# Deep Learning

Aymeric DIEULEVEUT

May 2021

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

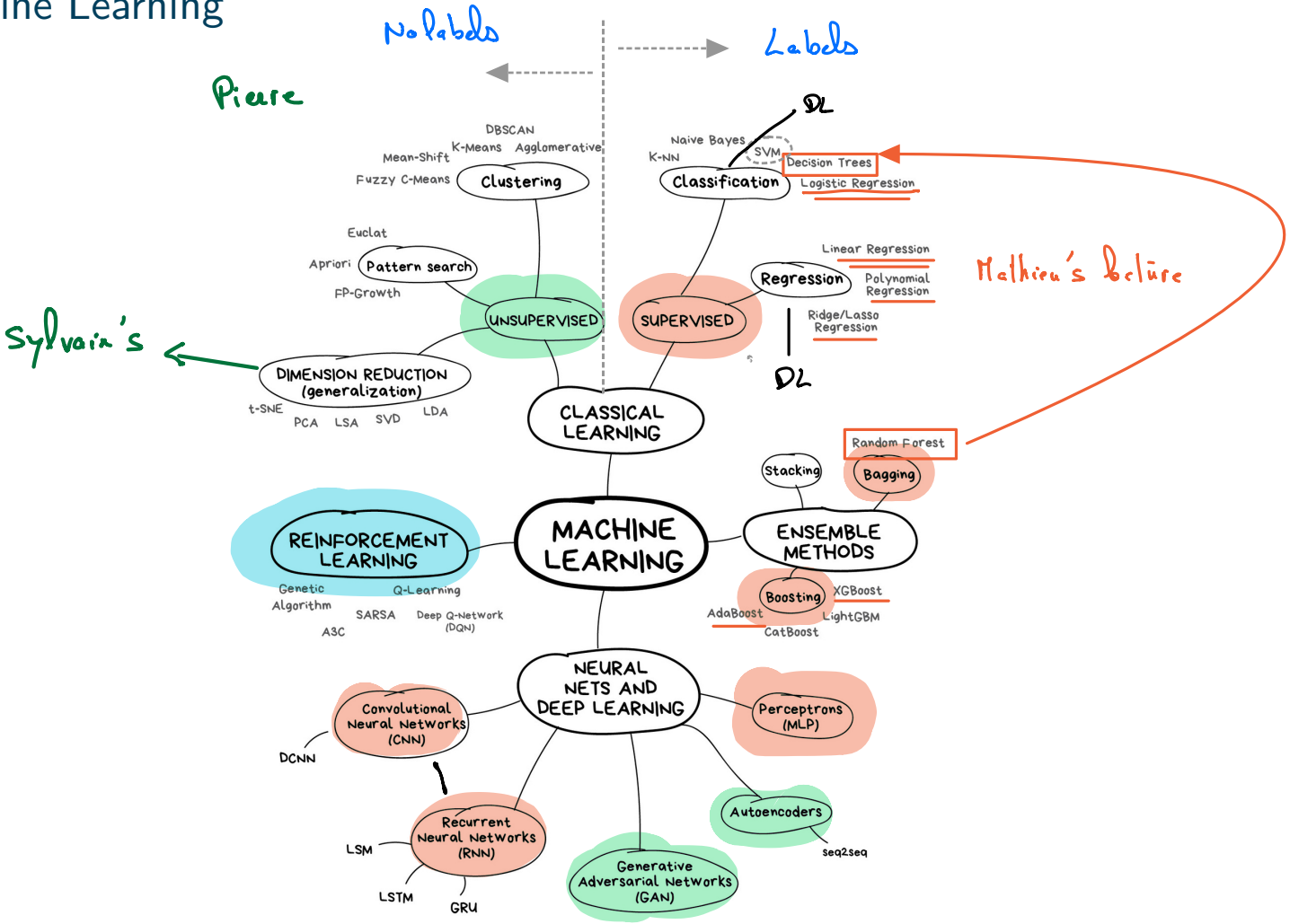
# Deep Learning in one slide + Goals

- **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.

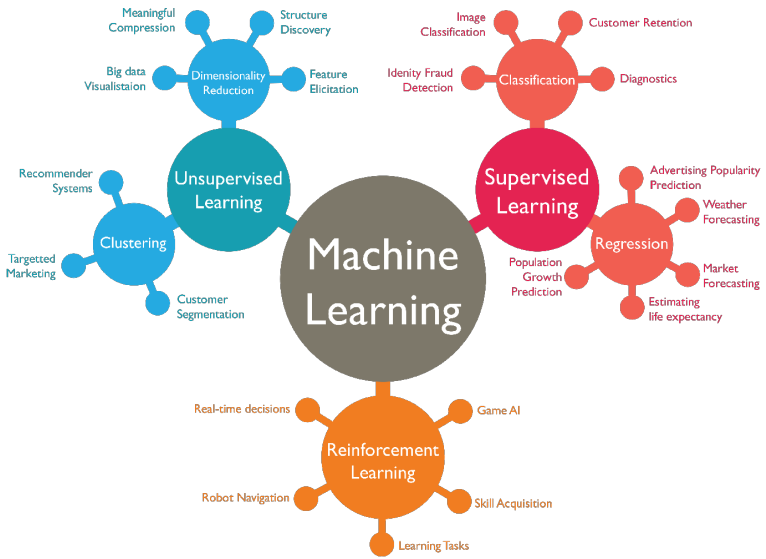
**Goals:** Understanding core concepts of Deep Learning.

- 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.

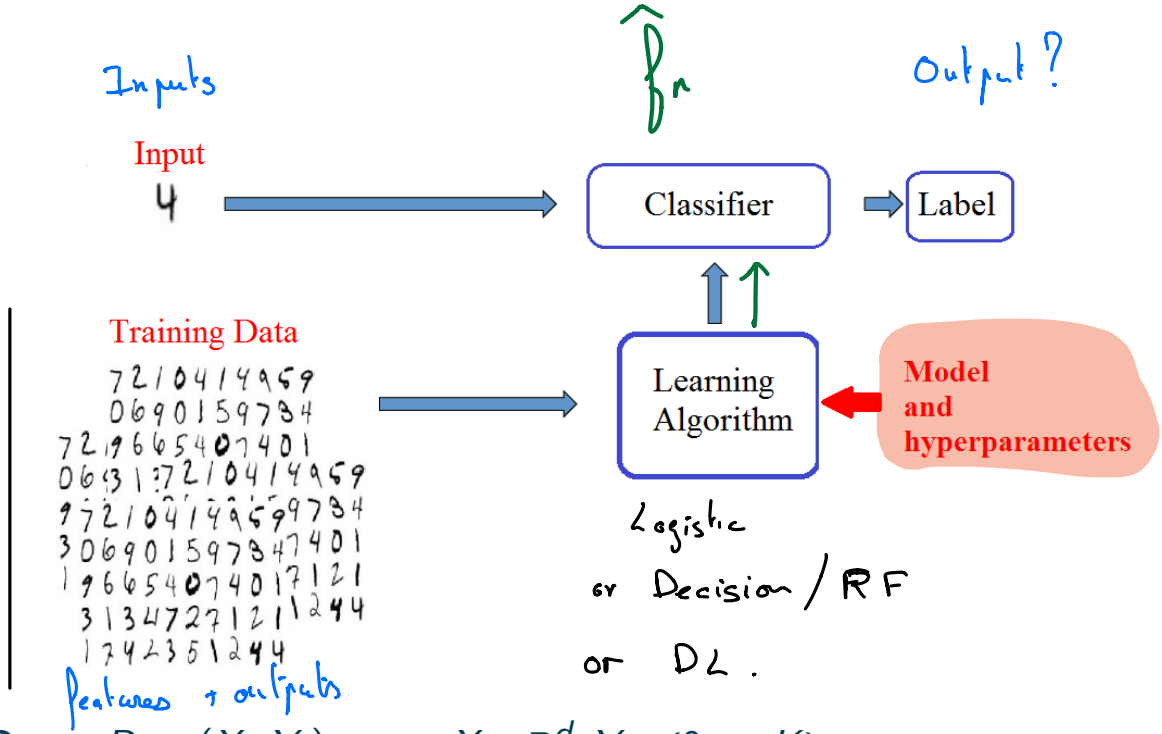
# Machine Learning



# Machine Learning



# Summary of previous lectures on Machine Learning




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

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

# Outline

# 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:
$$\min_{f \in \mathscr{F}} \mathbb{E}[\ell(f(X), Y)]$$
- Approach:
$$\min_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$$

# 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:
$$\min_{f \in \mathscr{F}} \mathbb{E}[\ell(f(X), Y)]$$
- Approach:
$$\min_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$$

Logistic regression can be seen both ways !

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

Logistic regression. : output $Y_i \in \{0, 1\}$

$x^i = (x^i_1, \dots, x^i_d)$

$d$ = nbr of features.

$i \in \{1, \dots n\}$ → nbr of obs.

Probability approach :

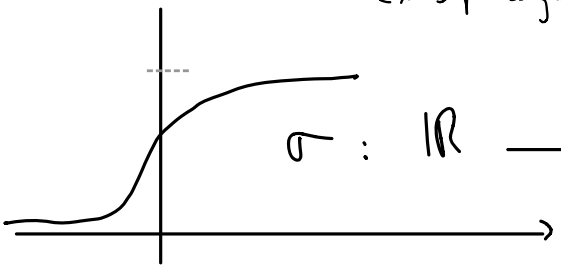
predict prob that $Y_i = 1$, if $X_i = \dots$

Learn. $\omega_1 \dots \omega_d$ such that

$$\sigma\left(\omega_1 x^i_1 + \dots + \omega_d x^i_d\right)$$

2x sepal length + ... − 1,11 × petal width.

$\sigma : \mathbb{R} \longrightarrow [0; 1]$



Iris.

$d = 4$

$n = 156$

Logistic regression : $\omega_1 x_1^i + \dots + \omega_d x_d^i = (\omega^T x^i)$

$$\sigma\left(\omega_1 x_1^i + \dots + \omega_d x_d^i\right) \longrightarrow$$

New point : $\sigma(x^{new}) = 0,7$

$$\longrightarrow \quad y^{new} = \begin{cases} \cancel{0} \\ 1 \end{cases}$$

$y \in \{0,1\}$

$\longrightarrow$ binary.

Predict° $y = \mathbb{1}_{\sigma(x^T\omega) > 0,5}$

$y = \{0,1,2\}$

how to find $\omega_1 \dots \omega_d$ : how to train.

Goal : learn weights s.t.

$$\mathbb{1}\left[\sigma\left(x_1^i w_1 + \dots + x_d^i w_d\right) > 0{,}5\right] \simeq y_i$$

Objective : $\times$ accuracy $\longrightarrow$ too hard for algo

$\downarrow$

logistic loss $\longrightarrow$ train the algo

More choices: later :)

Learning $\Leftrightarrow$ giving an obj to the machine and having it find the best estimator

$\mathbb{1}_{\sigma(x)>0,5} = \begin{cases} 1 \text{ if } \sigma(x)>0,5 \\ 0 \text{ else} \end{cases}$

$$\hat{Y}_i \simeq \begin{cases} 1 \text{ if } \sigma(\omega^T X_i) > 0,5 \\ 0 \text{ else} \end{cases} \longrightarrow \text{I want my predictor to be good!!}$$

accuracy

$$R_{n_{train}}(\omega) = \frac{1}{n_{train}} \sum_{i=1}^{n} \mathbb{1}_{Y_i \neq \mathbb{1}_{\sigma(\omega^T X_i)>0,5}}$$

$\mathbb{1}_{\sigma(\omega^T X_i)>0,5}$ : $\hat{Y}_i$

min this risk!

$>0 \Leftrightarrow Y_i = 1$

$<0 \Leftrightarrow Y_i = 0$

logistic loss

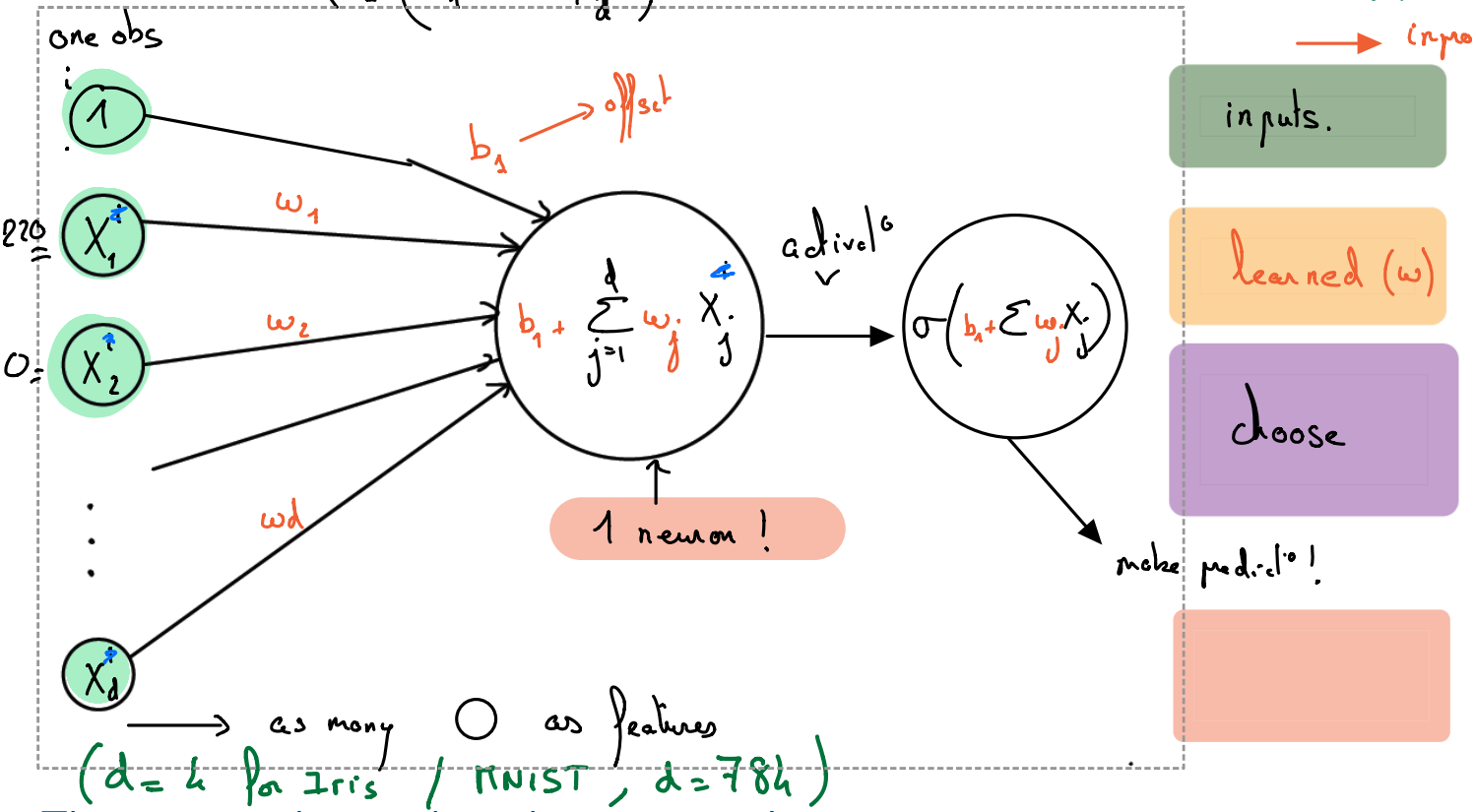
$$R^{log}_n(\omega) \simeq \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-(Y_i - 1/2)\left(\sigma(\omega^T X_i) - 1/2\right)}\right)$$

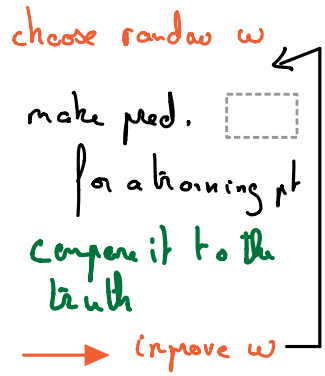
$\sigma(\omega^T X_i) - 1/2 > 0 \Leftrightarrow \hat{Y}_i = 1$

$\sigma(\omega^T X_i) - 1/2 < 0 \Leftrightarrow \hat{Y}_i = 0$

# 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 !

# Learning Process

init

choose random w:
each weight is different

make pred.
for a learning pt
compare it to the truth
→ improve w

optim algo

Some Learning process

Logistic regression

D L.

make pred.
for a learning pt
compare it to the truth
→ improve w

} gradient descent

Different weights and predicte boxes

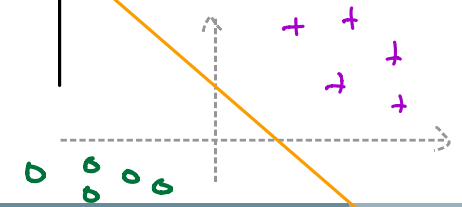
d weights (+1)

much more weights

L R

MLP.

split.



