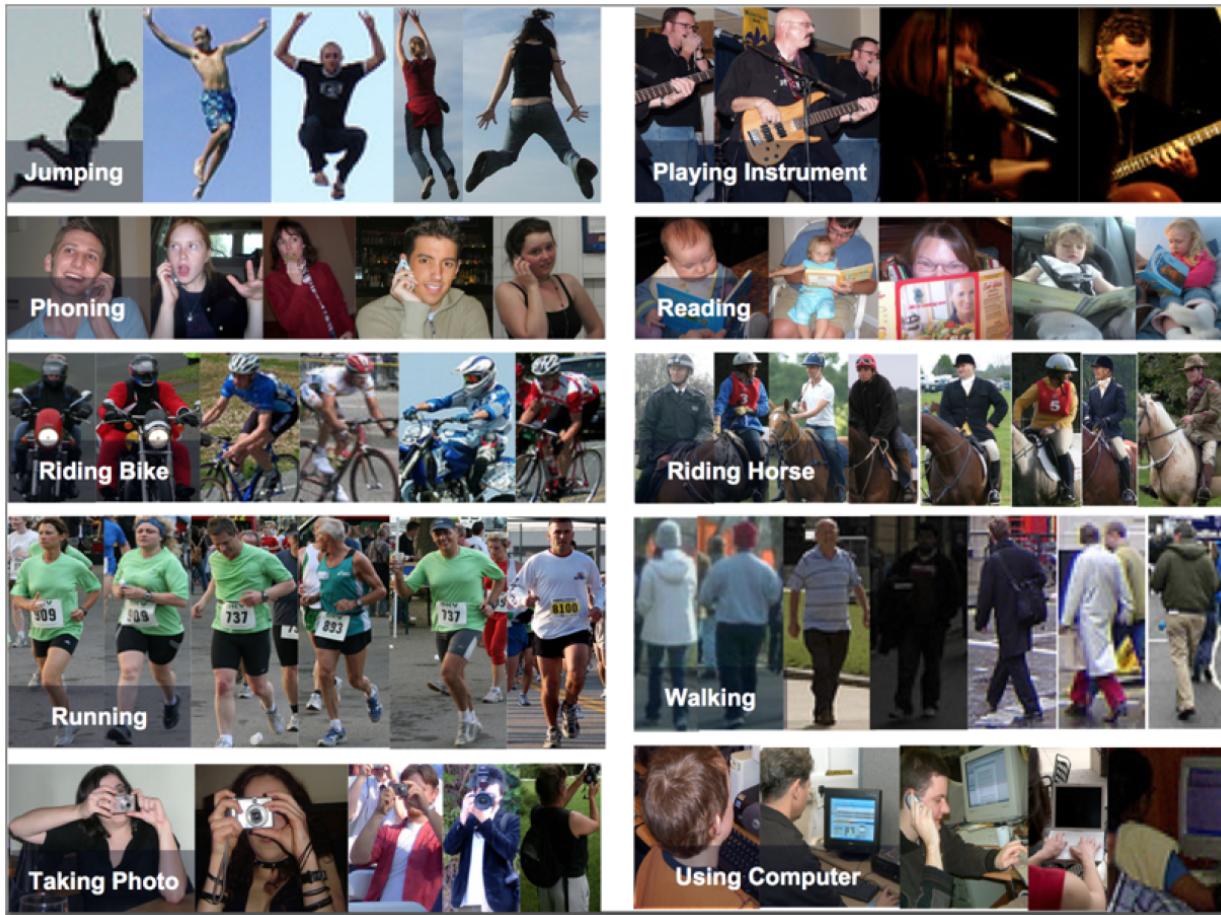
4 Applications

- 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.

