An introduction to Deep Learning
Part I - Fundamentals

JB Fiche, CBS-Montpellier & Plateforme PIBBS - MARS
Volker Bäcker, CRBM & MRI
**Cédric Hassen-Khodja,** CRBM & MRI

Goal of the training :

- Understand what an **Artificial Neural Network (ANN)** is and what are the main parameters to characterize them

- What is a **Convolutional Neural Network (CNN)** and why is it used for image processing

- Understand which **hyper-parameters** you will need to optimize

- Understand the **most common applications** and **where to find the tools for your applications**
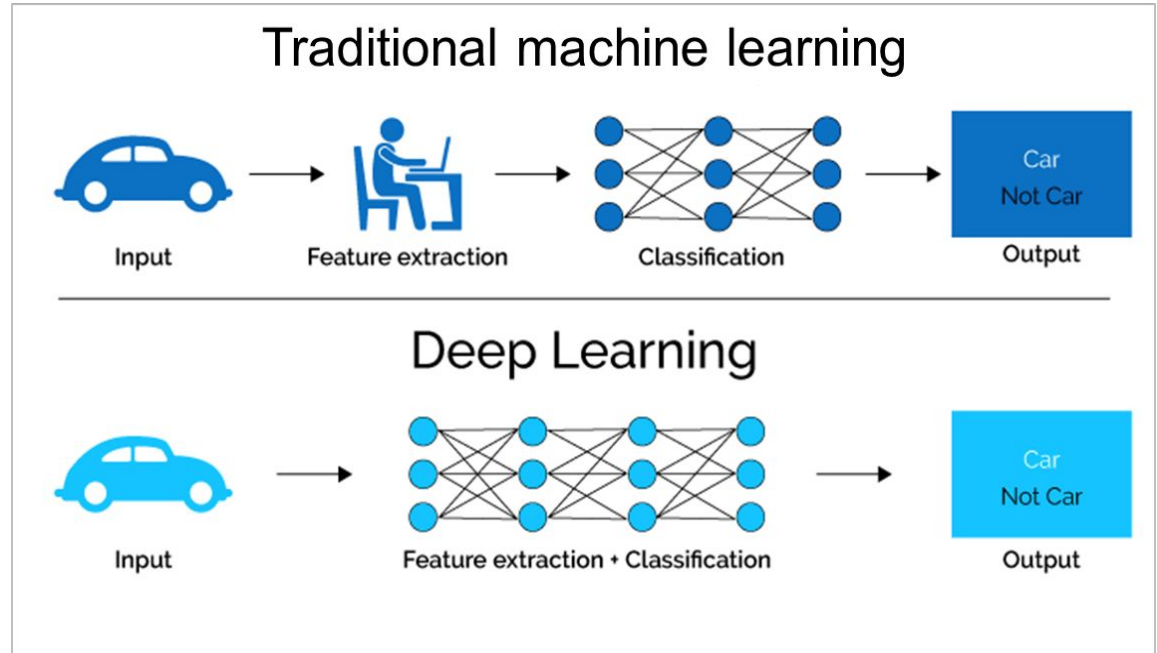
Outline of part I:

I. **Deep Learning : applications for image analysis**

II. General introduction & definition of neural networks

III. Example #1 : single neuron application

IV. Example #2: building a multi-layer network

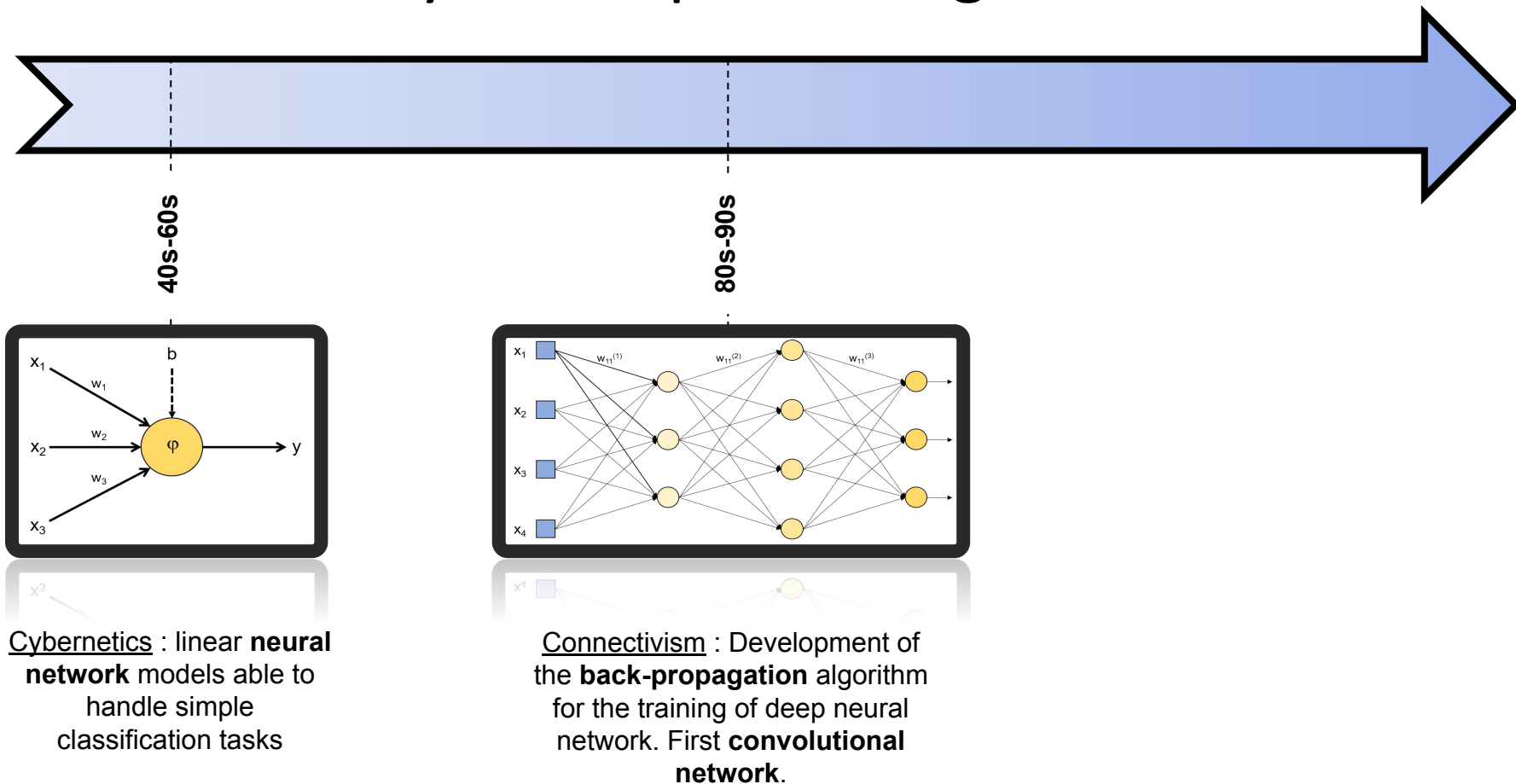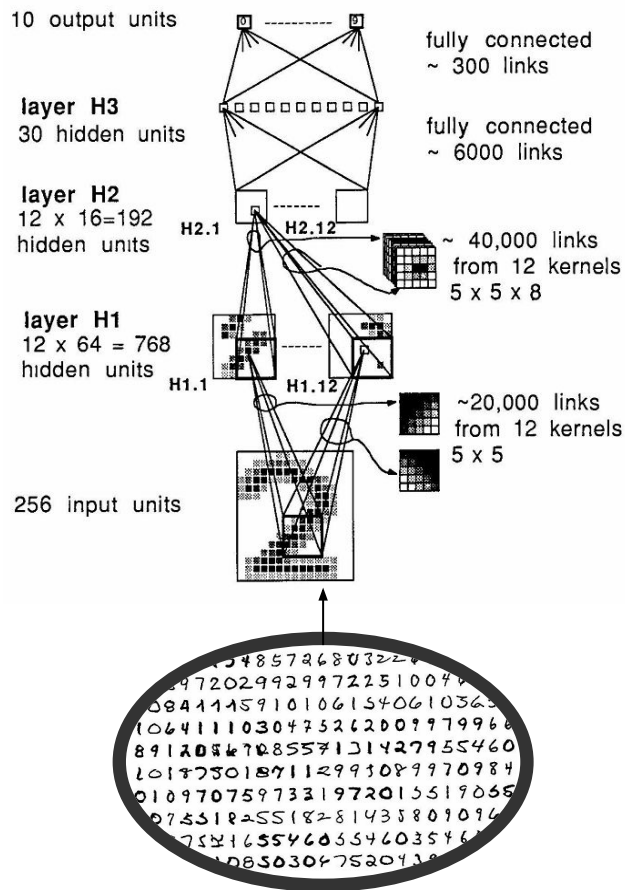# Machine learning vs. Deep Learning :
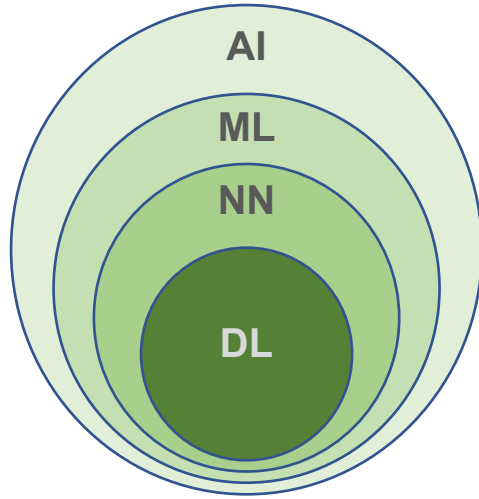
AI = artificial intelligence
ML = machine learning
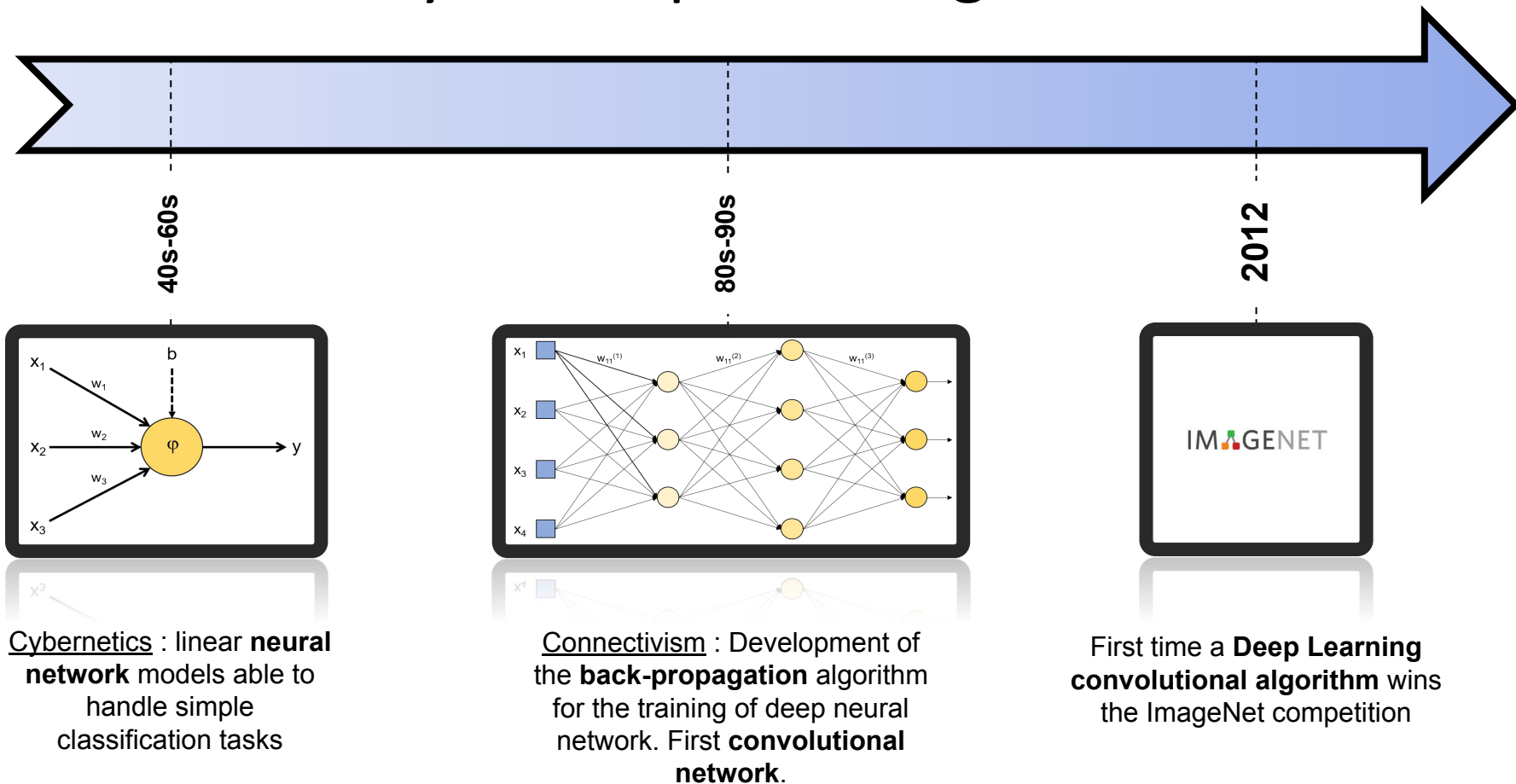NN = neural network
DL = deep learning