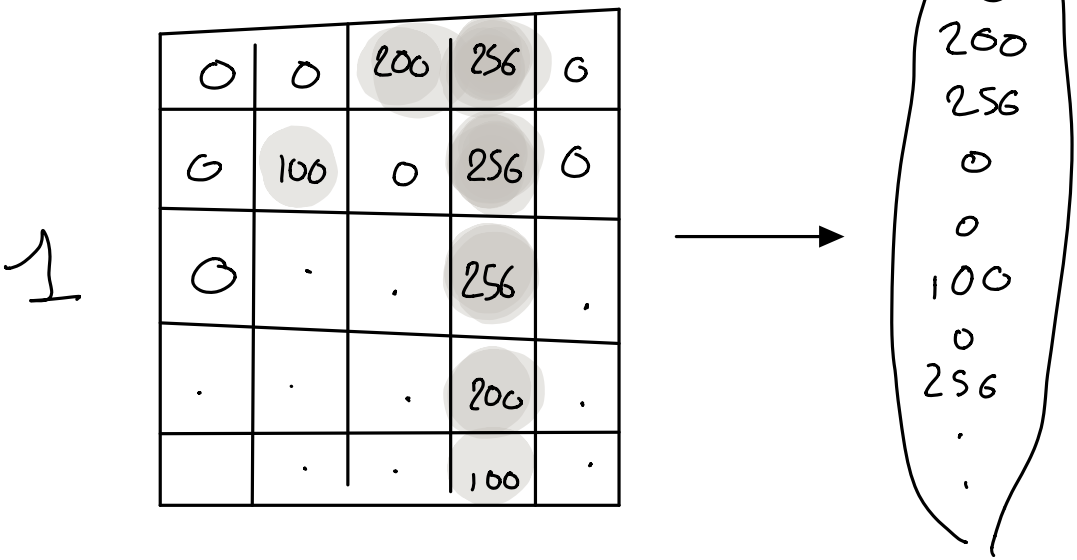
not linear.
Separate!

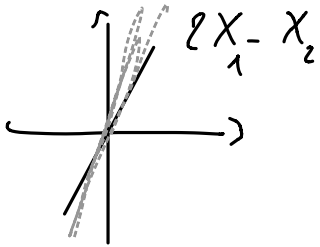
$X^i$ is an image.

$$X^i = (X^i_1, \dots, X^i_d)$$

↳ a pixel value

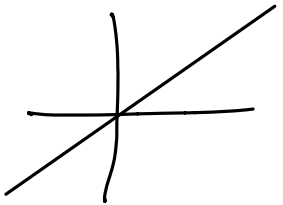
$d = 784$



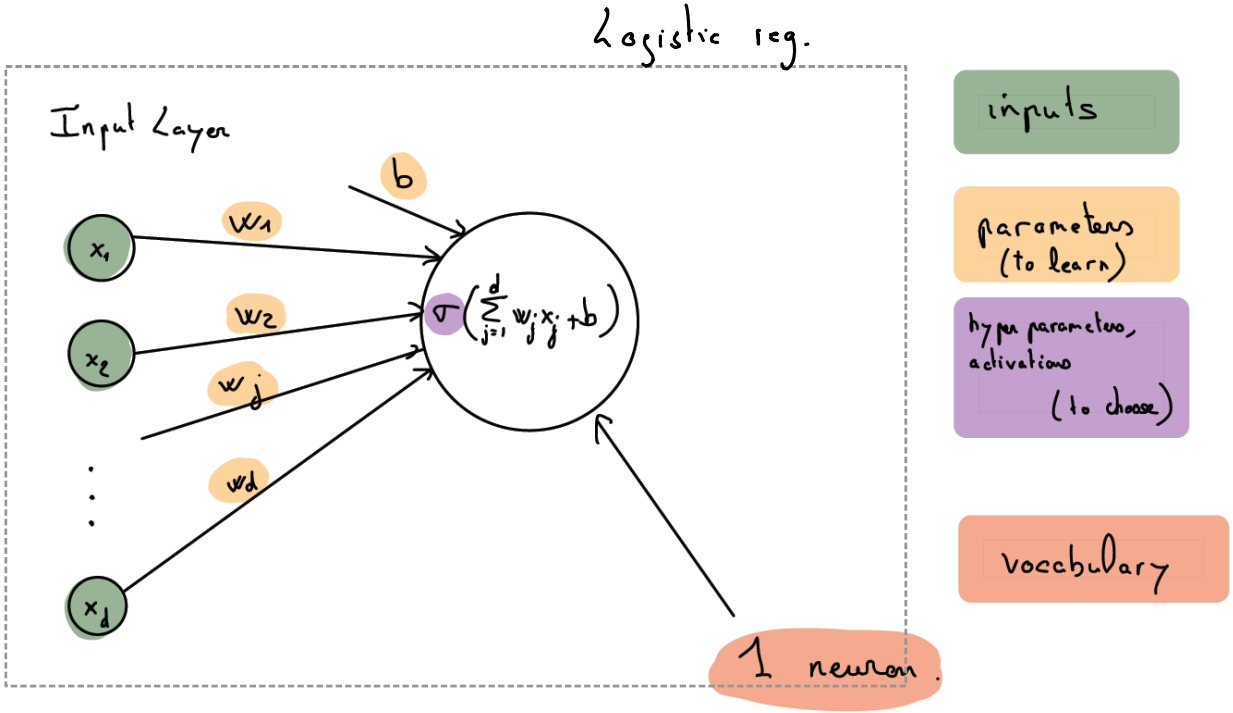

$X_1 - X_2 = 0$

$\Leftrightarrow X_1 = X_2$



# 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

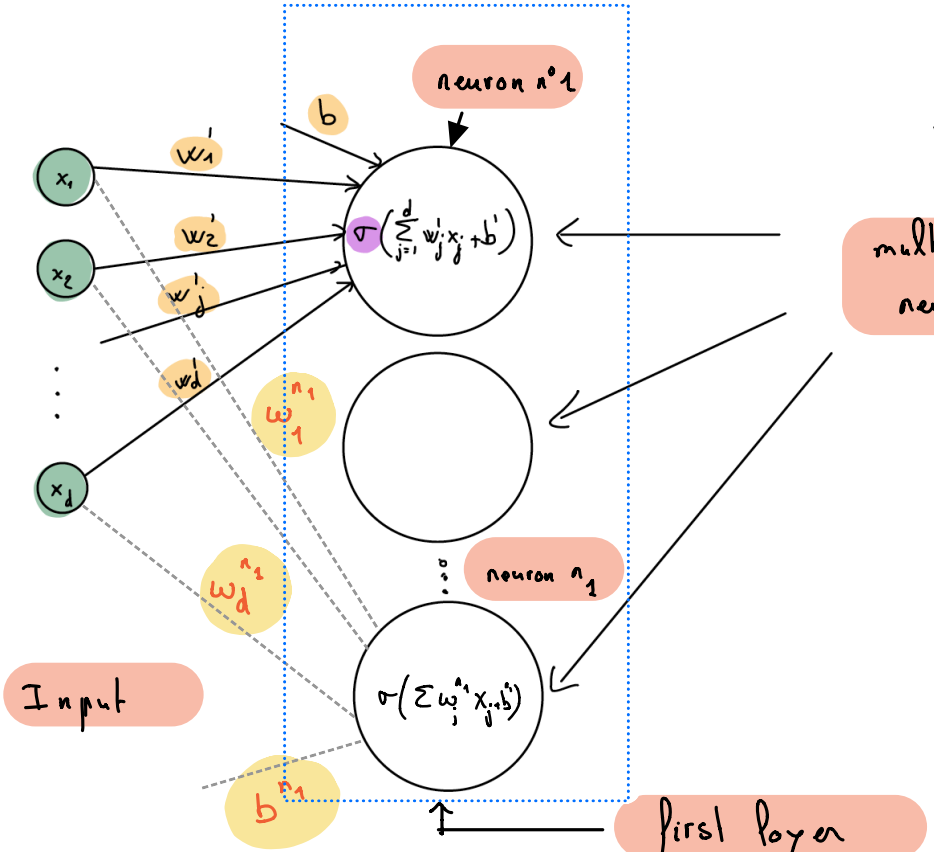



This is a neural network, with one neuron !

# From logistic regression to Multi Layer Perceptron 4

Extend to

- Multiple Neurons
- Multiple Layers




learnable parameters -

neuron n°1 ⟶ d weights + 1 bias.

⋮

neuron n° $n_1$ ⟶ d weights + 1 bias.

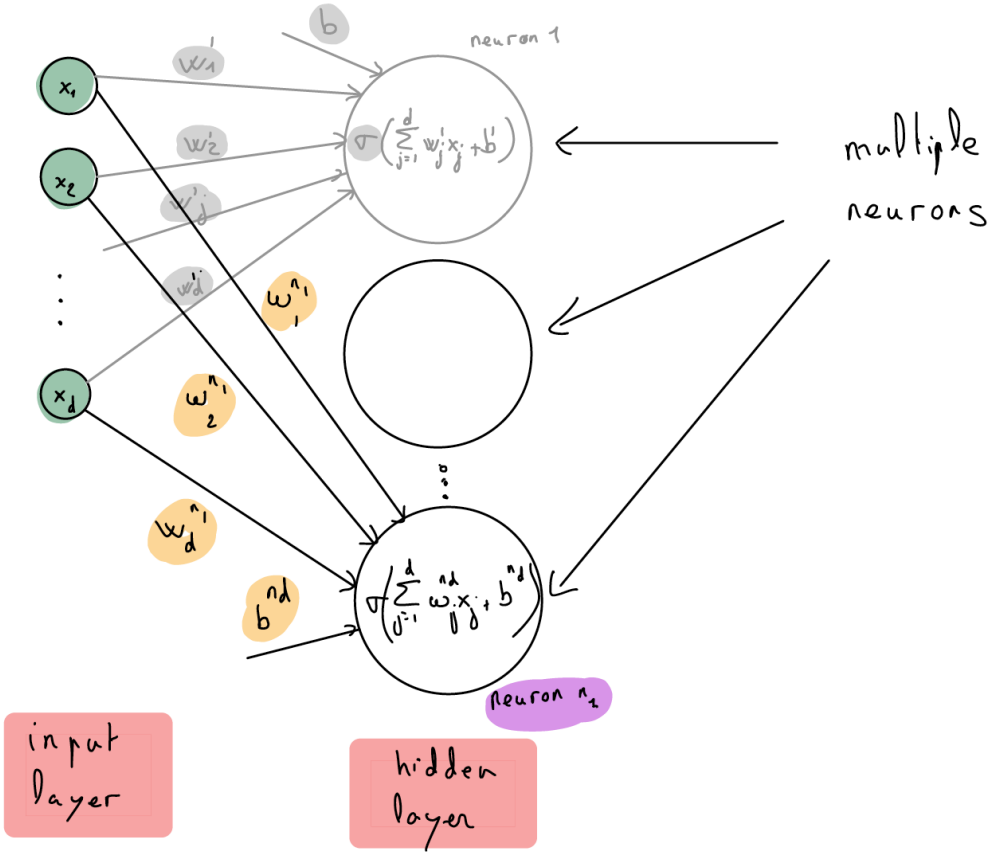
# parameters to learn

$n_1 \times (d+1)$ !

# From logistic regression to Multi Layer Perceptron 4

Extend to

- Multiple Neurons
- Multiple Layers

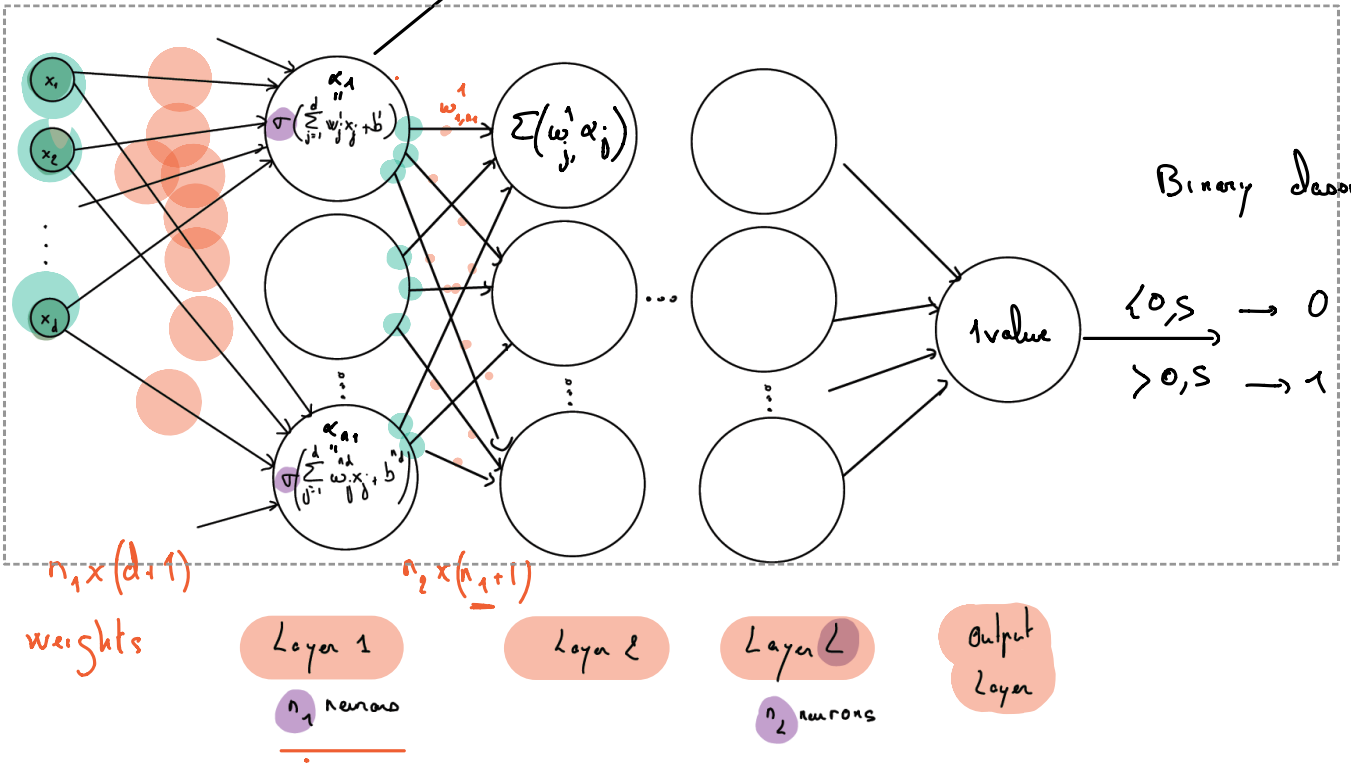


# From logistic regression to Multi Layer Perceptron 5

Extend to

- Multiple Neurons
- Multiple Layers

$$W_3^T \, \sigma\Big( W_2 \, \underbrace{\sigma\big(W_1 X + b\big)}_{\text{new vector}} + b \Big) \dots$$



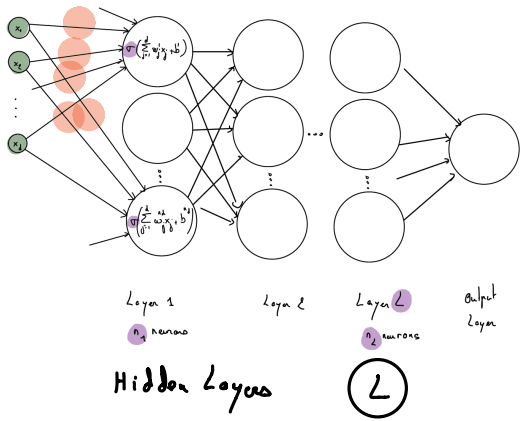
# From logistic regression to Multi Layer Perceptron 5

find $f$ defined by its weights

Neural network:

- Goal $\min_{f \in \mathcal{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$
- Class of non linear functions.
  $\mathcal{F}_{\text{MLP}} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(\ldots(b_1 + W_1 x))),$
  $W \in \mathbb{R}^{\cdots}, b \in \mathbb{R}^{\cdots}\}$

huge class of functions.