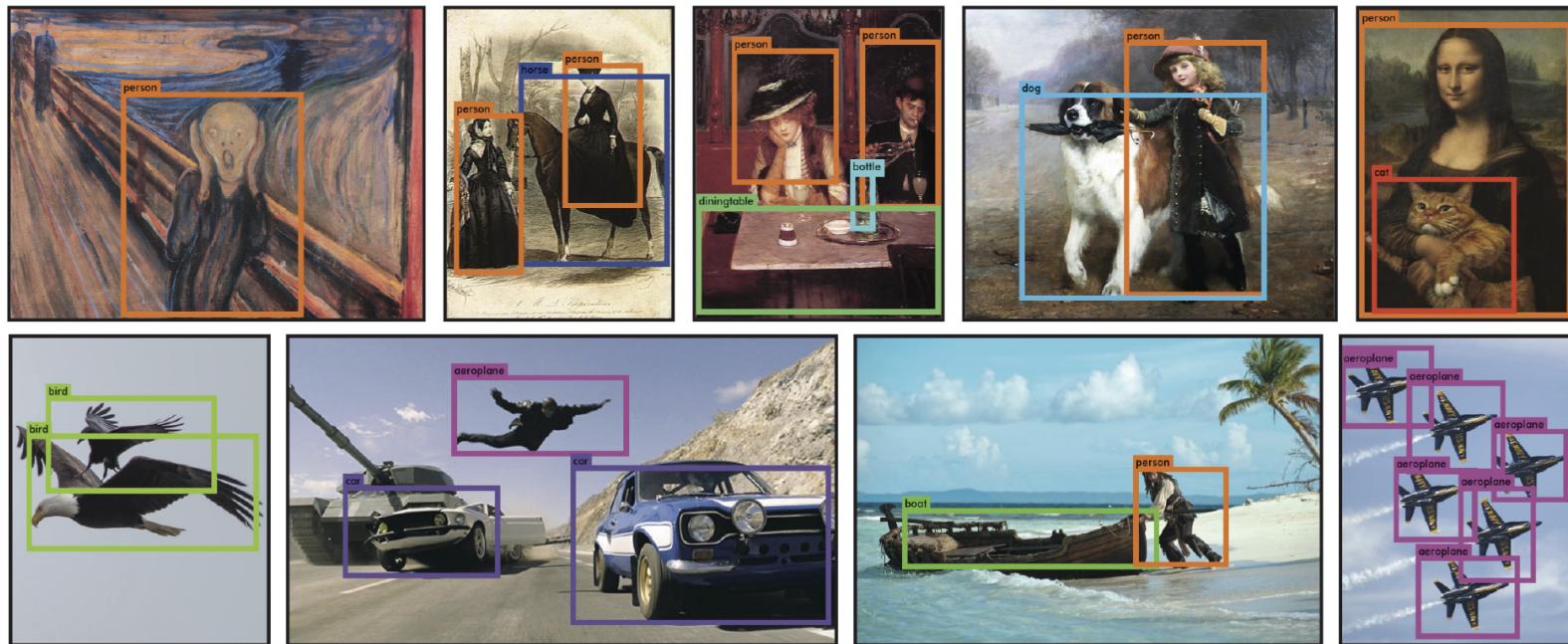