Pic Credit: Xenonstack | Machine Learning vs Deep Learning

A brief history of Deep Learning :

40s-60s

80s-90s

Cybernetics : linear **neural network** models able to handle simple classification tasks

Connectivism : Development of the **back-propagation** algorithm for the training of deep neural network. First **convolutional network**.

Handwritten zip code classification (1989)

10 output units

layer H3
30 hidden units

layer H2
12 x 16=192
hidden units

H2.1   H2.12

layer H1
12 x 64 = 768
hidden units

H1.1   H1.12

256 input units

fully connected
~ 300 links

fully connected
~ 6000 links
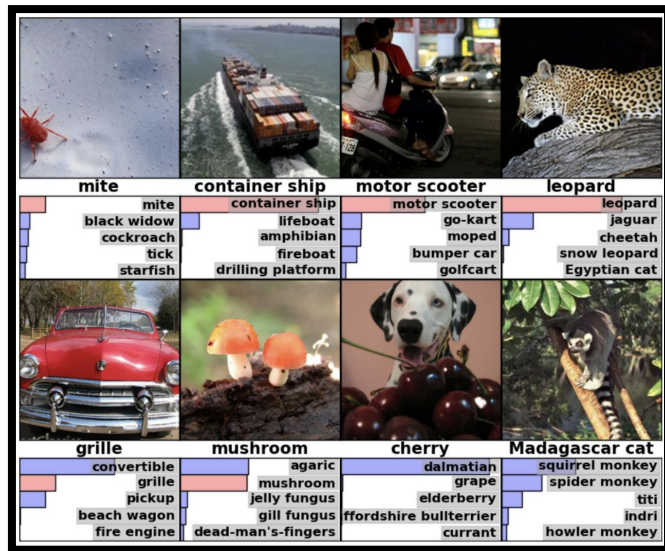
~ 40,000 links
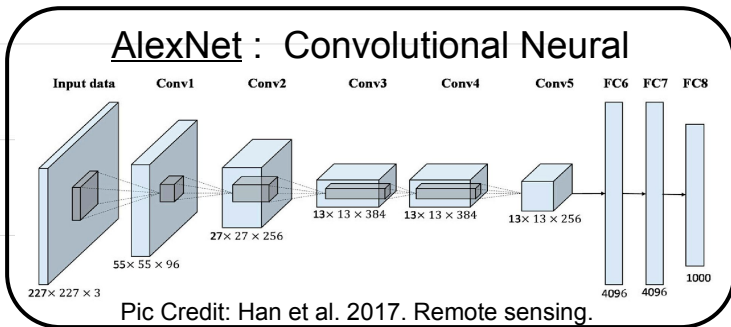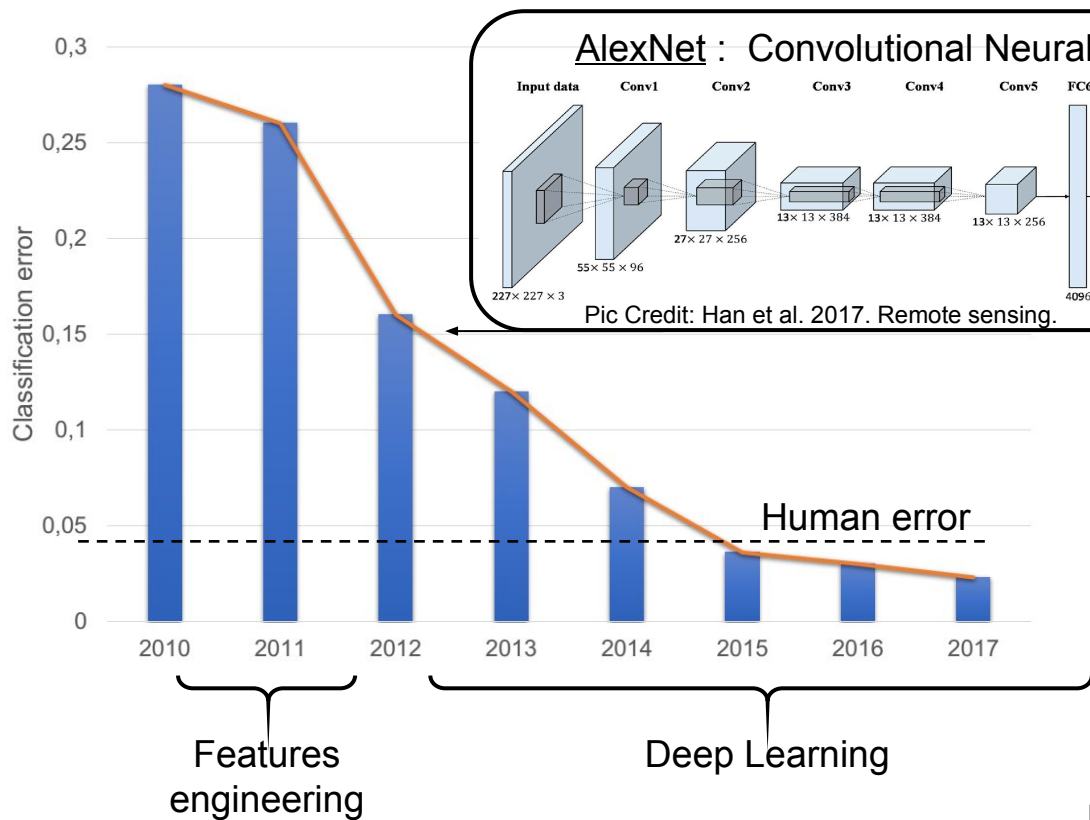from 12 kernels
5 x 5 x 8

~20,000 links
from 12 kernels
5 x 5

Deep learning models were already successfully used in the 90's for complicated tasks such as hand-written digit classification.