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

What needs to be learned

Weights / biases

plenty

What needs to be chosen

Layers nb $L$.

Neurons per layer $(n_h)$ $h = 1..L$

Activate funct° (RELU, sigmoid)

* Depth: Nb layers. $L \geq 10$.
* Width: $n_1$ increases

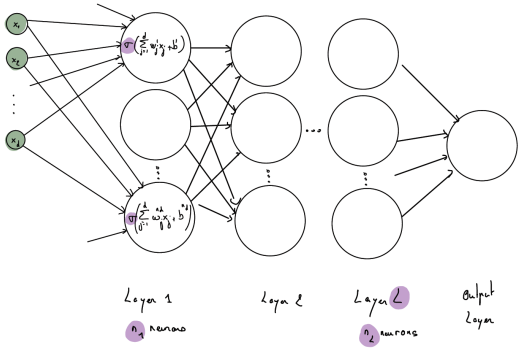
Vocabulary

* neuron
* activate funct°: transformate non linear that I apply to ...
* hidden layer
* fully connected layer

# From logistic regression to Multi Layer Perceptron 5

Neural network:

- Goal $\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$
- Class of non linear functions.
  $\mathscr{F}_{\text{MLP}} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(...(b_1 + W_1 x))),$
  $W \in \mathbb{R}^{\cdots}, b \in \mathbb{R}^{\cdots}\}$




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

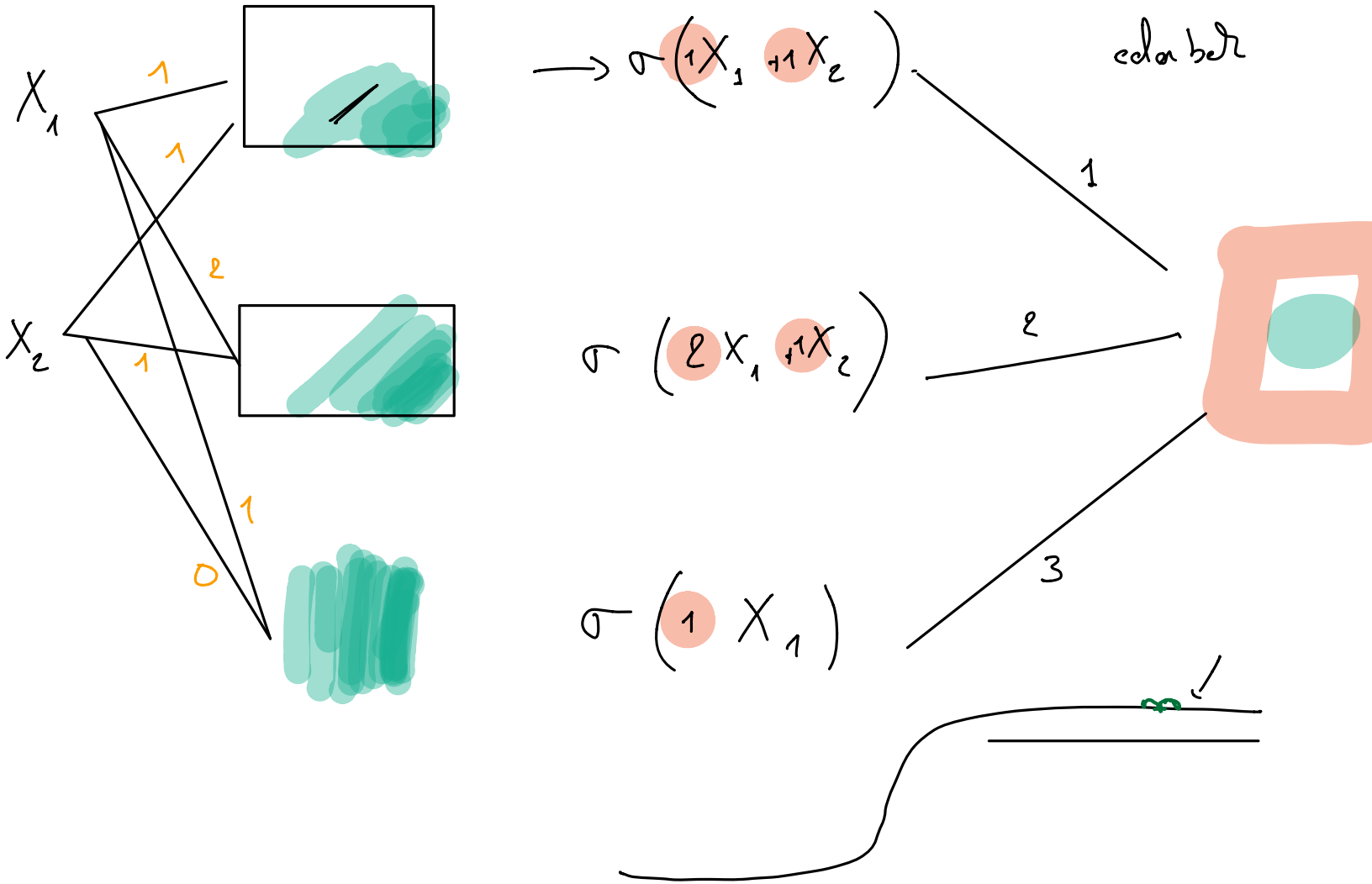
- Activation $\sigma$
- Number of layers $L$
- Number of neurons per hidden layer $n_i, i = 1, \ldots, L$.

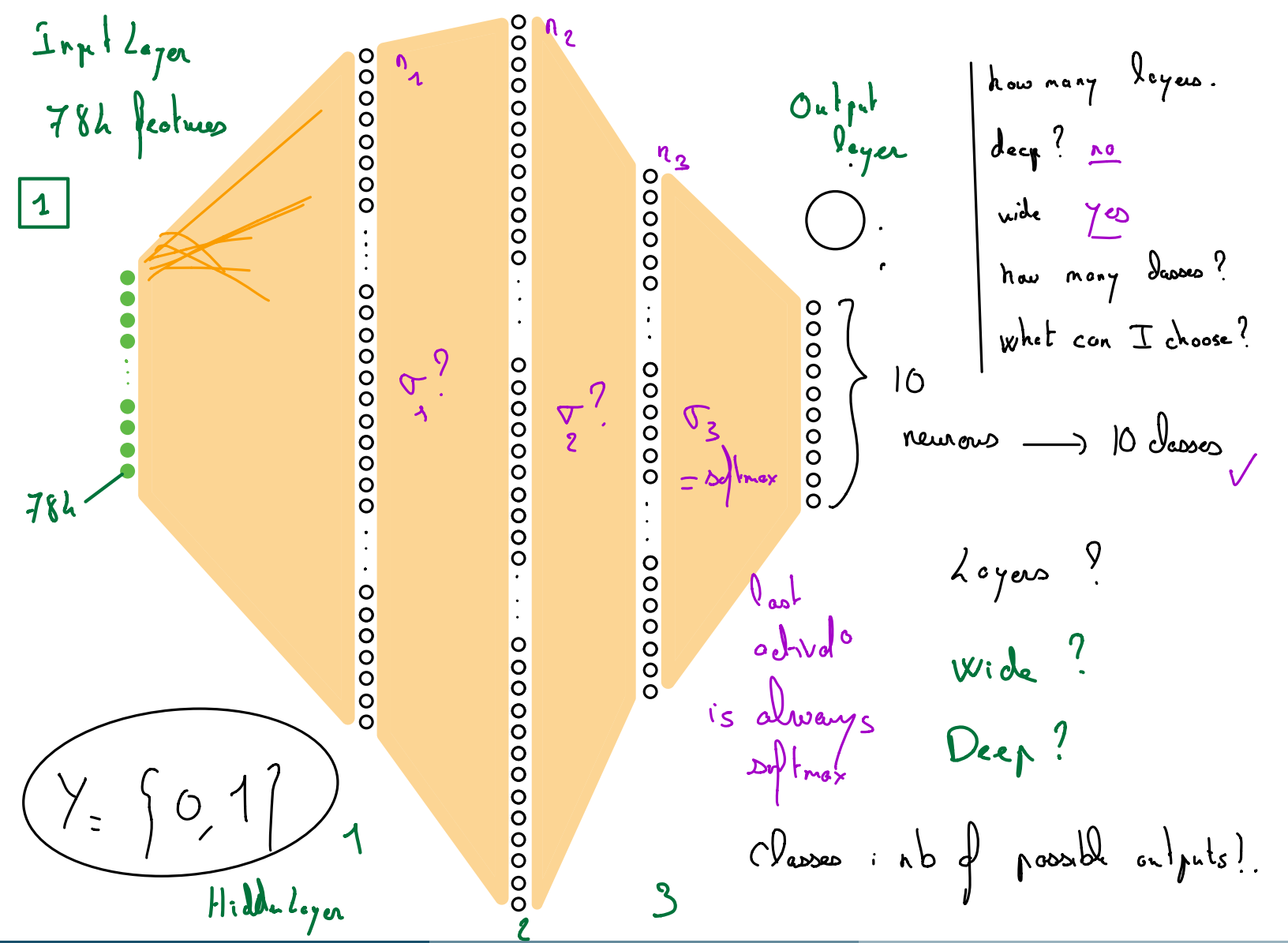
Vocabulary

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

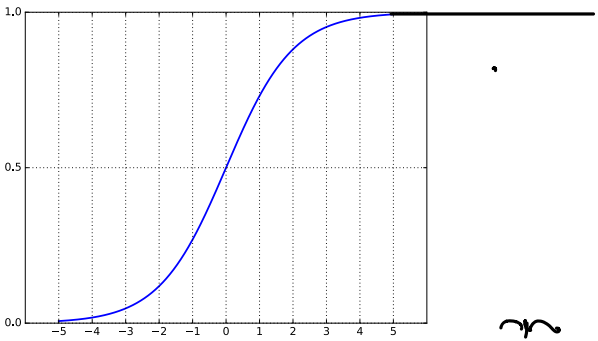
Neural network play ground :

* compare non linearly separable to lin. separable data
* what is logistic regression doing ? (use 1 hidden layer, 1 neuron, sigmoid act)
* compare to 6 neurons. What happens ?
* add more layers : what happens for sigmoid ?
* change to RELU act.vd°.


$X_1$
$X_2$
1
1
2
1
0
$\sigma(1X_1 + 1X_2)$
$\sigma(2X_1 + 1X_2)$
$\sigma(1X_1)$
1
2
3
ecart bet

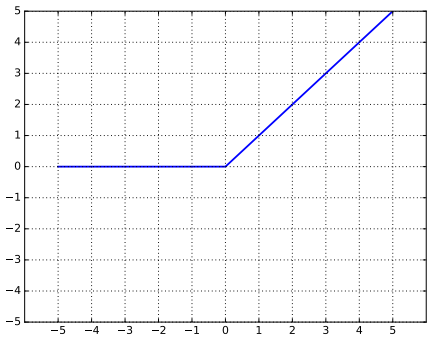

Input Layer
784 features
1
784
$n_1$
$n_2$
$n_3$
$\sigma_1$ ?
$\sigma_2$ ?
$\sigma_3$ = softmax
Output layer
10
neurons ⟶ 10 classes ✓
how many layers.
deep ? no
wide yes
how many classes ?
what can I choose ?
Layers ?
Wide ?
Deep ?
last activation is always softmax
$Y = \{0, 1\}$
1
Hidden Layer
3
2
classes : nb of possible outputs !.

# 2 possible activations: Sigmoid and Rectified Linear Unit




## Sigmoid function

- $x \mapsto \frac{\exp(x)}{1+\exp(x)}$
- Problems:
  1. Saturated function: gradient killer -> need for rescaling data




## Rectified Linear Unit (ReLU)

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

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

replaces sigmoid

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.

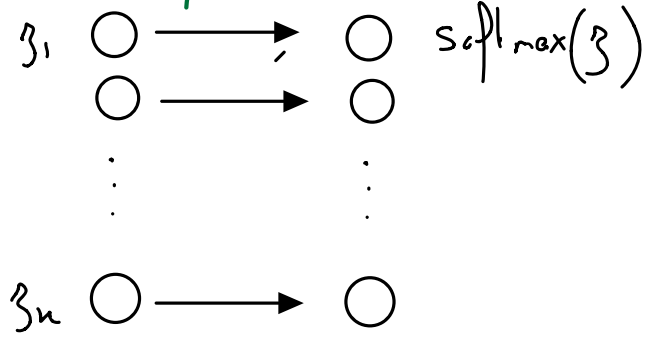
Softmax output unit, used to predict $\{1, \dots, K\}$:

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

how to predict probabilities for K classes

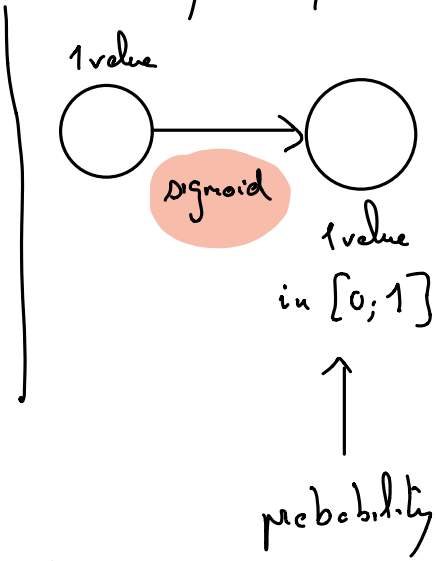
K = 3 iris !

K = 10 MNIST !




↳ probabilities of each class !

Binary classifier




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 → classical computing unit
   - GPU → graphical PU.
   - TPU → 100x faster


2. Software (Python packages):
   - Pytorch/Tensorflow
   - → Keras : TF high level API. Ideal for applications.

# Keras - A few lines ![1]

What do we need to specify?

architecture : { number and sizes of layers

activat° .

1 hidden layer
1 output layer

What do we need to do next?

→ start the learning process

learning algo

→ kind of GD.

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
# get data
(train_images, train_labels), (test_images,
                               test_labels) = keras.datasets.mnist.load_data()

# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# train model
model.fit(train_images, train_labels, epochs=5)

# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)

# make predictions
predictions = model.predict(test_images)

```

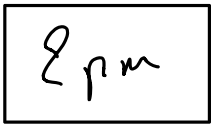
→ (784 × 1)

× sigmoid each

× 128 = $n_1$

[1] Notebook Python

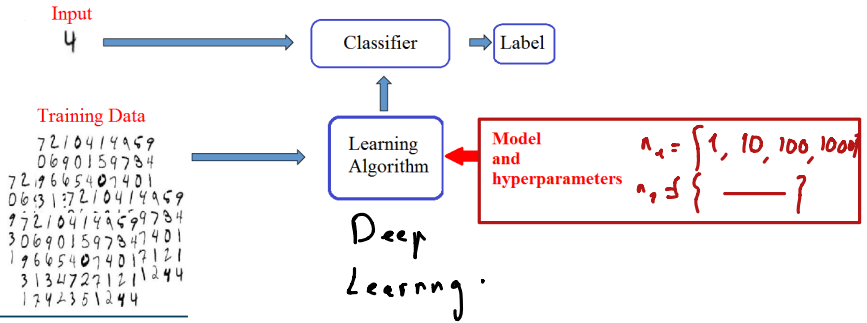
# Keras - A few lines ![1]



## What do we need to specify?

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
# get data
(train_images, train_labels), (test_images,
                               test_labels) = keras.datasets.mnist.load_data()

# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# train model
model.fit(train_images, train_labels, epochs=5)

# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)

# make predictions
predictions = model.predict(test_images)