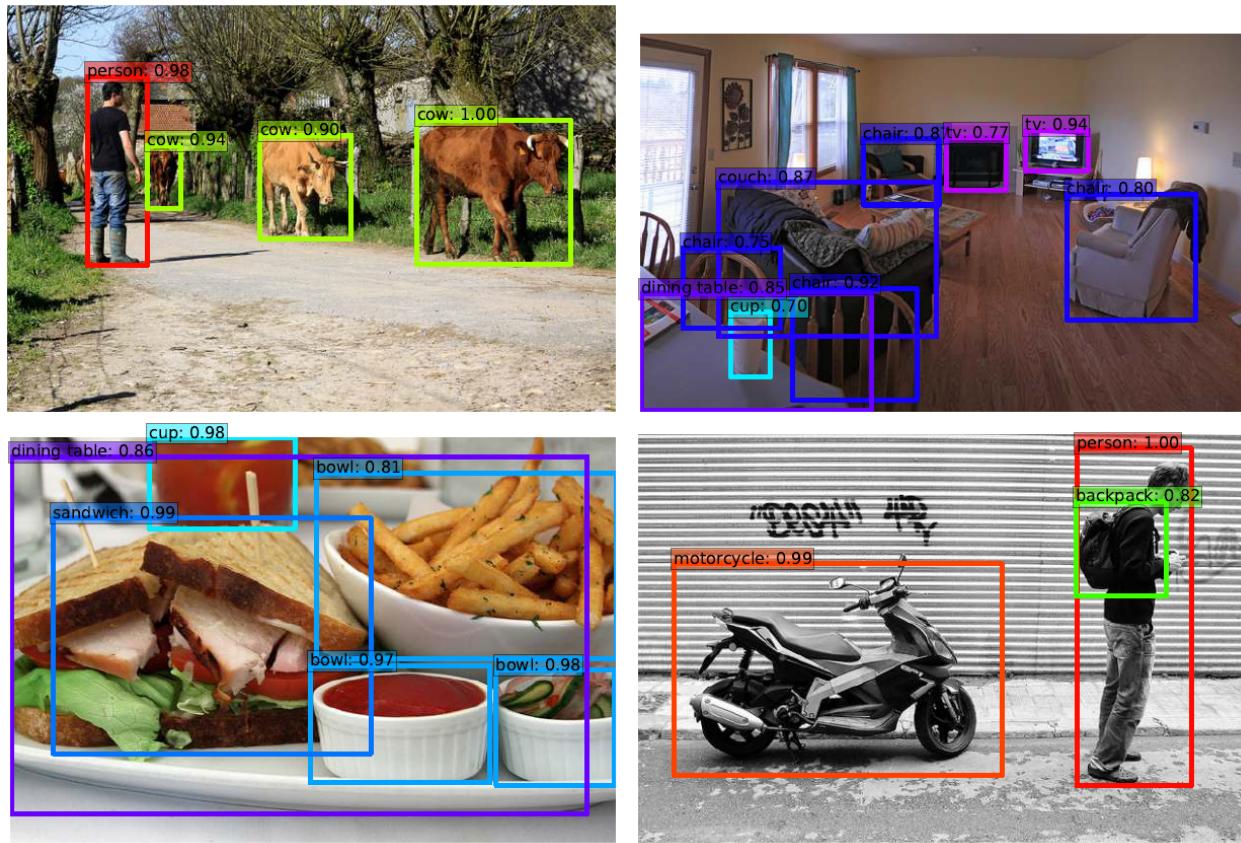