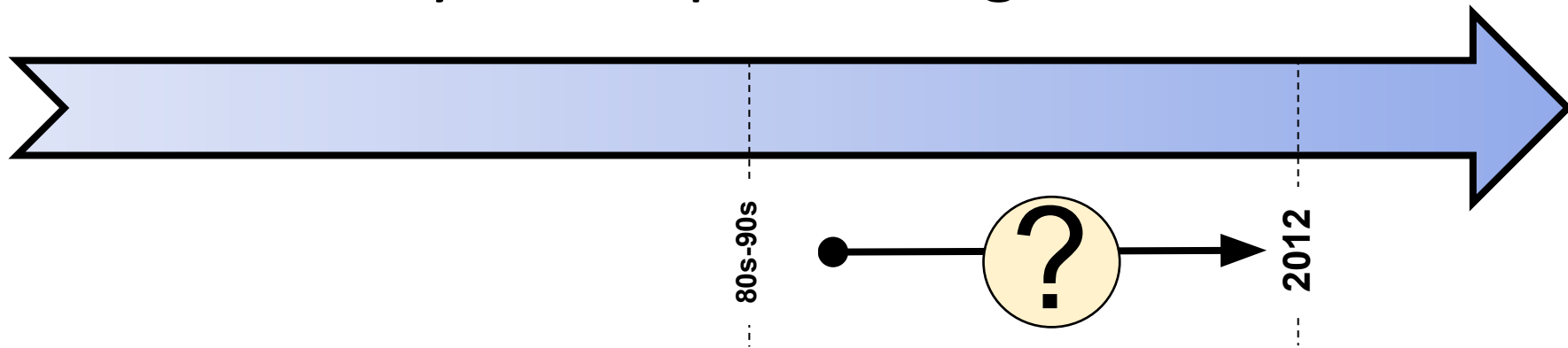The same algorithms are still used nowadays for image analysis.

A brief history of Deep Learning :

40s-60s

80s-90s

2012

Cybernetics : linear **neural network** models able to handle simple classification tasks

Connectivism : Development of the **back-propagation** algorithm for the training of deep neural network. First **convolutional network**.

First time a **Deep Learning convolutional algorithm** wins the ImageNet competition

Deep Learning breakthrough for image classification (2012) :

AlexNet : Convolutional Neural

Pic Credit: Han et al. 2017. Remote sensing.

Human error

Features engineering

Deep Learning

Krizhevsky et al. 2012. Advances in neural information processing systems

A brief history of Deep Learning :

1- Internet : offers a huge source of **high-resolution annotated images**, essential for an efficient **training**.

2- Computer resources : faster CPU and GPU, able to perform $>10^{12}$ **operations/s**

NVIDIA Titan RTX
>4000€

3- New algorithms : activation function (ReLu), faster optimizers, methods to avoid overfitting (Dropout), etc.

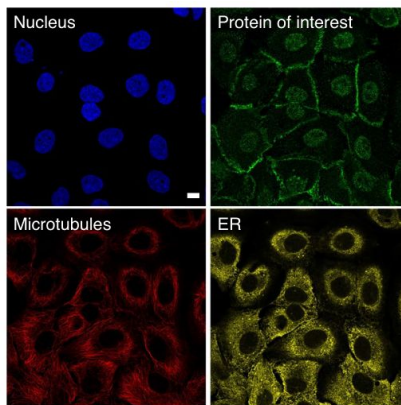# Most common applications for image analysis:

1- Image classification :

Pl@ntNet

https://plantnet.org/en/
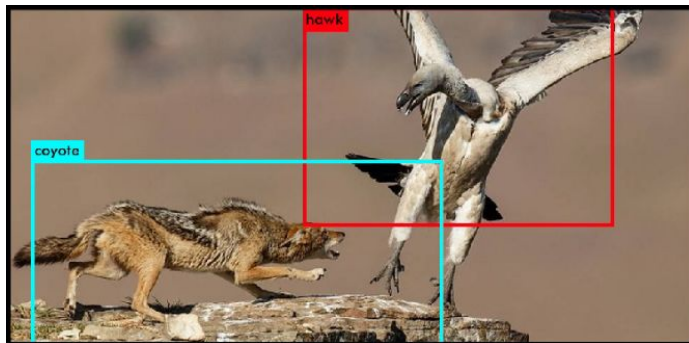
Redmon & Farhadi - 2016 YOLO9000, better, faster, stronger.
Von Charmier et al. - 2020 ZeroCostDL4Mic: an open platform to use Deep-Learning in Microscopy.
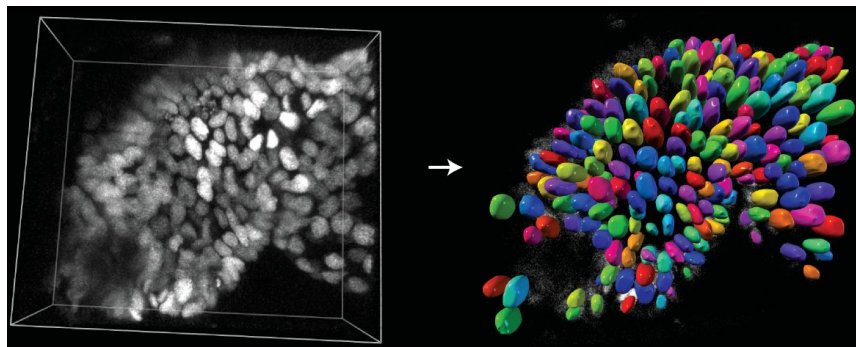https://github.com/HenriquesLab/ZeroCostDL4Mic

Ouyang et al. - 2019 Analysis of the Human Protein Atlas Image Classification competition.

Most common applications for image analysis:

1- Image classification :
2- Image segmentation :

2D / 3D segmentation of objects

https://github.com/stardist/stardist - Schmidt et al. - 2018 Cell Detection with Star-Convex Polygons

https://github.com/hci-unihd/plant-seg - Wolny et al. - 2020 Accurate and versatile 3D segmentation of plant tissues at cellular resolution

https://github.com/MouseLand/cellpose - Stringer et al. 2021 Cellpose: a generalist algorithm for cellular segmentation

https://github.com/kevinjohncutler/omnipose - Cutler et al. - 2022 Omnipose: a high-precision morphology independent solution for bacterial cell segmentation

https://github.com/vanvalenlab/intro-to-deepcell - Greenwald et al. - 2022 Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning

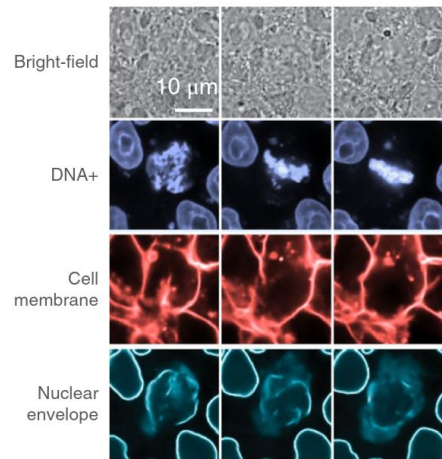# Most common applications for image analysis:

1- Image classification :
2- Image segmentation :
3- Augmented microscopy :
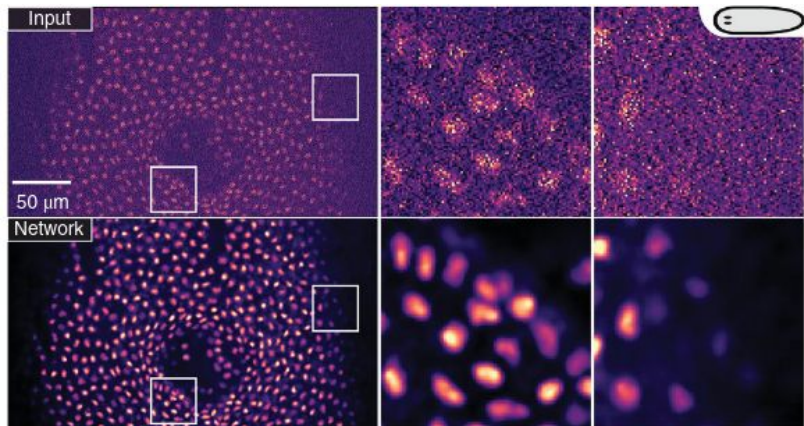
https://github.com/CSBDeep/CSBDeep - Weigert et al. 2017. Content-aware image restoration: pushing the limits of fluorescence microscopy
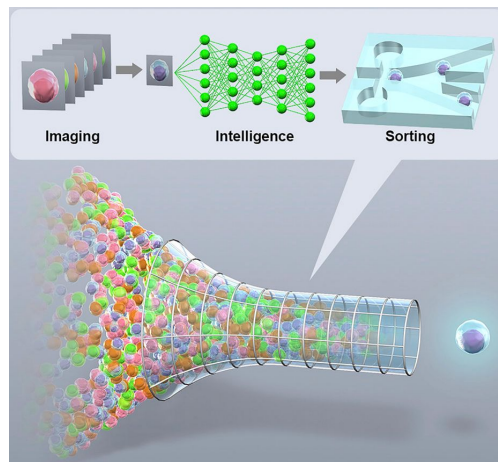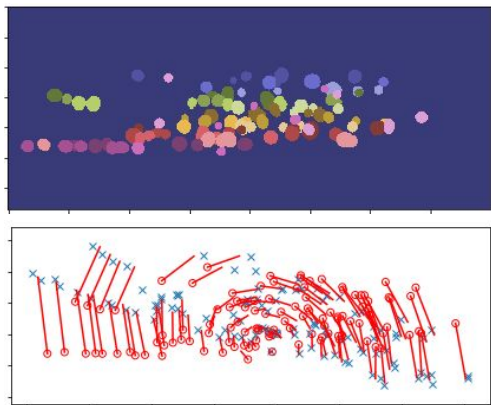
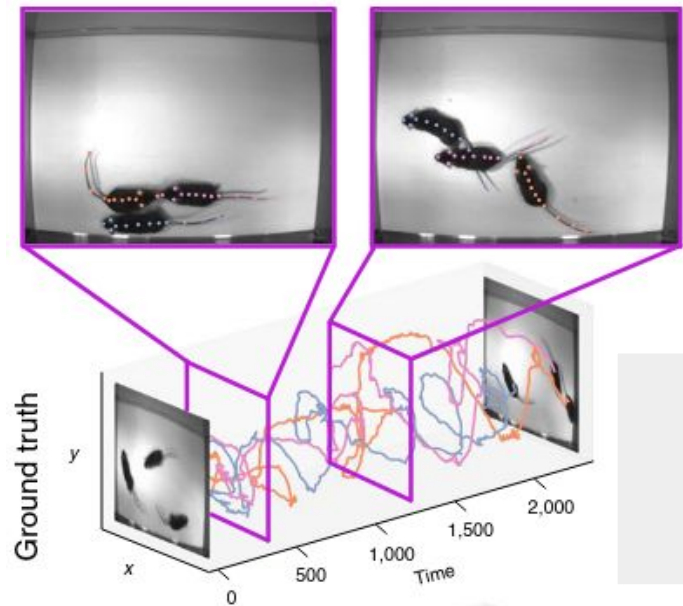Nitta et al. 2018. Intelligent Image-Activated Cell Sorting

Ounkomol et al. 2018. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy

Most common applications for image analysis:

1- Image classification :
2- Image segmentation :
3- Augmented microscopy :
4- Tracking :

Wen et al. - 2021 3DeeCellTracker, a deep learning-based pipeline for segmenting and tracking cells in 3D time lapse images
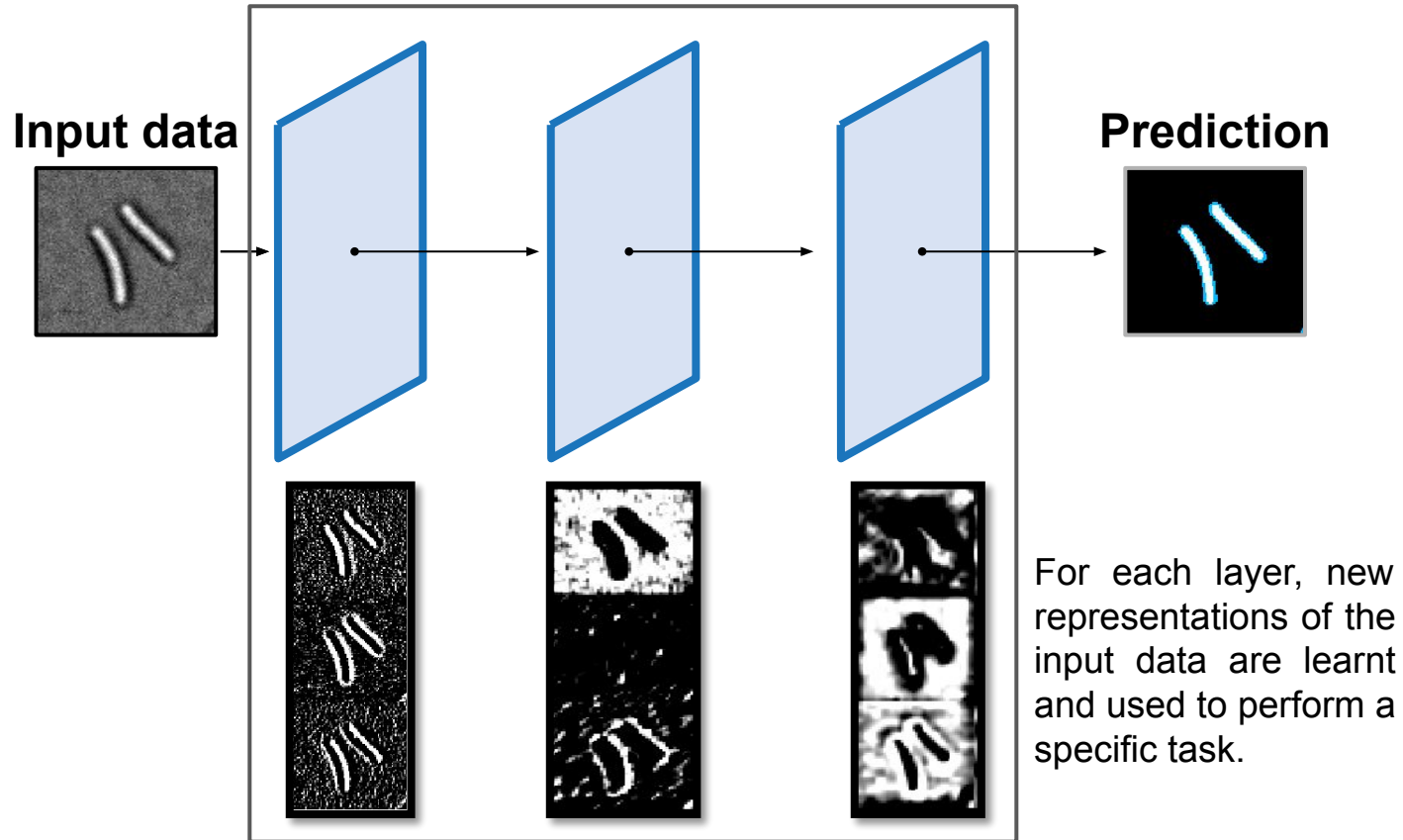