```

## What do we need to do next?




[1] Notebook Python

# Intuition

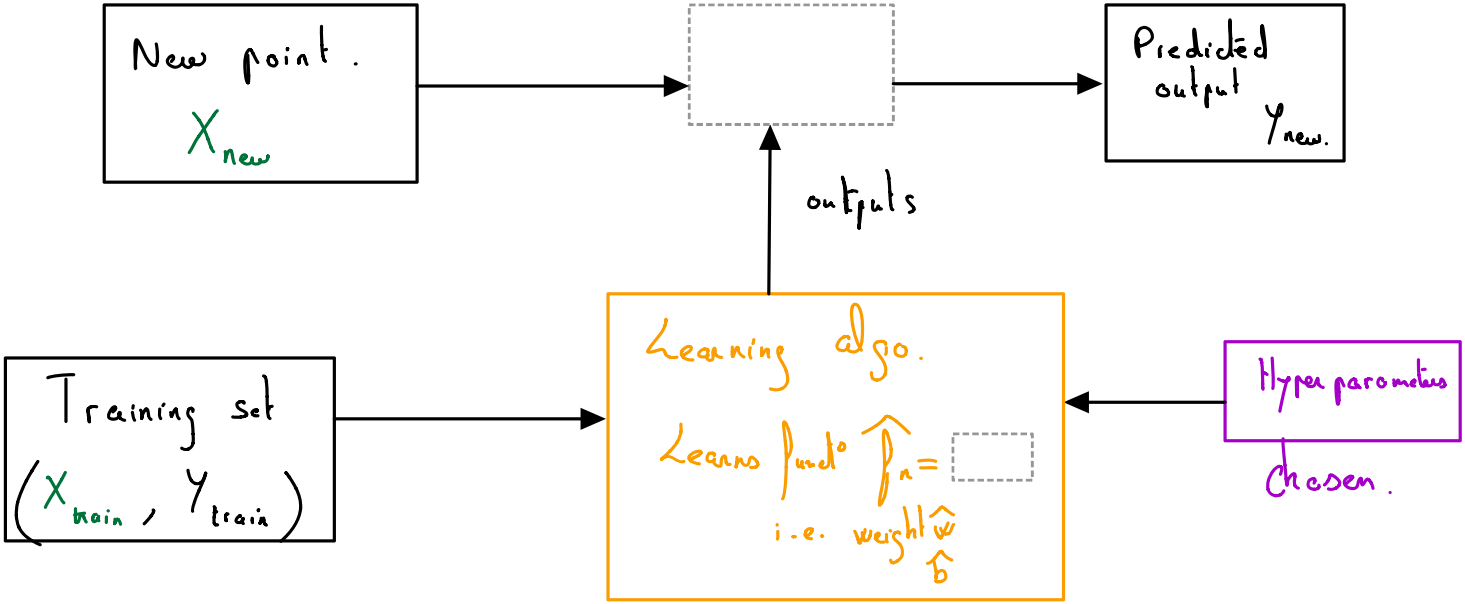
Up to that point, we have seen :

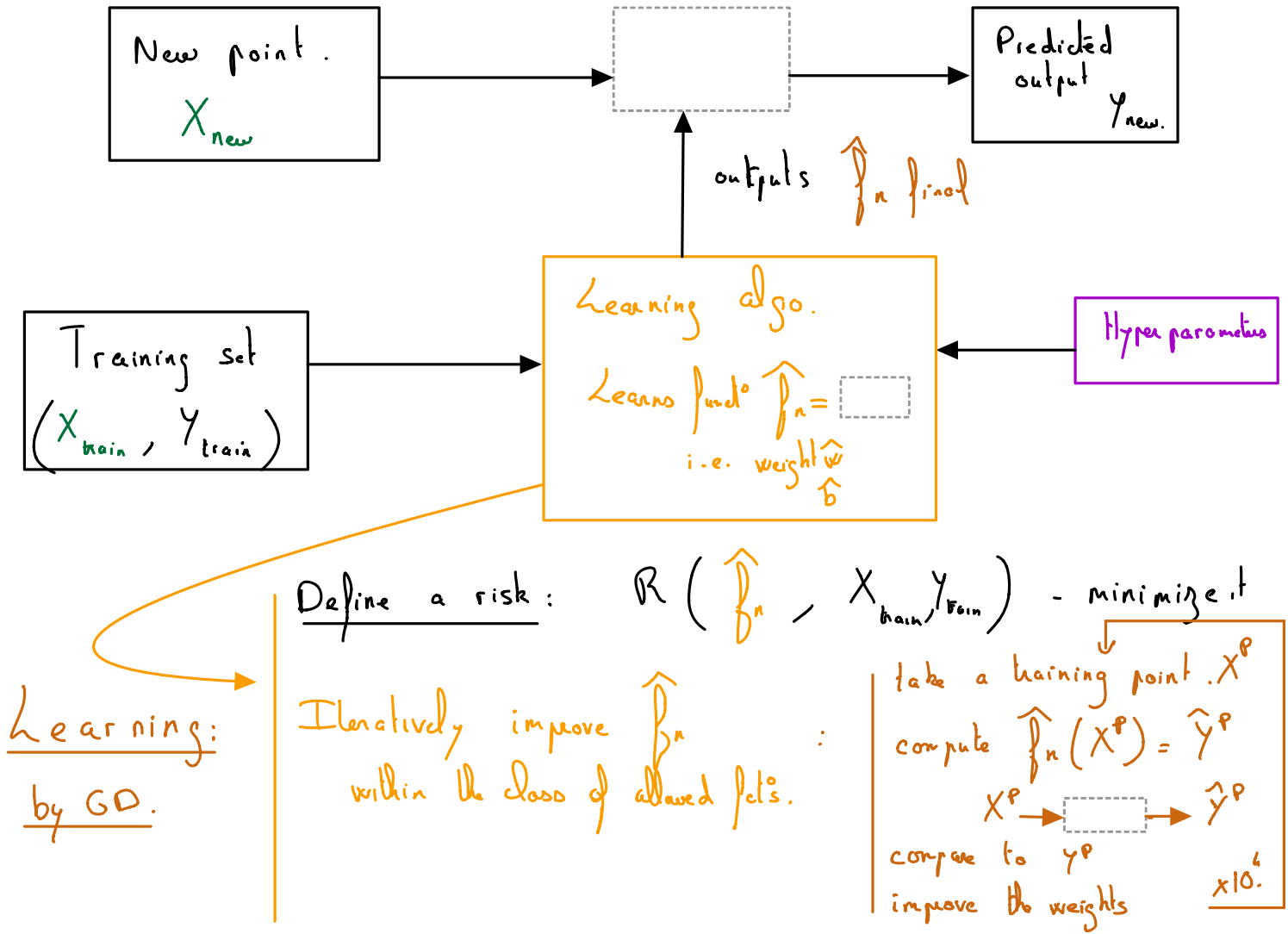
- What a Neural Network was.
- How to implement it in Python.

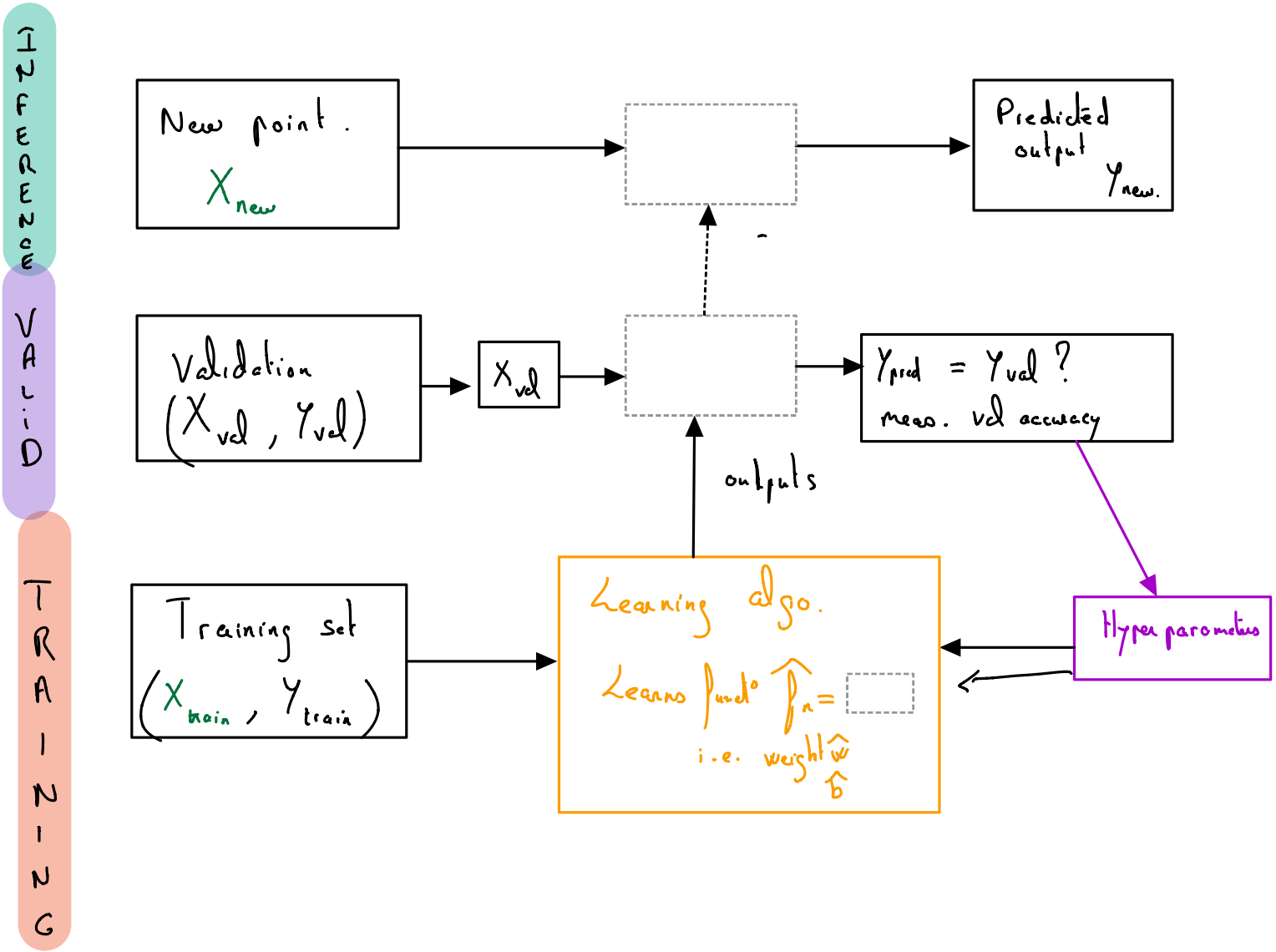
Next Question: why and when does it work ? → 2 theorems (representation, optimization)

→ CNNs

→ Applications.


New point.
$X_{new}$
Predicted output
$Y_{new}$.
outputs
Learning algo.
Learns fonct° $\hat{f}_n =$
i.e. weight $\hat{w}$ $\hat{b}$
Training set
$(X_{train}, Y_{train})$
Hyperparametres
Chosen.


New point.
$X_{new}$
Predicted output
$y_{new}$.
outputs $\hat{f}_n$ final
Learning algo.
Learns funct° $\hat{f}_n =$
i.e. weight $\hat{w}$
$\hat{b}$
Training set
$(X_{train}, y_{train})$
Hyperparameters
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 fct's.
take a training point $X^p$
compute $\hat{f}_n(X^p) = \hat{y}^p$
$X^p \rightarrow \hat{y}^p$
compare to $y^p$
improve the weights
$\times 10^6$.


INFERENCE
VALID
TRAINING
New point.
$X_{new}$
Predicted output
$Y_{new}$.
Validation
$(X_{val}, Y_{val})$
$X_{val}$
$Y_{pred} = Y_{val}$ ?
meas. val accuracy
outputs
Training set
$(X_{train}, Y_{train})$
Learning algo.
Learns fonct° $\hat{f}_n =$
i.e. weight $\hat{w}$
$\hat{b}$
Hyper parametres

# 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

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

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.

$\looparrowright$ Very powerful in terms of approximations.
Not very surprising: non linear function with millions of parameters !

# Optimization

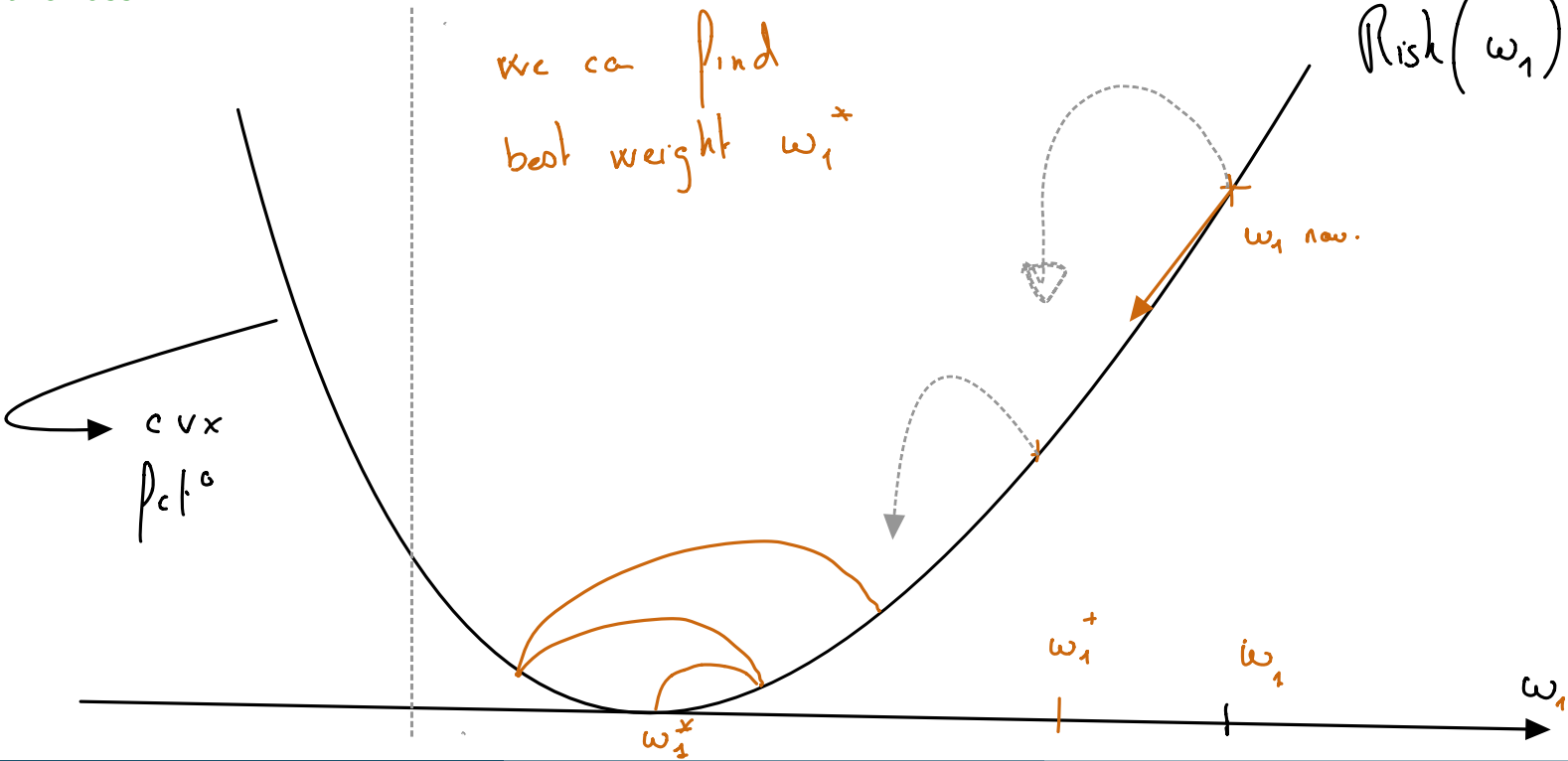
Risk ( $\omega$ ) logistic

$\longrightarrow$ cvx

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

$(f \circ g)' = (f' \circ g) \times (g')$

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.

$f$ : 

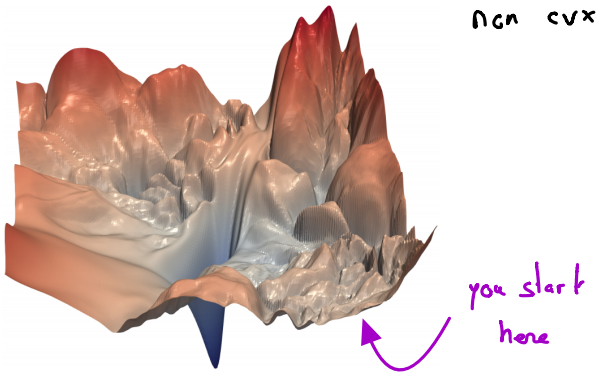
computing the gradient.

chain rule

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 is too bad local minima. (specialized optim algos: Adam, AdaDelta, etc.)

non cvx is hard.




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.