Figure: YOLO results



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

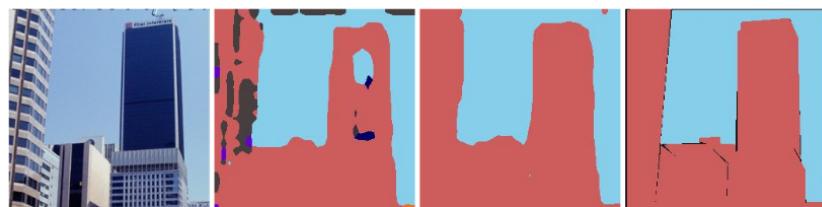


Input Image

CNN

DAG-RNN

Ground Truth



Input Image

CNN

DAG-RNN

Ground Truth

Outline

1 Goals

2 The first Neural Network: the Perceptron

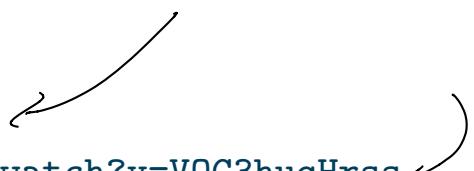
3 Convolutional Neural Networks

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>
- 
- A handwritten curly brace is drawn over the two URLs in the list, grouping them together.

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 !

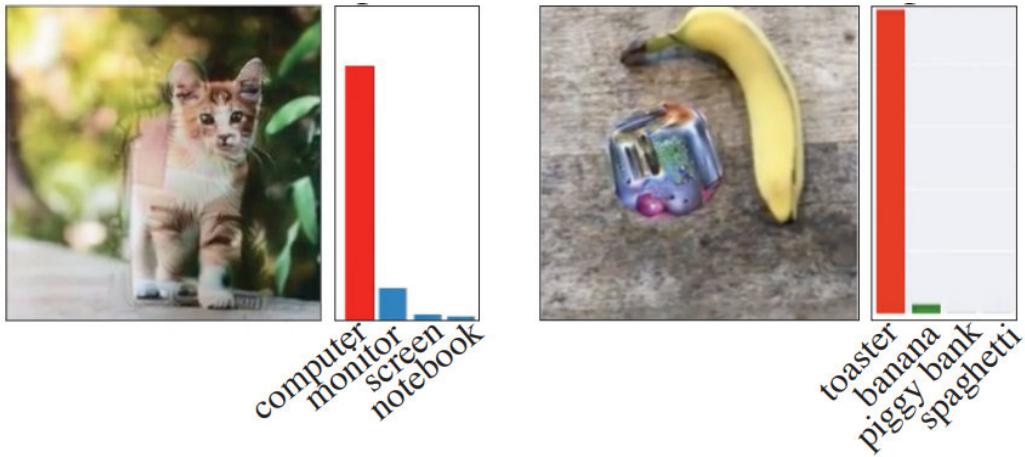
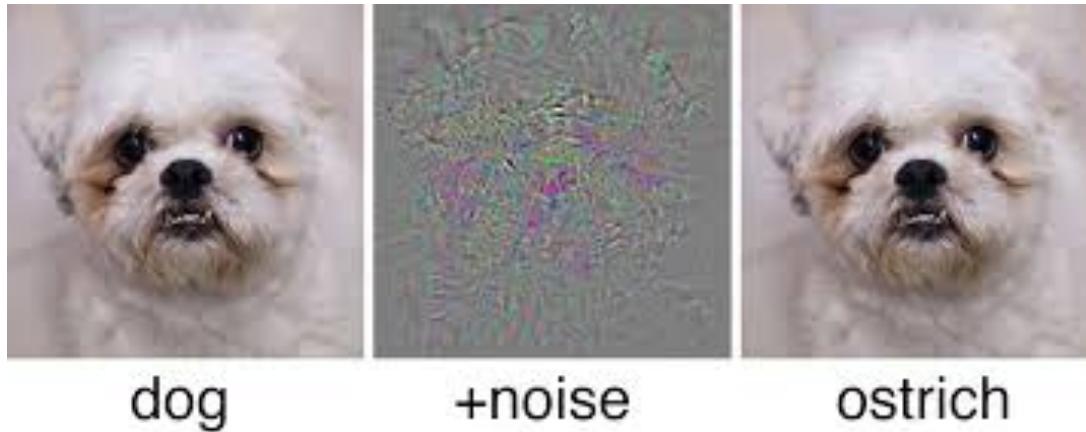


Figure: What happened here ??

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

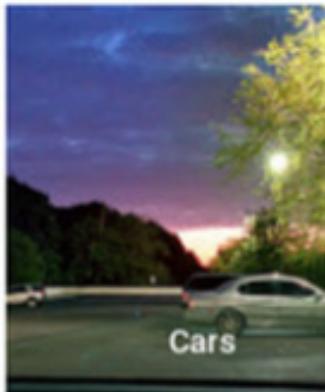
There can be biases in Learning sets:



Skyscrapers



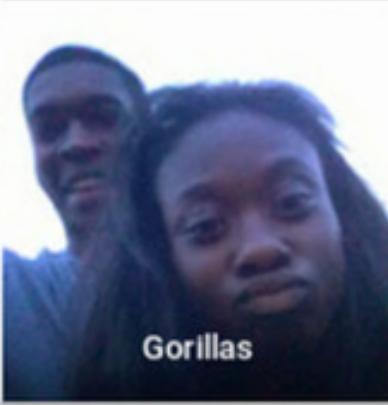
Airplanes



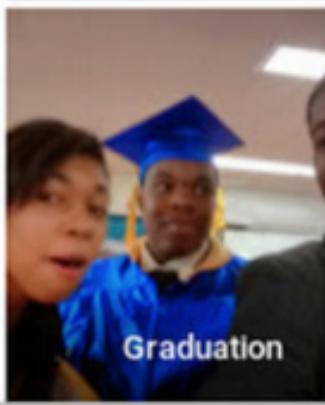
Cars



Bikes



Gorillas



Graduation

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://futureoflife.org/ai-principles/>
- 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.