Lauer et al. - 2022 Multi-animal pose estimation, identification and tracking with DeepLabCut
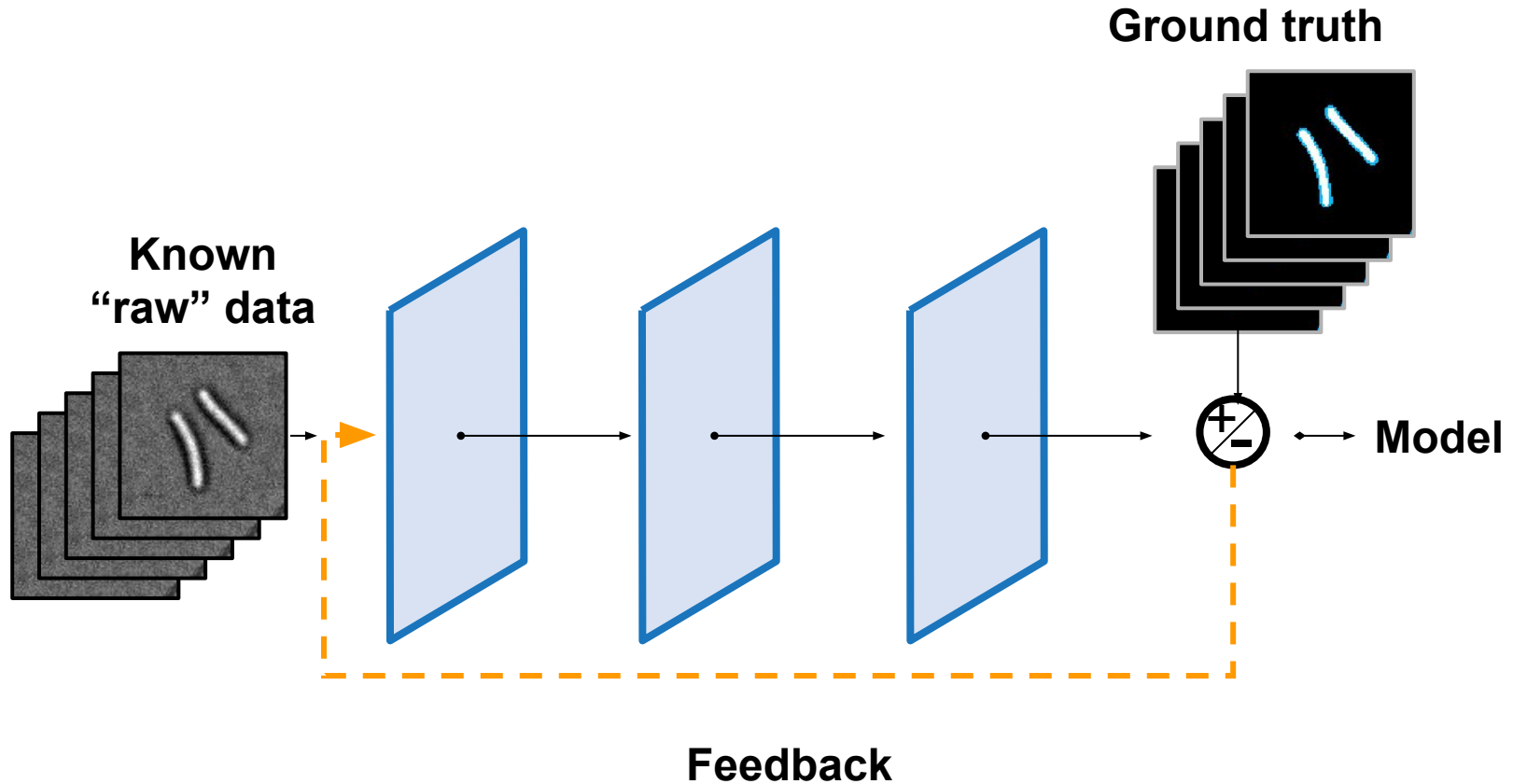
Outline :

Deep Learning : why "Deep"?

Input data

Prediction

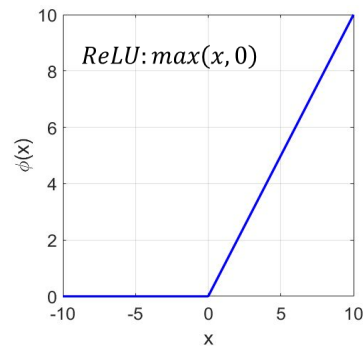For each layer, new representations of the input data are learnt and used to perform a specific task.

"Learning" under supervision :

Supervised deep learning network :

Training set :

Trained model

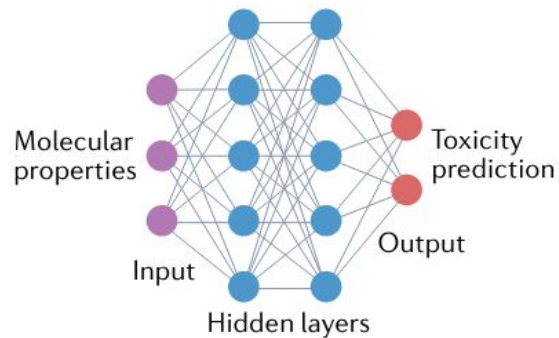**Validation/testing sets :**

Prediction

95,6%

Definition of a single neuron

Many network architectures

a **Multilayer perceptron**
Molecular properties
Input
Hidden layers
Toxicity prediction
Output

b **Convolutional neural network**
Microscopy image
Filter
Next layer

c **Recurrent neural network**
DNA sequence
A C T C
Input
Hidden state
Output
0.4 0.8 0.7 0.3
Transcription factor binding

d **Graph convolutional network**
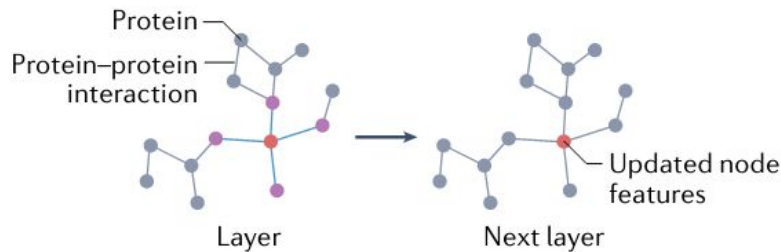Protein
Protein–protein interaction
Layer
Next layer
Updated node features

e **Autoencoder**
Encoder neural network
Decoder neural network
Protein sequence
R G A D I
Protein sequence
R G A E I
Latent representation
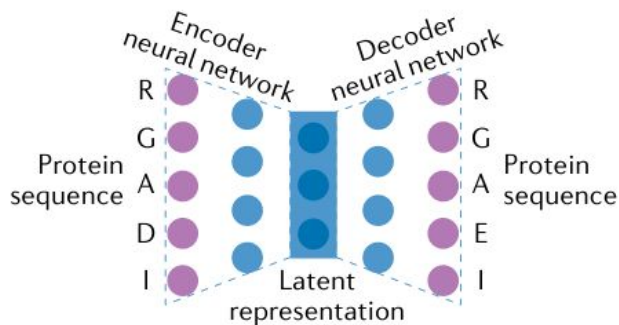
Regression vs. Classification
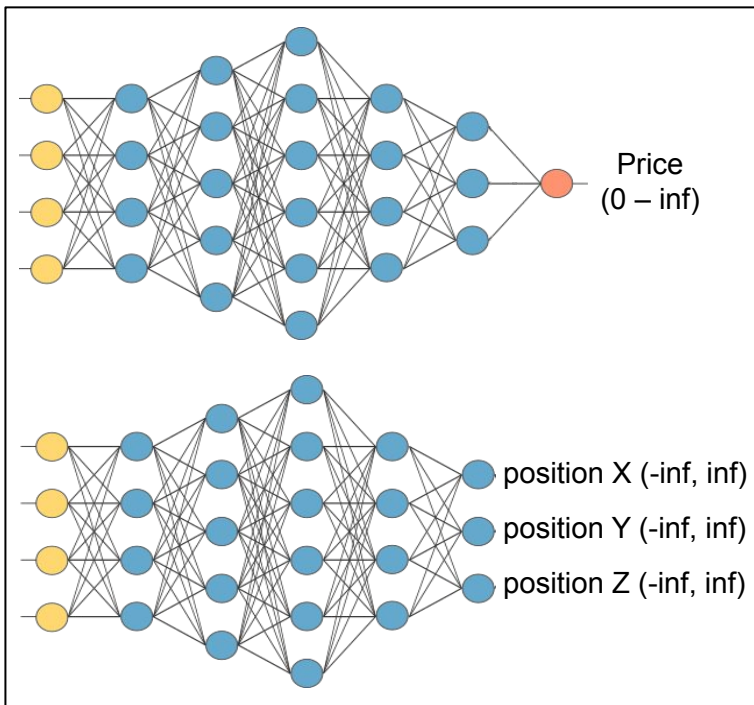

Regression : output is one or more real numbers

Price (0 – inf)

position X (-inf, inf)
position Y (-inf, inf)
position Z (-inf, inf)

Classification : output is the probability that input belong to one or more classes

1 – 0
Yes – No
Cat - dog

prob. Class 1
prob. Class 2
prob. Class 3

Training, testing and validation sets

Outline :

Train a single neuron classifier

Example n°1 : Ex1_Clusterization_linearly_separated.ipynb

1. Understand the principle of the training
2. Train the classifier and test its accuracy
3. First step with Keras/TensorFlow

Train a single neuron classifier

Input data:
X,Y
coordinates

Goal of the training = find
the best set of
parameters {w_1, w_2, b}
allowing for an accurate
clusterization of the input
data