$\hookrightarrow$ **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

# CNN

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

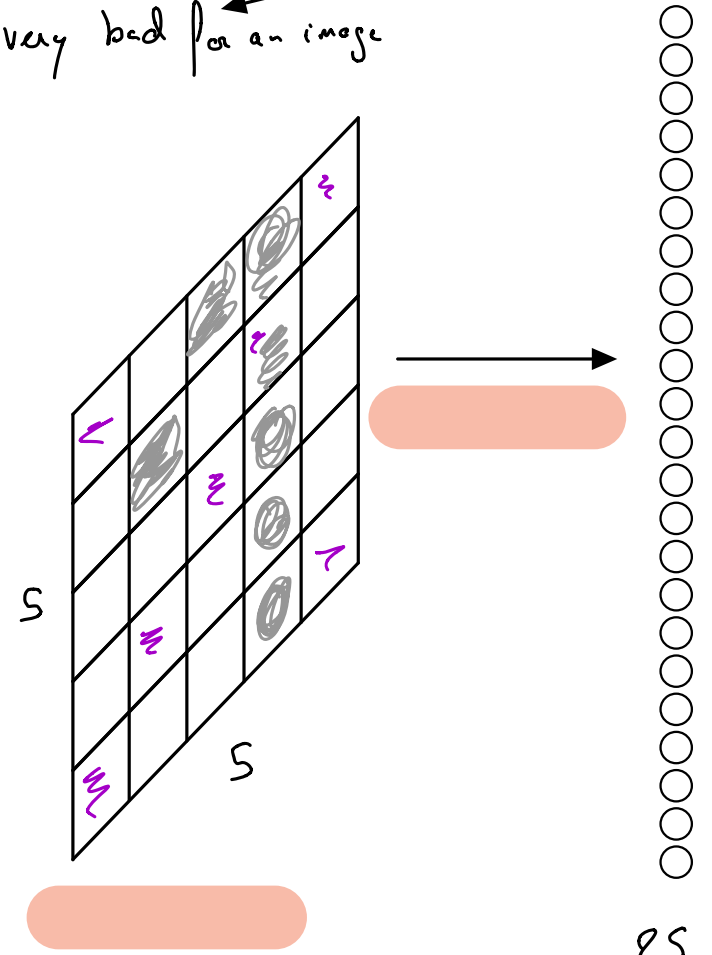

Figure: CIFAR dataset

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 vs Dense = Fully connected

very bad for an image




Iris.

$$X_{train} = \underset{\times 1}{P_w}, \underset{\times 2}{P_l}, \underset{\times -3}{S_w}, \underset{\times 0}{S_l}$$

vs

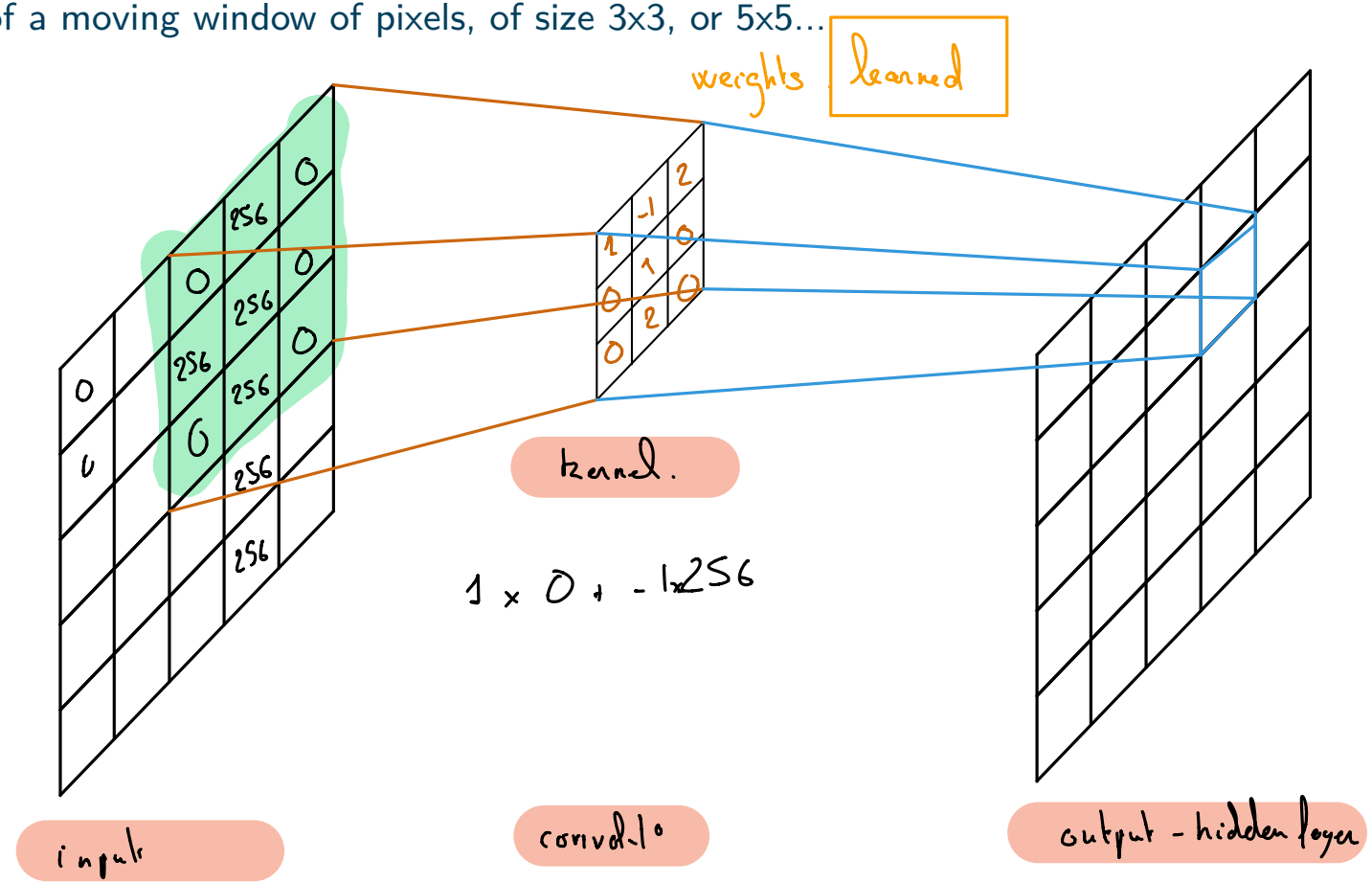
$$X_{train} = \underset{\times -3}{S_w}, \underset{\times 1}{P_w}, \underset{\times 2}{P_l}, \underset{\times 0}{S_l}$$

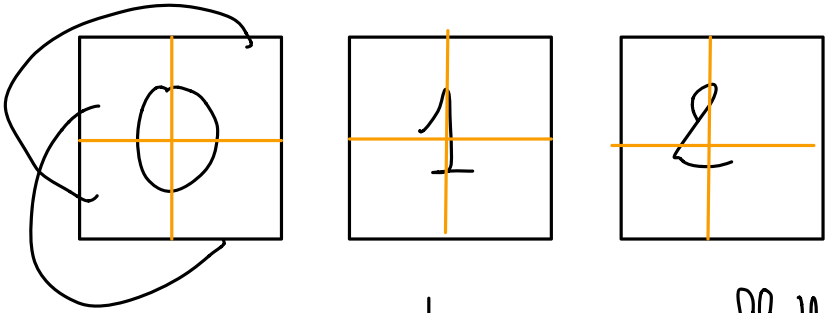
$$\sum \omega_i X_i$$

# Convolutional Layer

0 0

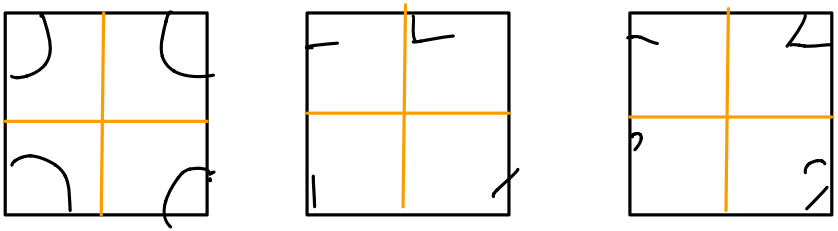
Instead of making a linear combination of all the pixels, we consider a weighted average of a moving window of pixels, of size 3x3, or 5x5...





$X_1$

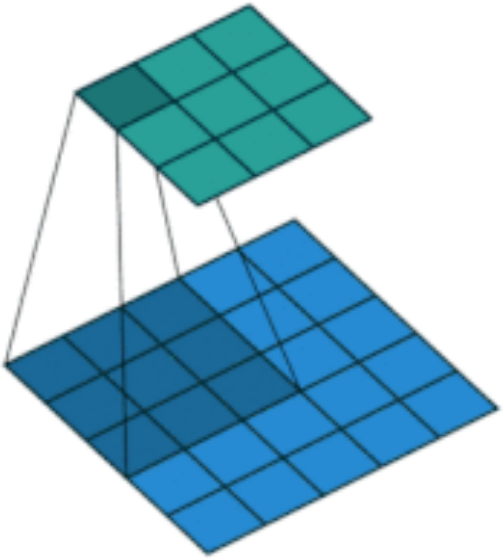
using a flatten first layer means that I would learn the same weights if I had.