Prediction :
class red or
blue

● = 0
● = 1
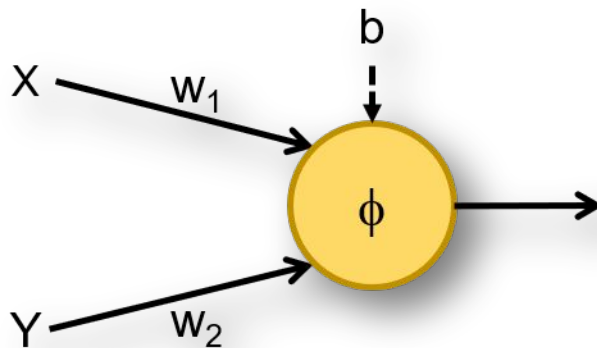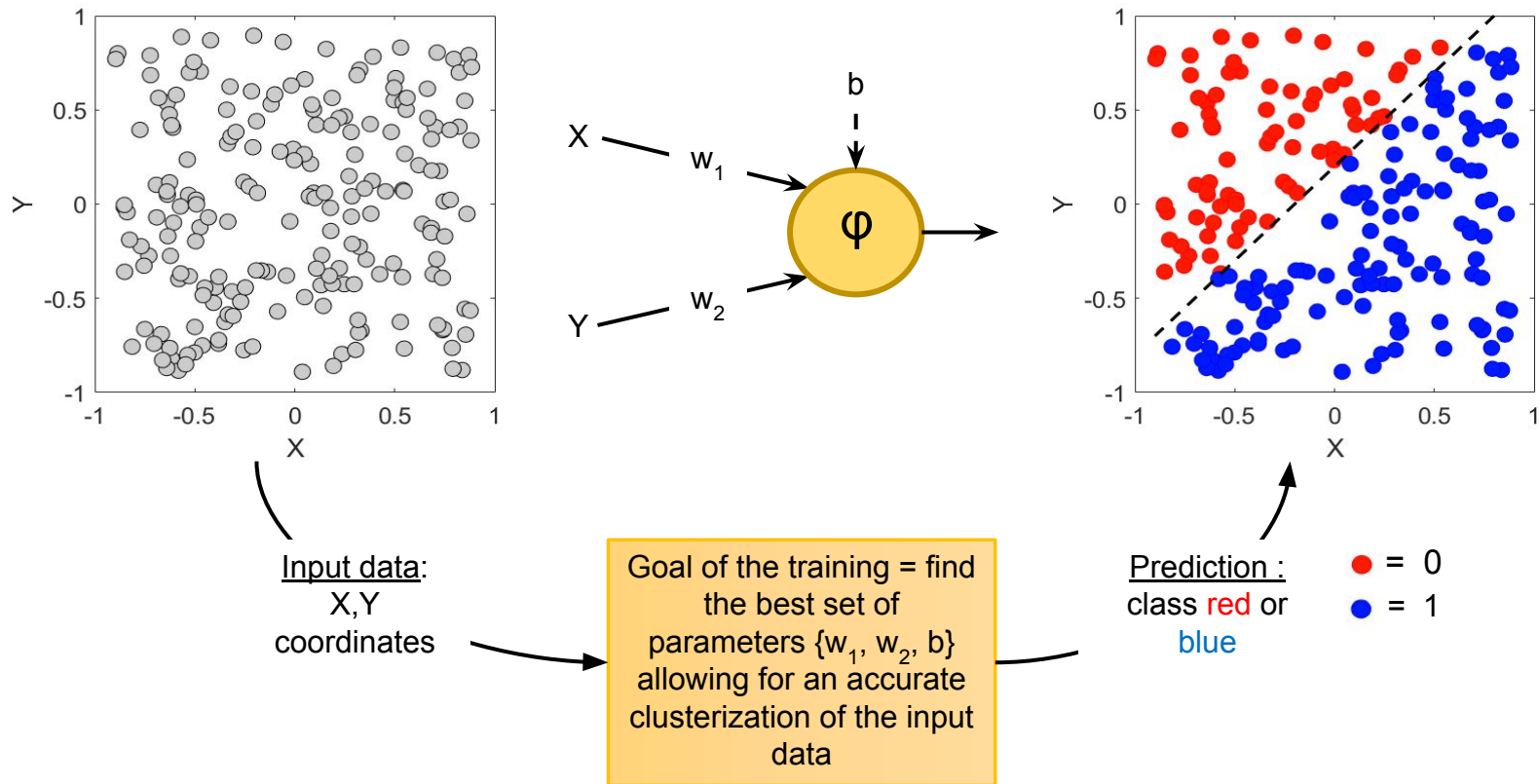
Definition of a classifier with Keras

1- Definition of the network architecture

```python
from keras import models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
  Dense(1,activation='sigmoid', input_shape=(2,))
])
```

2- Definition of the training options

```python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

3- Training

```python
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```

Definition of a single neuron
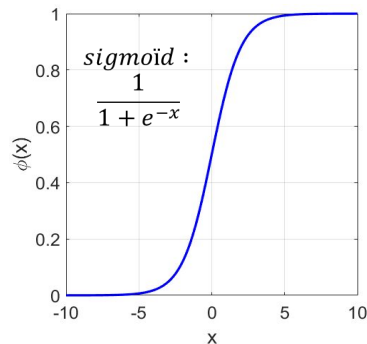
1- Definition of the network architecture

```
from keras import models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
  Dense(1,activation='sigmoid', input_shape=(2,))
])
```

# of neurons ϕ X,Y

b

X w_1

Y w_2

ϕ

$\phi(Xw_1 + Yw_2 + b)$

$sigmoïd :$
$$\frac{1}{1 + e^{-x}}$$

How the neuron works ?

$$\phi(Xw_1 + Yw_2 + b)$$

Here we choose the **sigmoid** as *activation function* and a "prediction" is calculated

The weights are randomly initialized and the bias set to zero.