$Y_1 = 0$

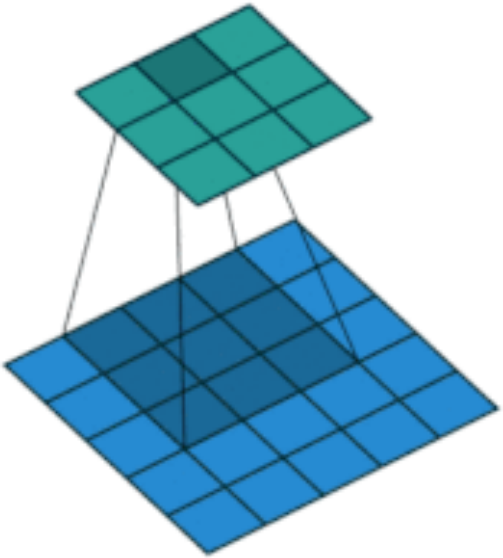
# Convolutional Layer





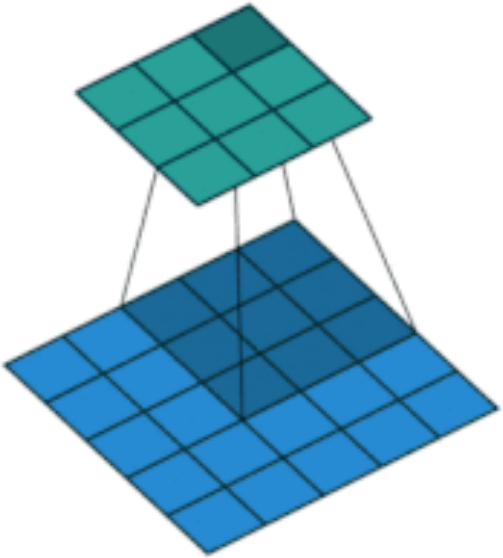
# Convolutional Layer





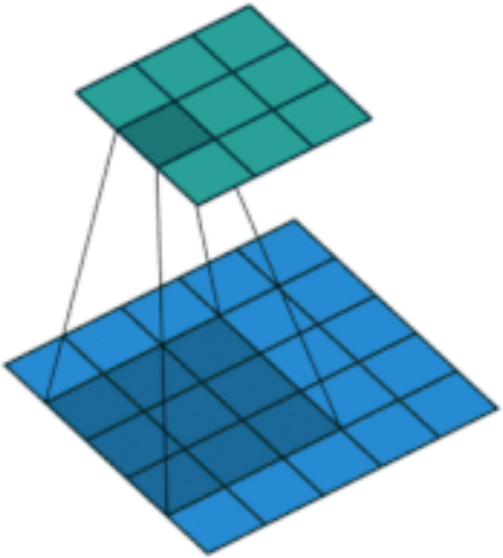
# Convolutional Layer





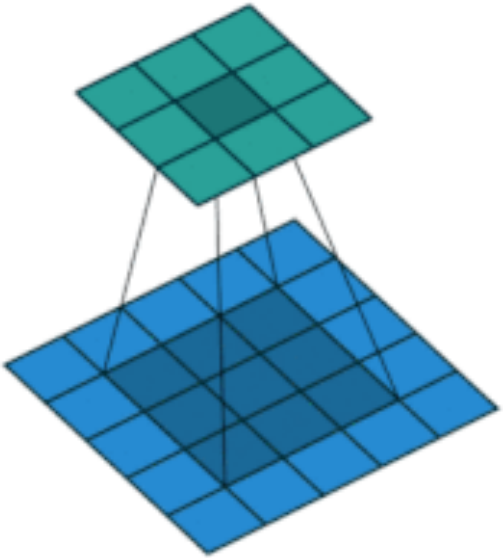
# Convolutional Layer





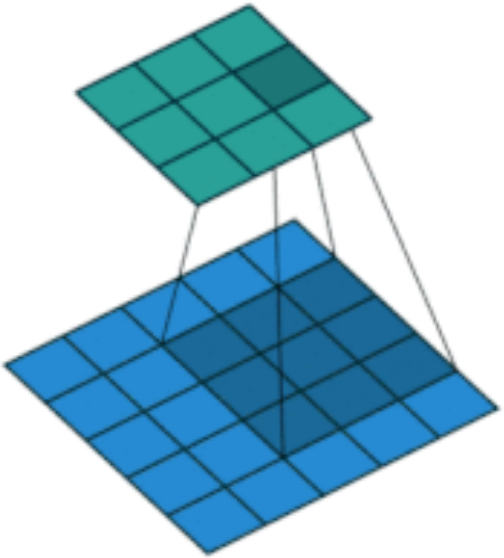
# Convolutional Layer





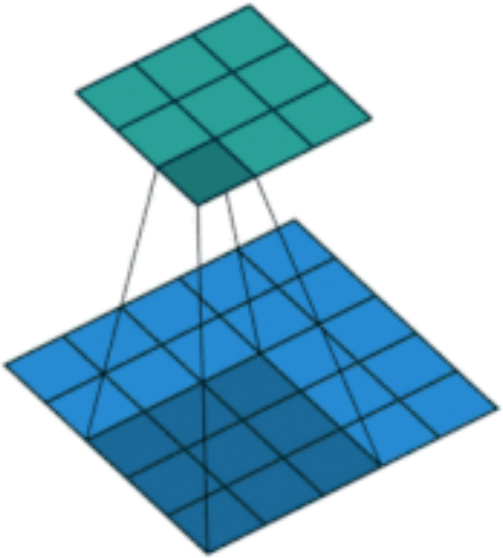
# Convolutional Layer





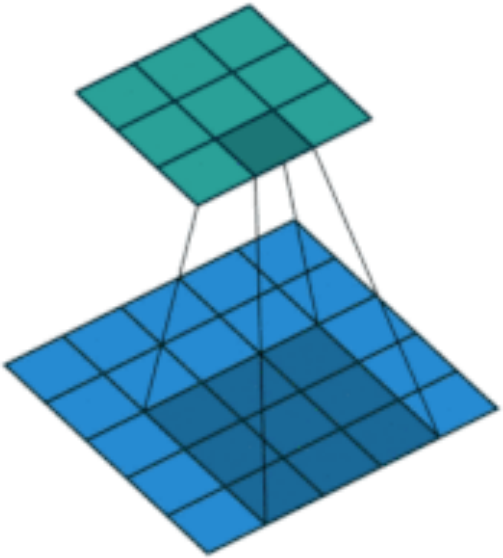
# Convolutional Layer





# Convolutional Layer

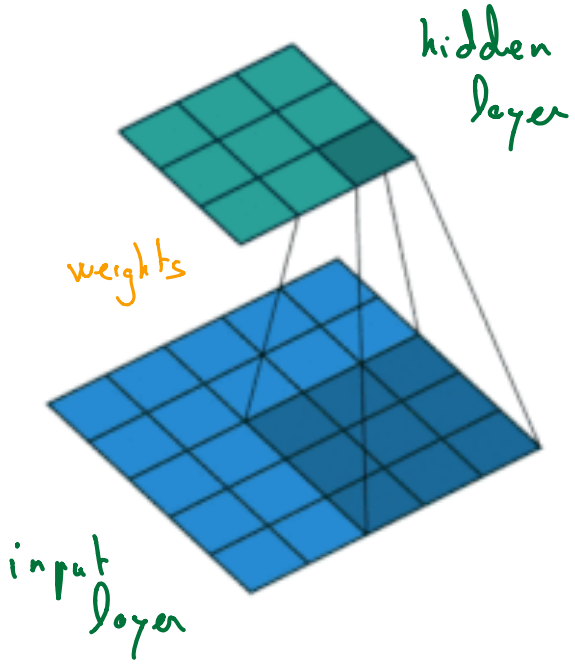




# Convolutional Layer

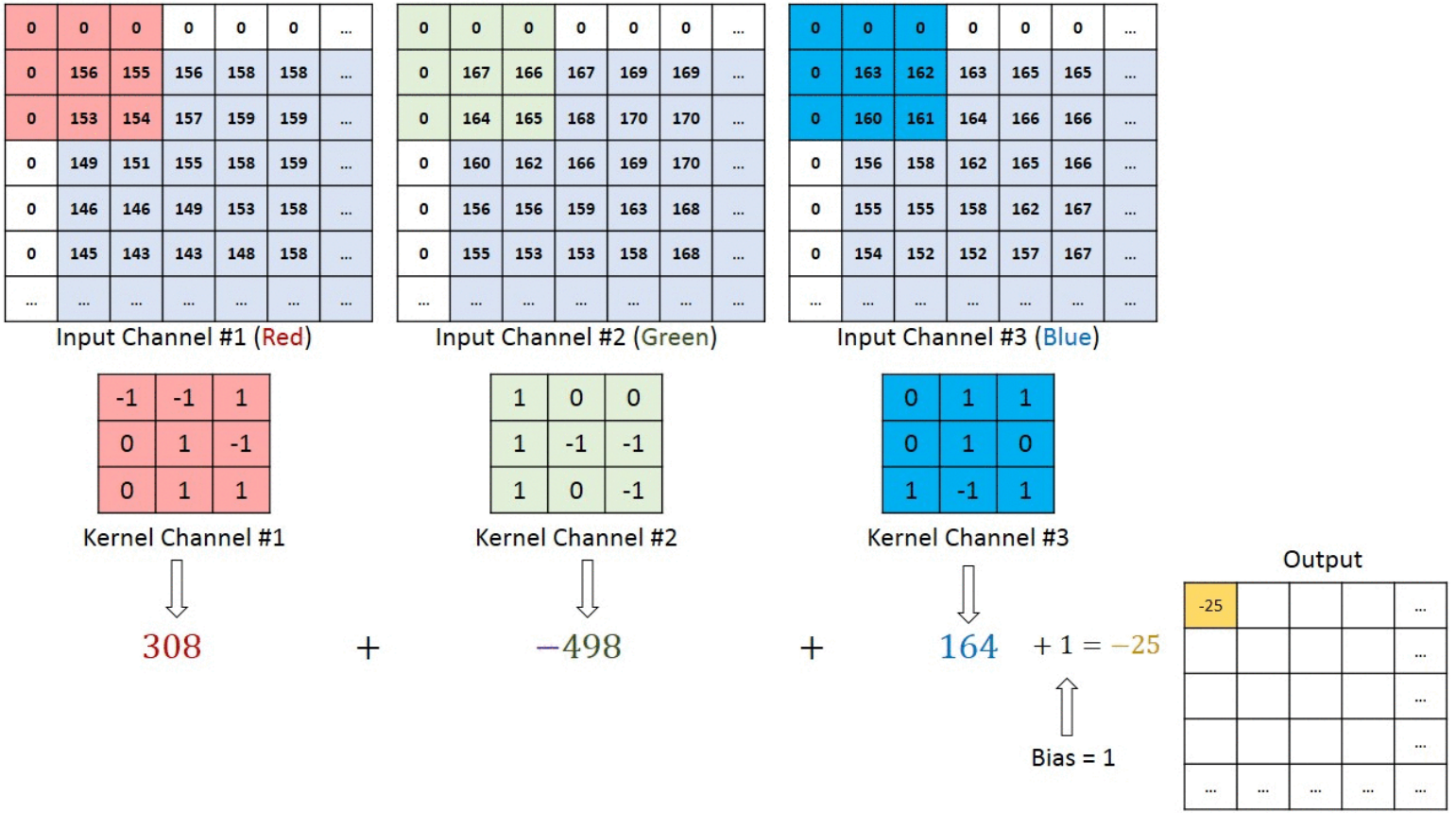
| | | | | |
|---|---|---|---|---|
| 3 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 3 | 1 |
| 3 | 1 | $2_0$ | $2_1$ | $3_2$ |
| 2 | 0 | $0_2$ | $2_2$ | $2_0$ |
| 2 | 0 | $0_0$ | $0_1$ | $1_2$ |

| | | |
|---|---|---|
| 12.0 | 12.0 | 17.0 |
| 10.0 | 17.0 | 19.0 |
| 9.0 | 6.0 | 14.0 |

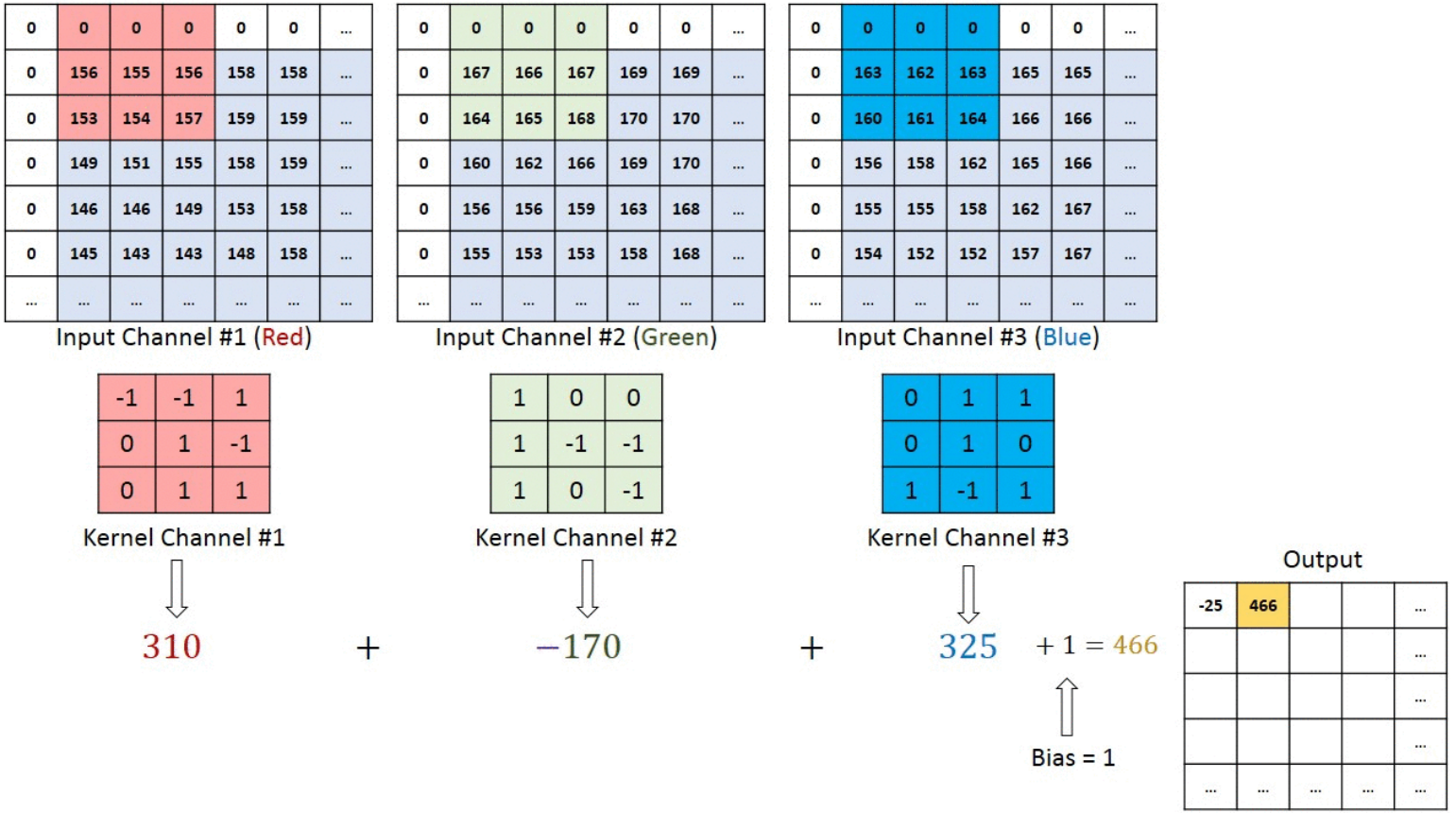


# Convolutional Layer

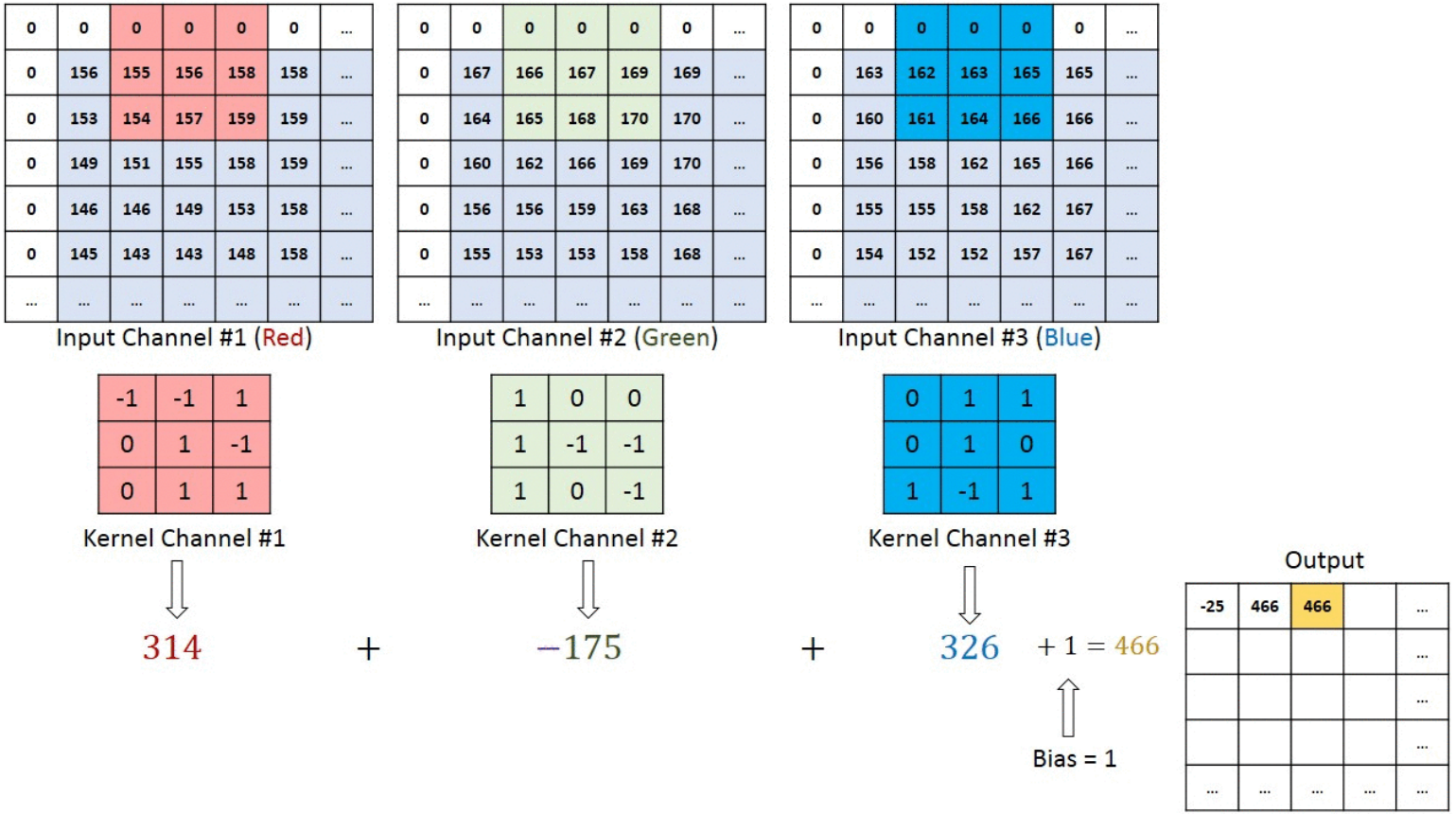
# Convolutional Layer



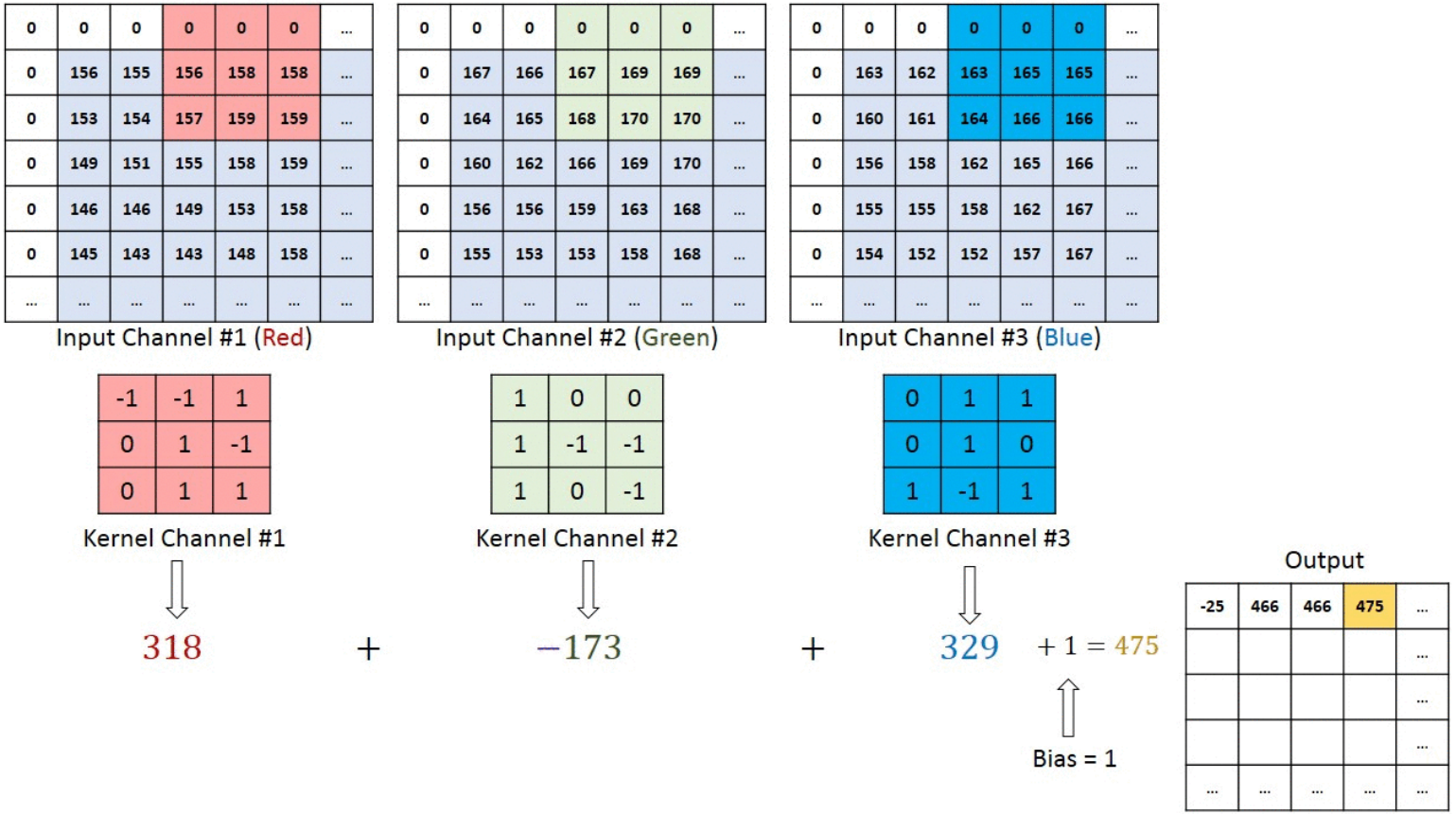
# Convolutional Layer



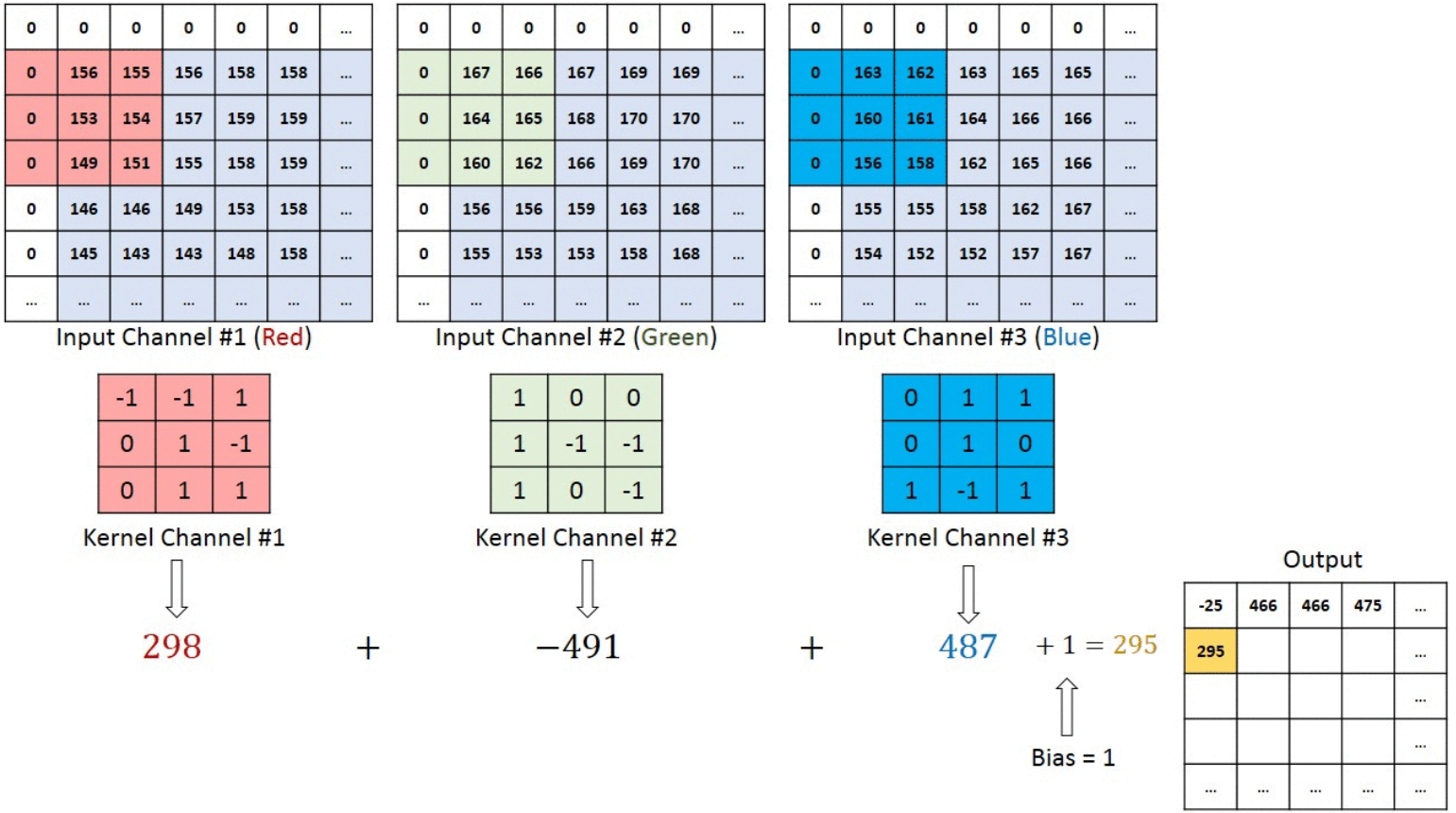
# Convolutional Layer



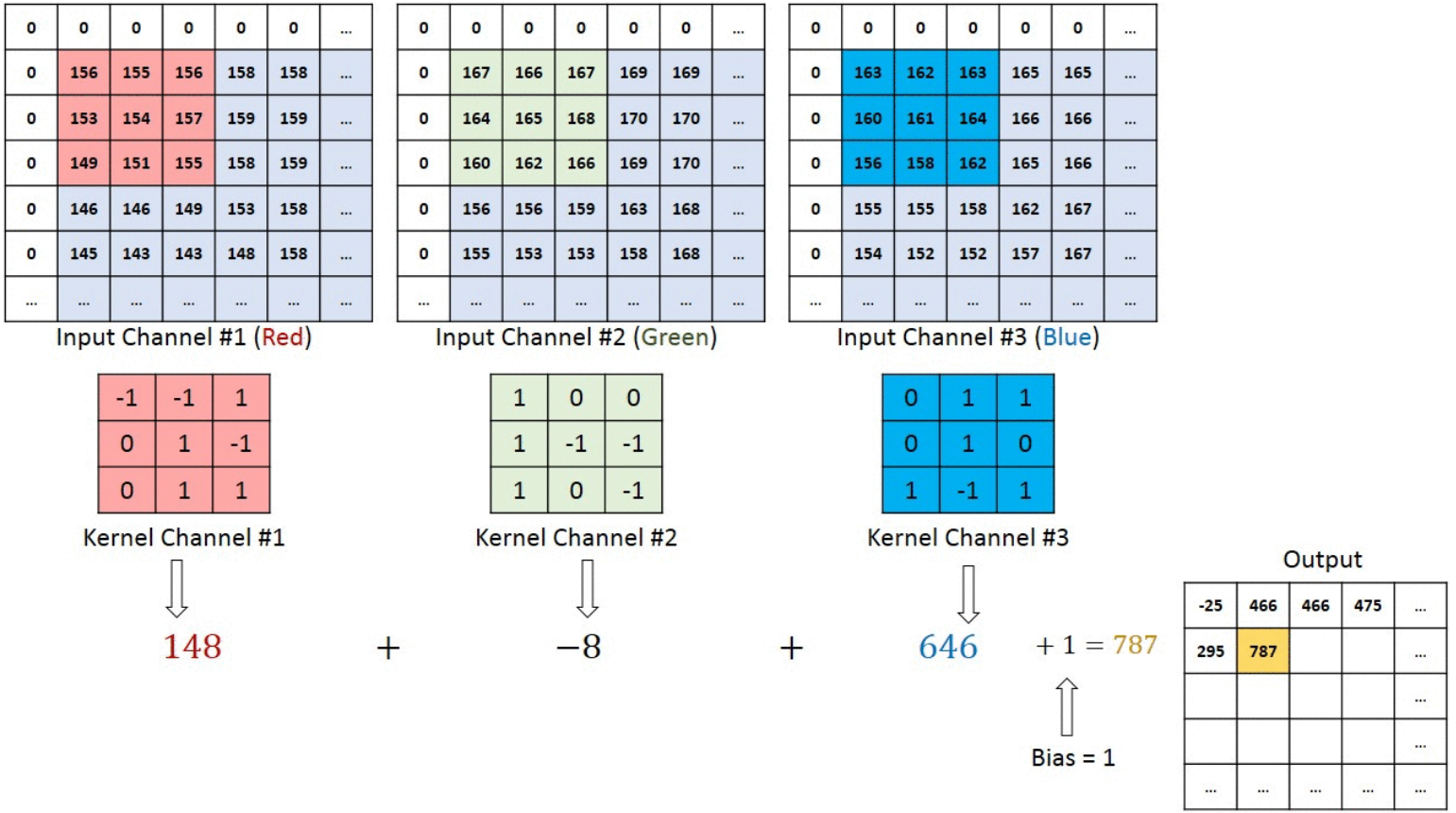
# Convolutional Layer



# Convolutional Layer



# Convolutional Layer



# 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:**

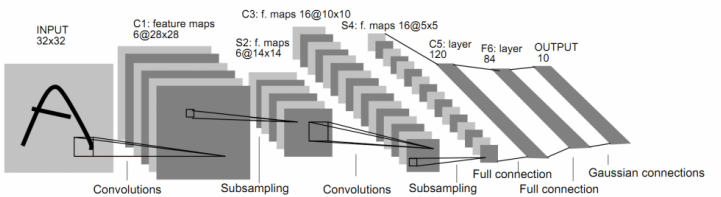
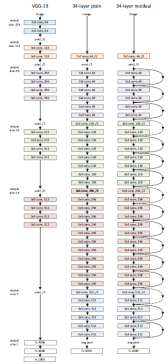



Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

8 layers, 60k parameters.

**Resnet 2016**



~ 40 Layers, 140M parameters

# Outline

# Outline

# Image classification - CNN Tree

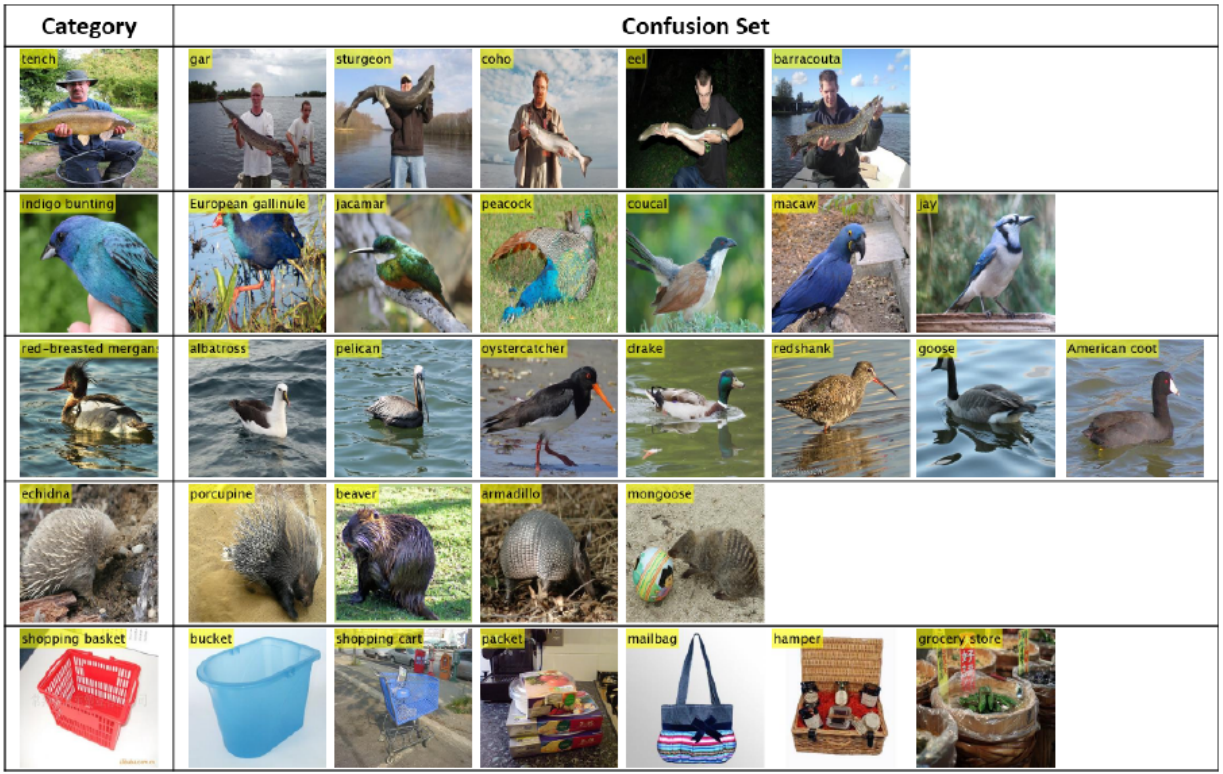



Figure: Confusion set outputs by AlexNet softmax prediction on validation set of ILSVRC 2015.

# Image classification - CNN Tree

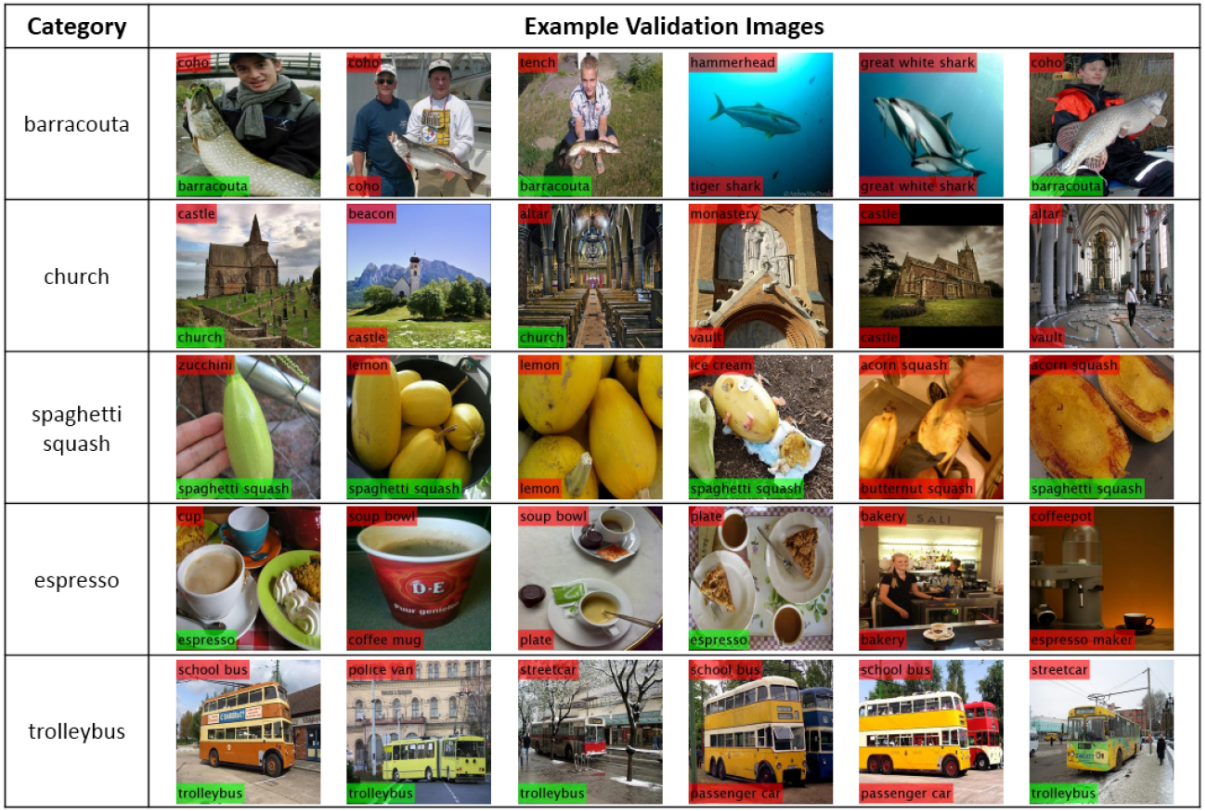
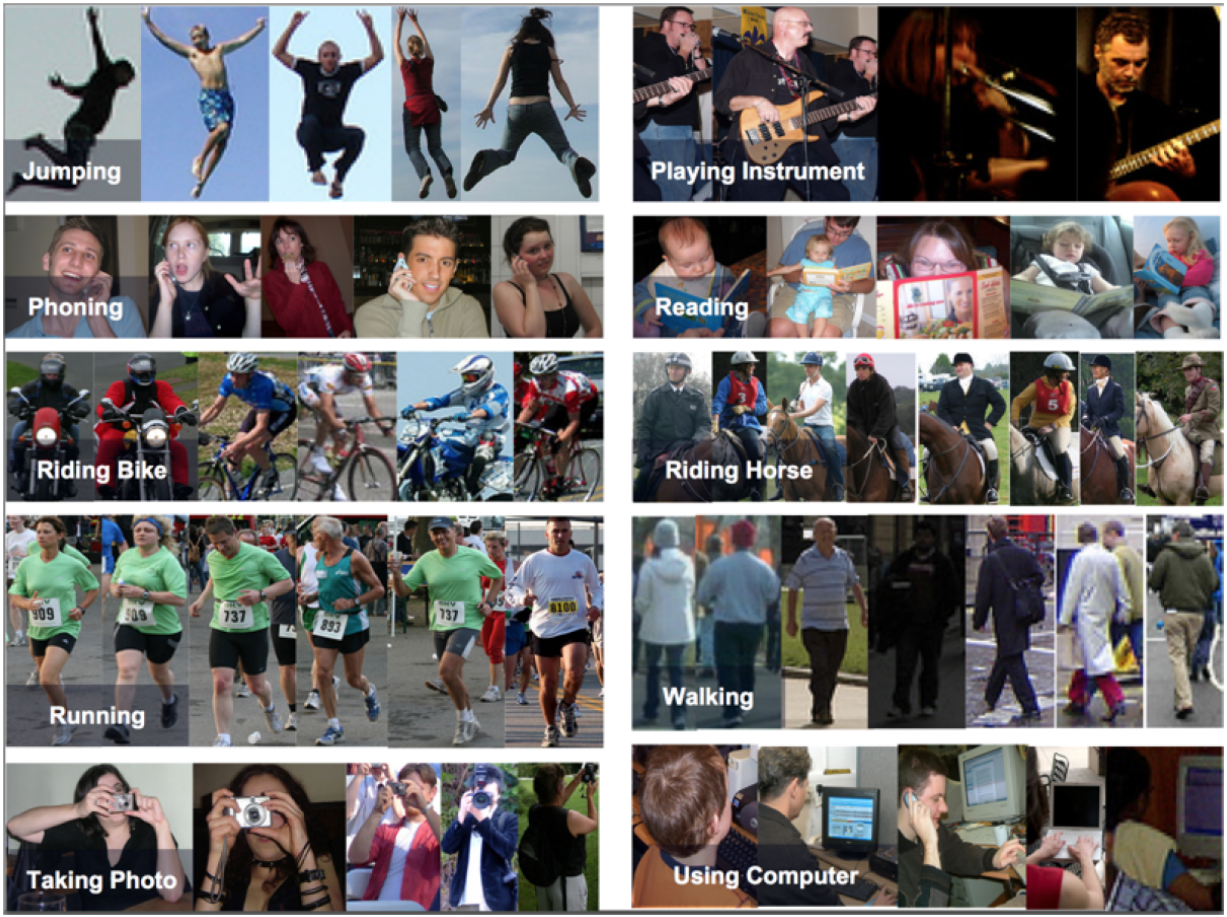