$sigmo\ddot{i}d :$
$$\frac{1}{1 + e^{-x}}$$

>0,5

0.5

< 0,5

Definition of the training options
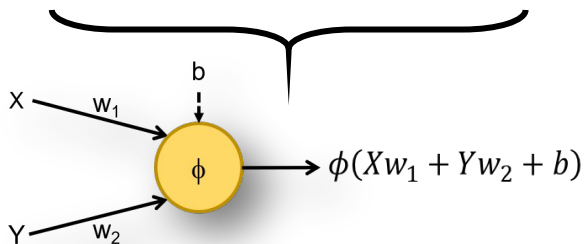
1- Definition of the network architecture

```python
from keras import models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(1,activation='sigmoid', input_shape=(2,))
])
```
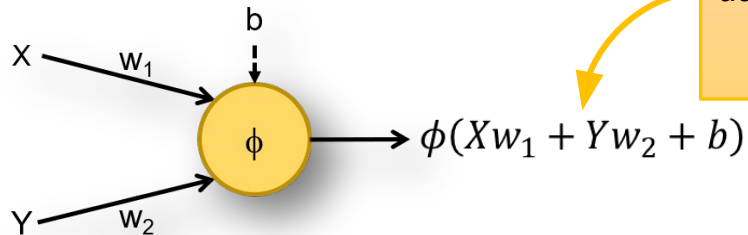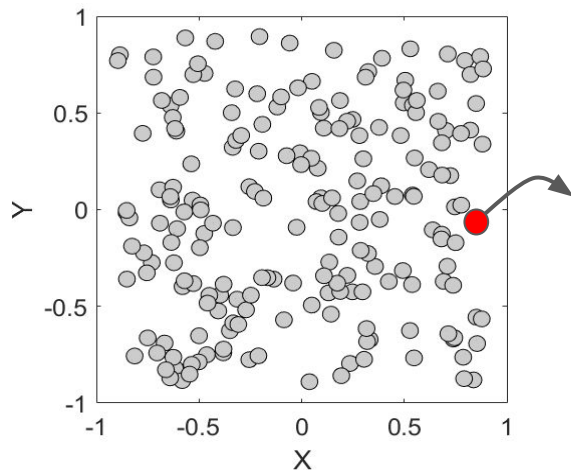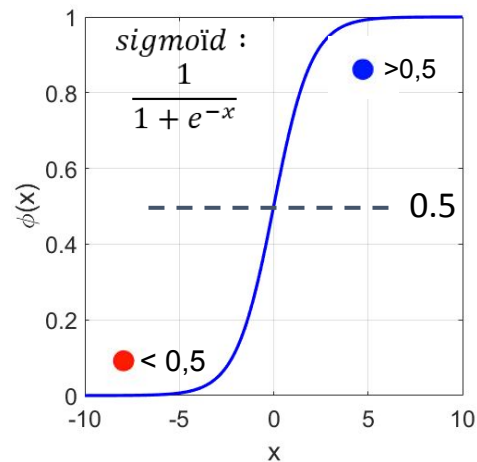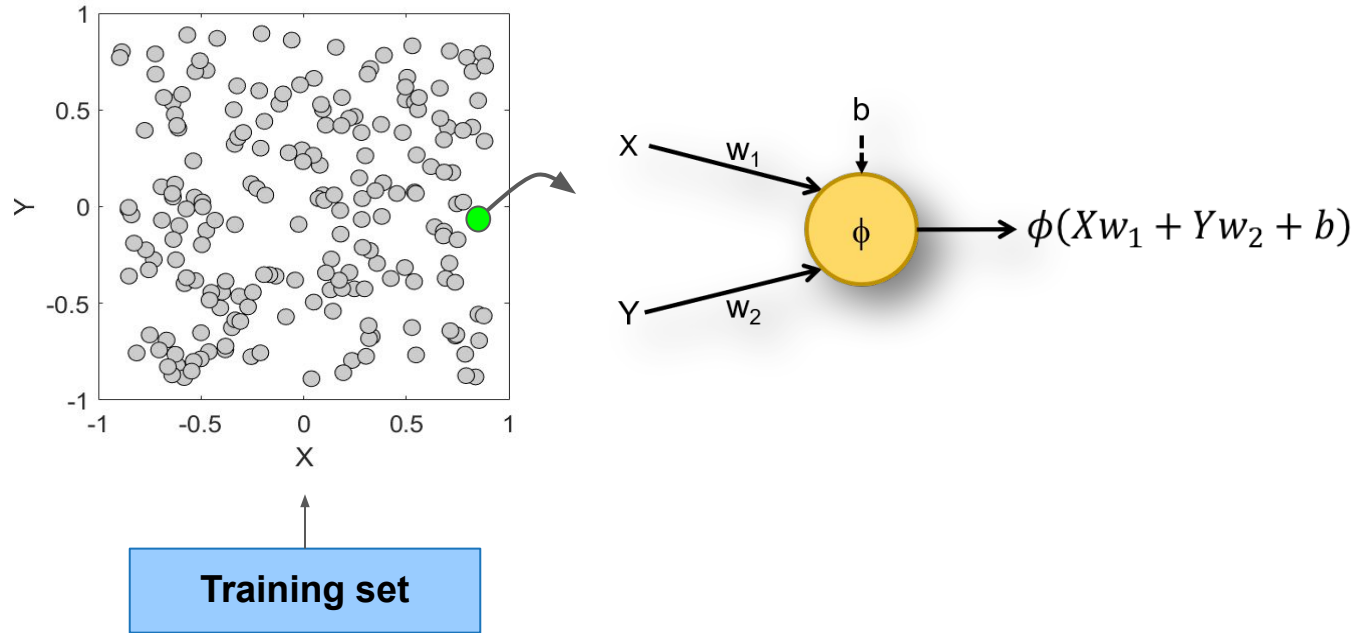
2- Definition of the training options

```python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```
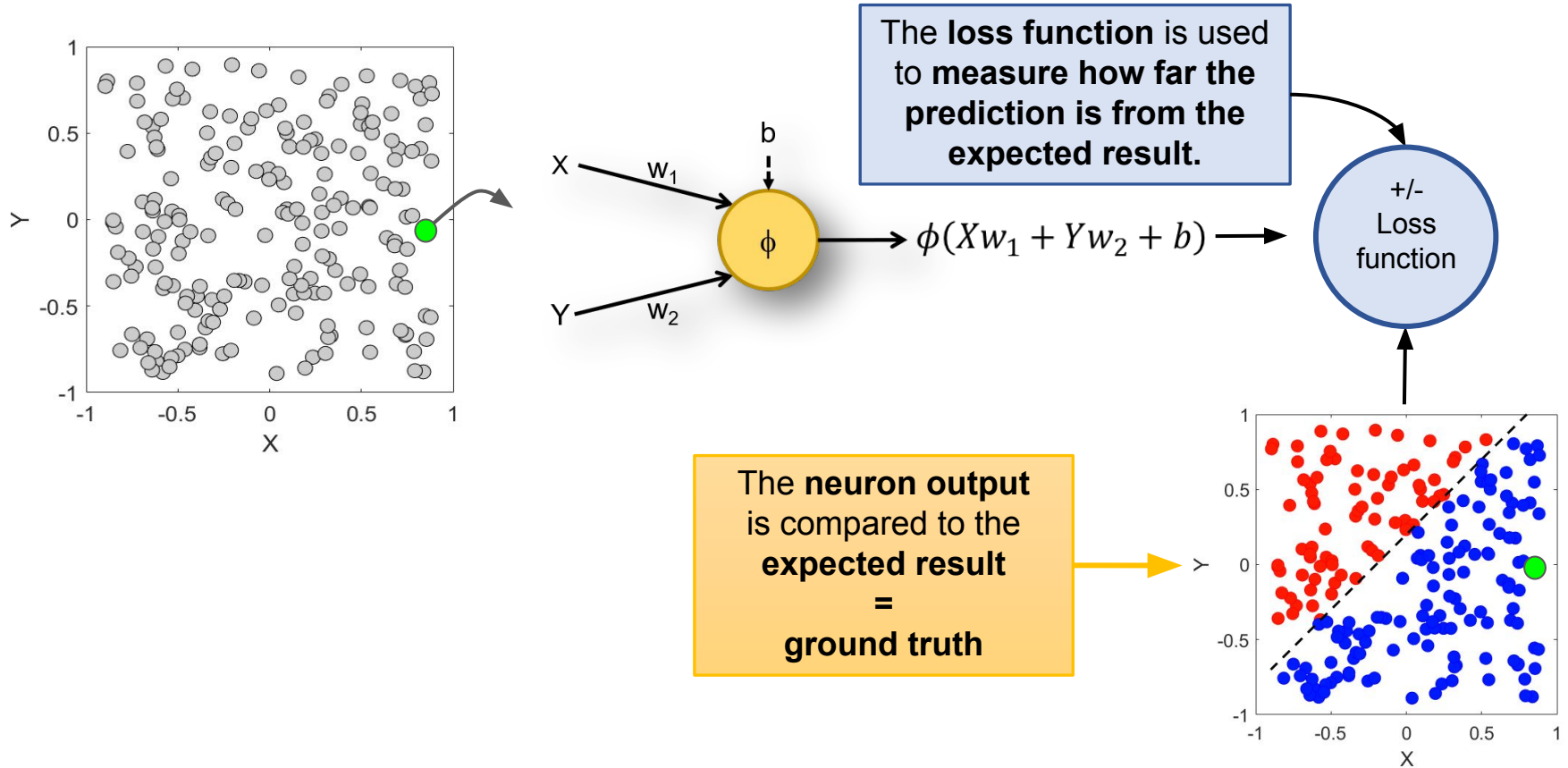
3- Training

```python
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```

Compiling : defining training process

Training set

$$\phi(Xw_1 + Yw_2 + b)$$

Compiling : defining training process

The **loss function** is used to **measure how far the prediction is from the expected result.**

$$\phi(Xw_1 + Yw_2 + b)$$

+/-
Loss function

The **neuron output** is compared to the **expected result**
=
ground truth

Compiling : defining training process

Optimizer
sgd, adam, etc.

Loss score

$\Delta w_2, \Delta w_1, \Delta b$

b

X $\xrightarrow{w_1}$

ϕ

$\phi(Xw_1 + Yw_2 + b) \longrightarrow$

+/-
Loss
function

Y $\xrightarrow{w_2}$

$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$

α is the learning rate

Loss score E

i

i+1

i+2

i+3

Wi
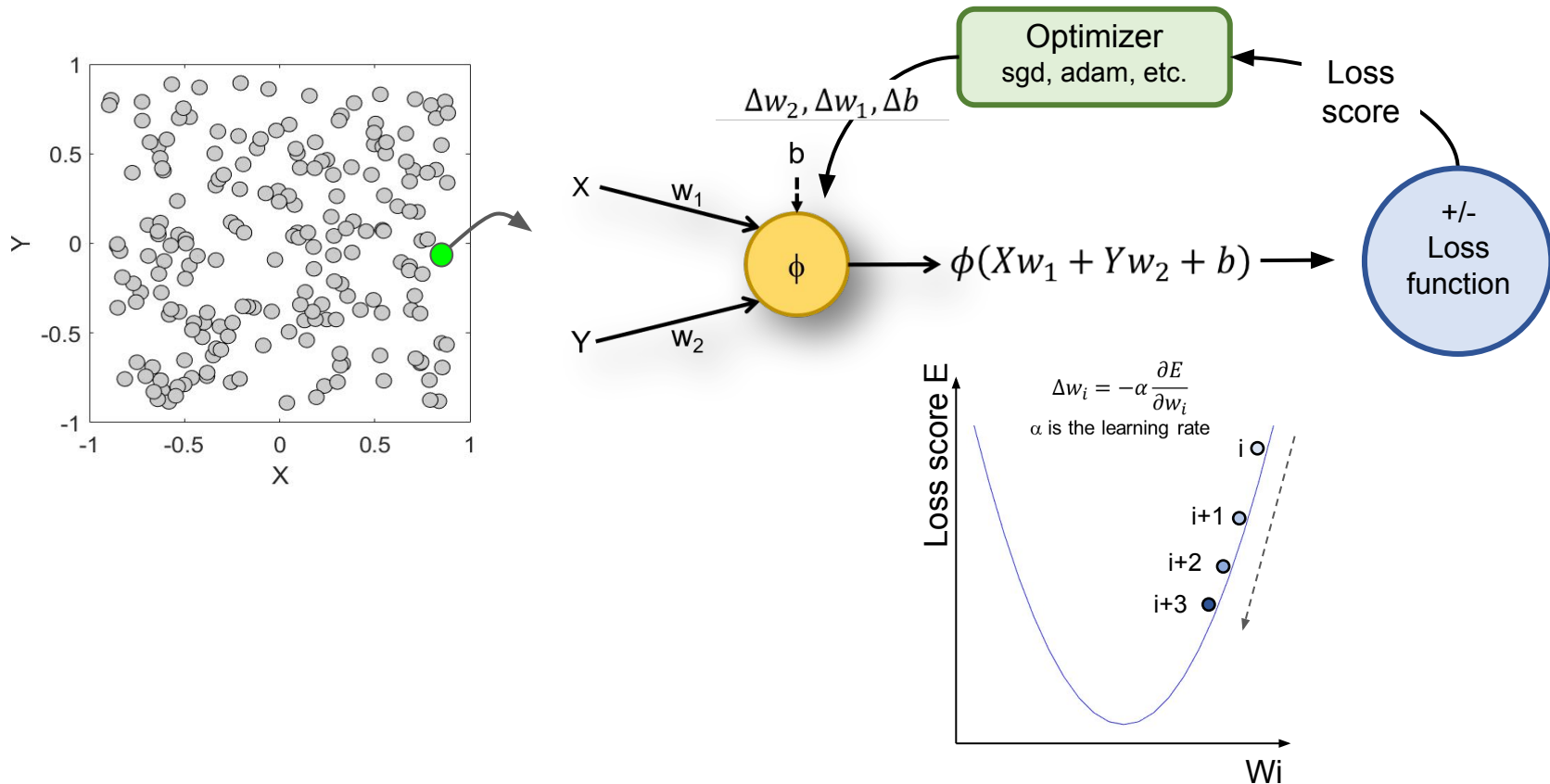
Model compiling

2- Definition of the training options

```python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Launching the training

1- Definition of the network architecture