Figure: Top label is given by basic AlexNet CNN while bottom one is given by CNNTree (green color corresponds to a correct prediction)

# Outline

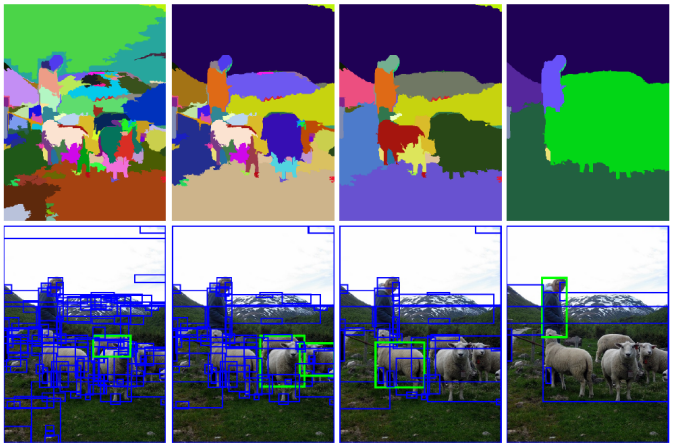
# Pose estimation - Deeppose

# Action recognition



# Outline

# Object detection and segmentation



(a)

(b)

# 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.

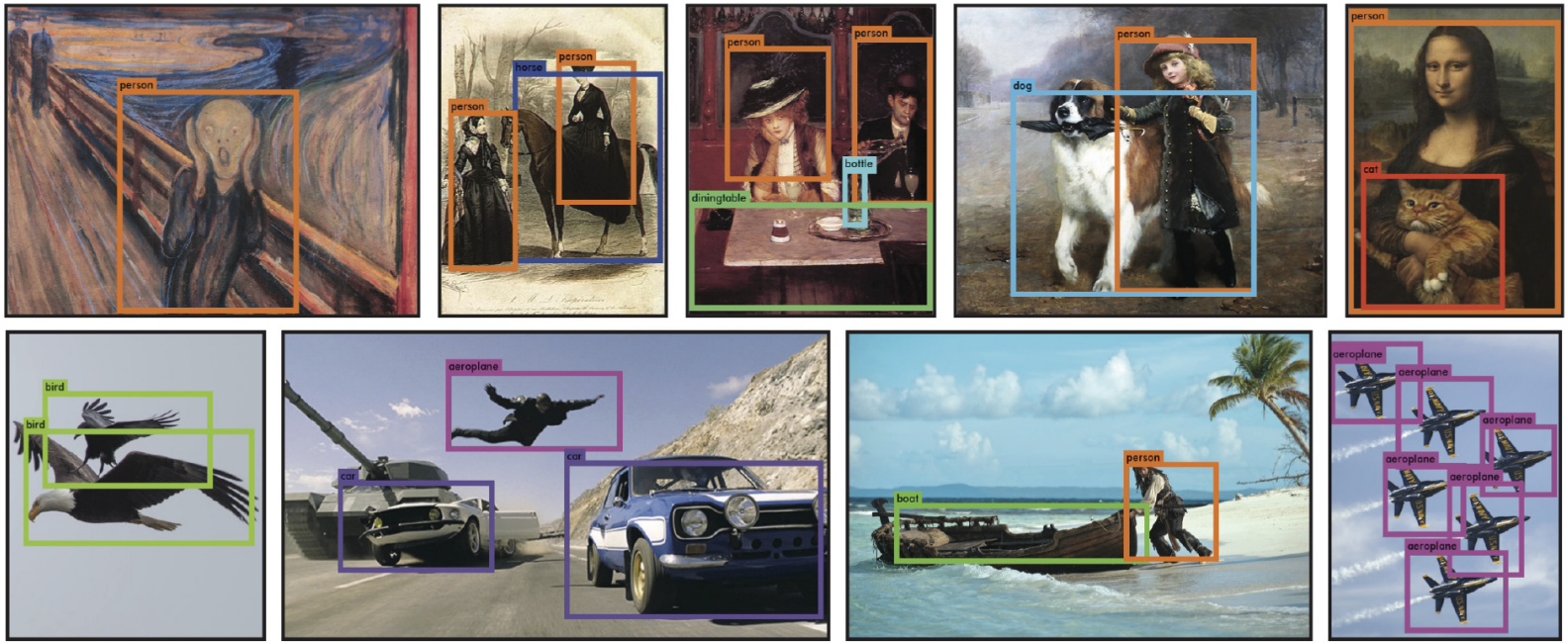
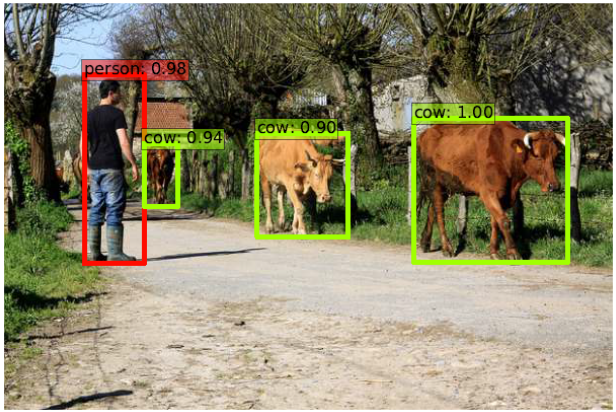
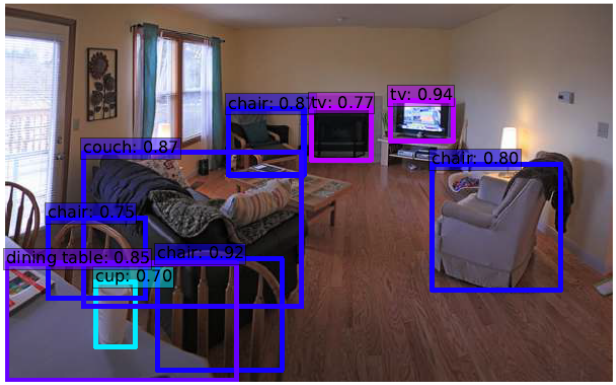
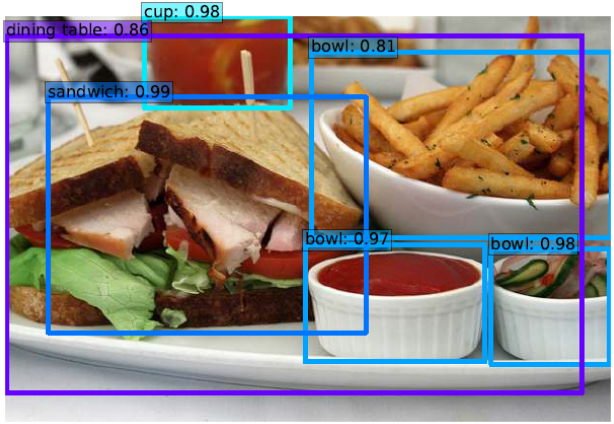



Figure: YOLO results

# SSD


person: 0.98
cow: 0.94
cow: 0.90
cow: 1.00


chair: 0.8
tv: 0.77
tv: 0.94
couch: 0.87
chair: 0.80
chair: 0.75
dining table: 0.85
chair: 0.92
cup: 0.70

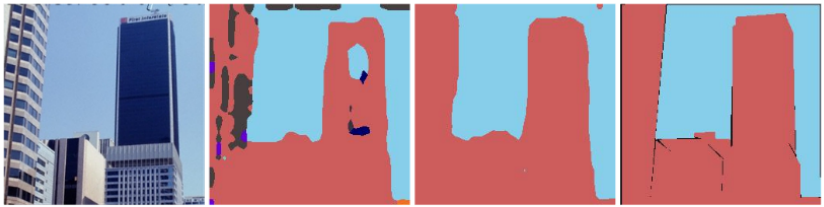

cup: 0.98
dining table: 0.86
bowl: 0.81
sandwich: 0.99
bowl: 0.97
bowl: 0.98


person: 1.00
backpack: 0.82
motorcycle: 0.99

# Outline

# Scene labeling - DAG-RNN

Input Image CNN DAG-RNN Ground Truth



Input Image CNN DAG-RNN Ground Truth

# Outline

# Object tracking

Object Detection in a video:

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

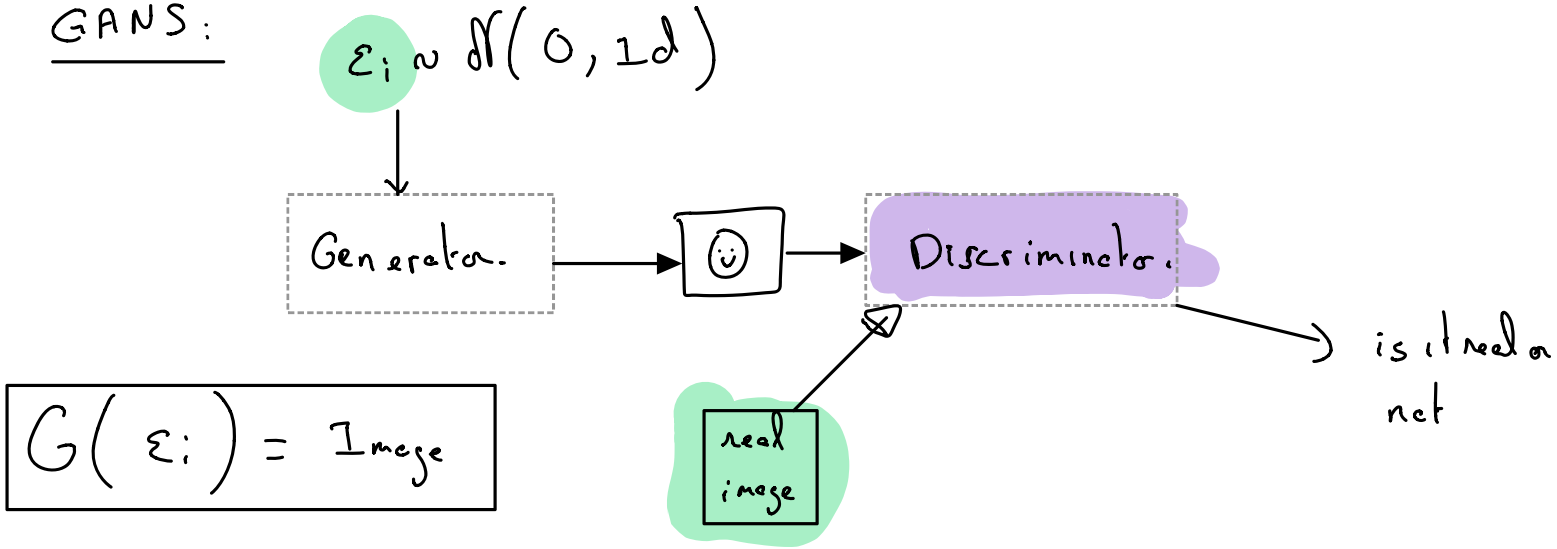
# Deep Learning - Many many other applications !

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.


`http://www.whichfaceisreal.com/`

GANS:




$$\min_{\text{Generator}} \left[ \max_{\text{Image}} \left( \text{acc}\left( \text{Discrim.} \right) \right) \right]$$

# 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 !

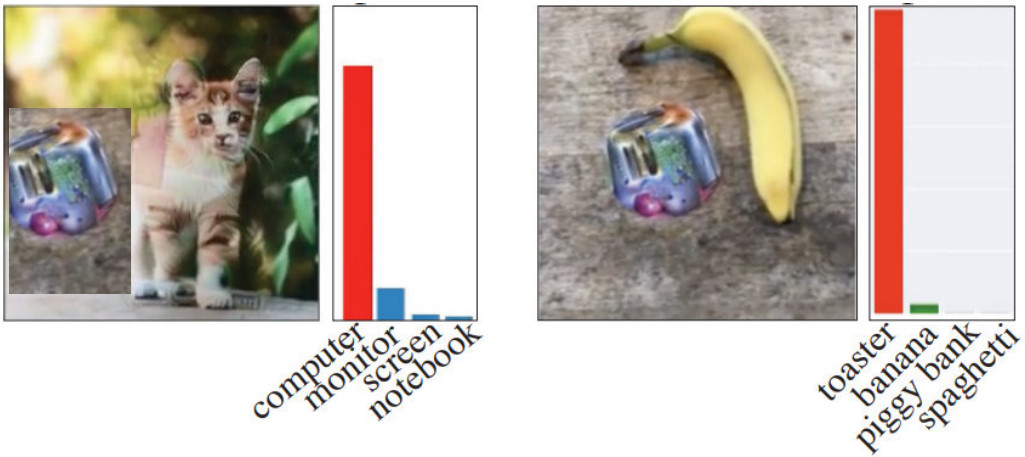
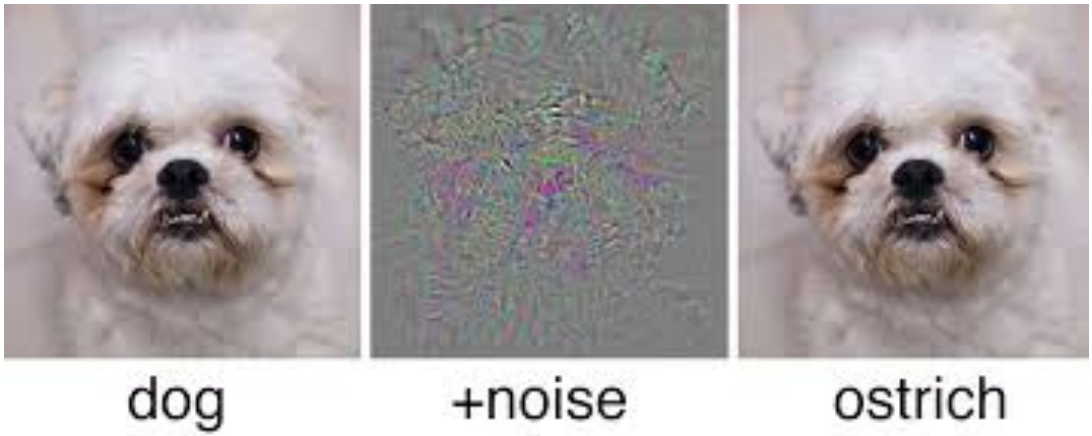



Figure: What happened here ??

# 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.:

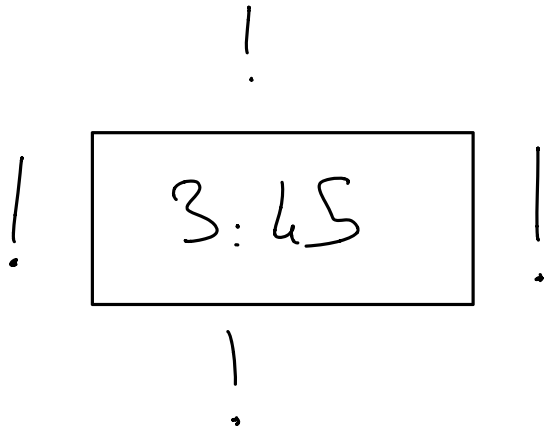
- mass surveillance
- tumor recognition

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-oheigeartaigh.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.