```
from keras import models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
  Dense(1,activation='sigmoid', input_shape=(2,))
])
```

2- Definition of the training options

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```
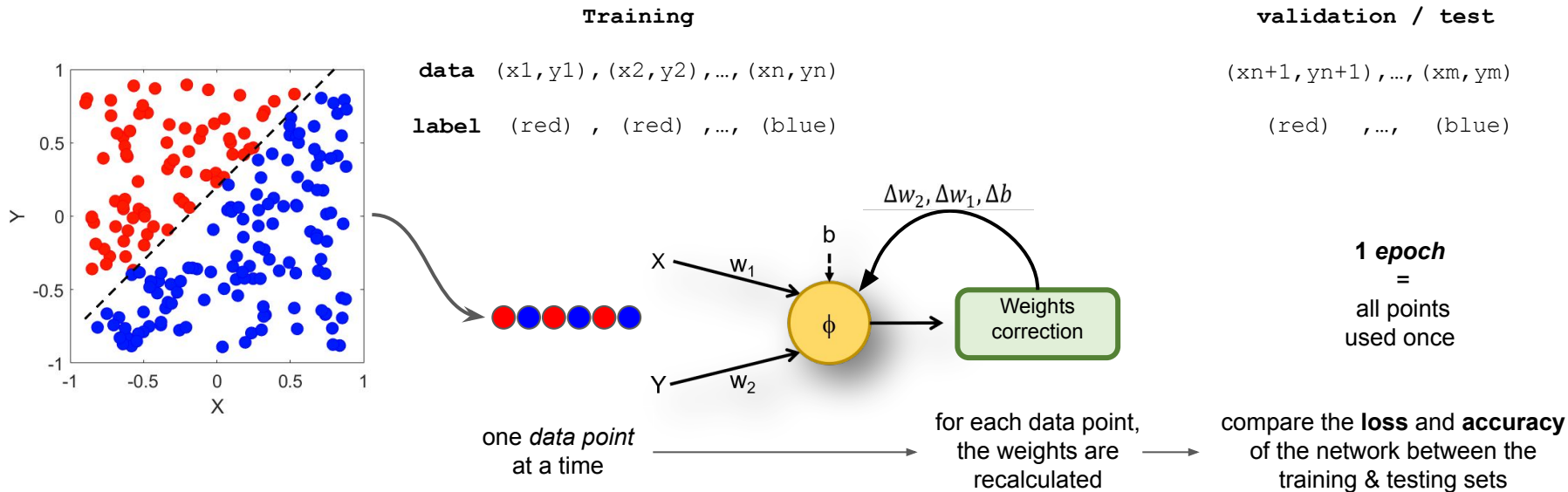
3- Training

```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```
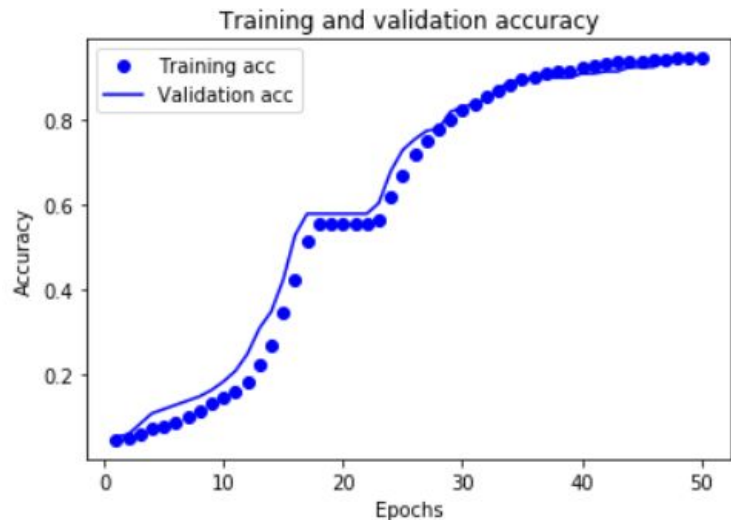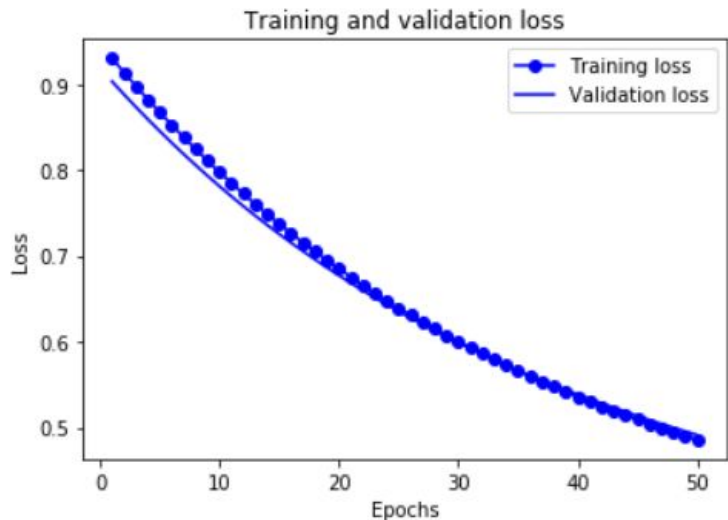
Start the training

3- Training

```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```

Training

validation / test

data (x1,y1),(x2,y2),…,(xn,yn)

(xn+1,yn+1),…,(xm,ym)

label (red) , (red) ,…, (blue)

(red) ,…, (blue)

$\Delta w_2, \Delta w_1, \Delta b$

b

X w_1

φ

Weights correction

Y w_2

1 *epoch*
=
all points
used once

one *data point*
at a time

for each data point,
the weights are
recalculated

compare the **loss** and **accuracy**
of the network between the
training & testing sets

Training results

3- Training

```
history = model.fit(Training_data,
                     Training_label,
                     epochs = 100,
                     validation_data = (Validation_data, Validation_label))
```


Training and validation loss

Training and validation accuracy

Summary

1- Definition of the network architecture

```python
from keras import models
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
  Dense(1,activation='sigmoid', input_shape=(2,))
])
```
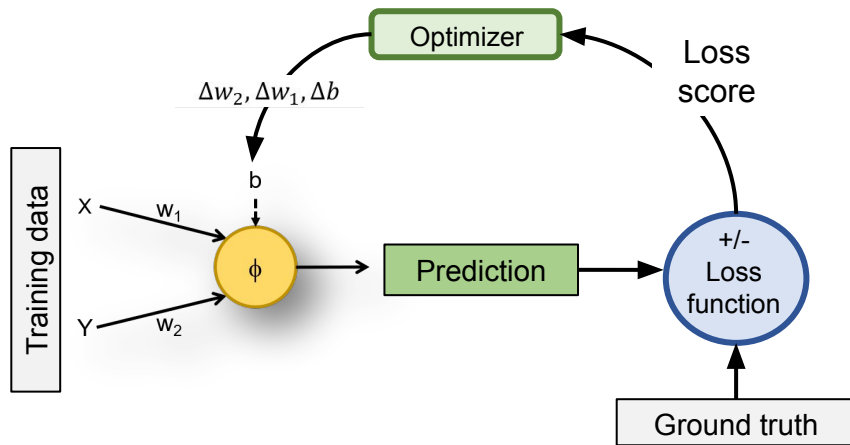
2- Definition of the training options

```python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

3- Training

```python
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```
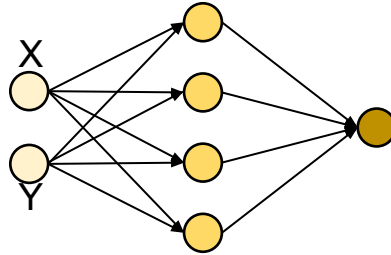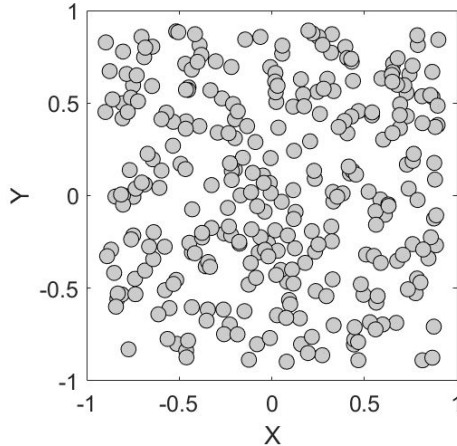
Outline :

Example 2: classify non-linearly separable data

Example n°2 : Clusterization_not_linearly_separated_parabole

1. Observe the limitations of single-layer model
2. Find a simple architecture able to solve this classification problem
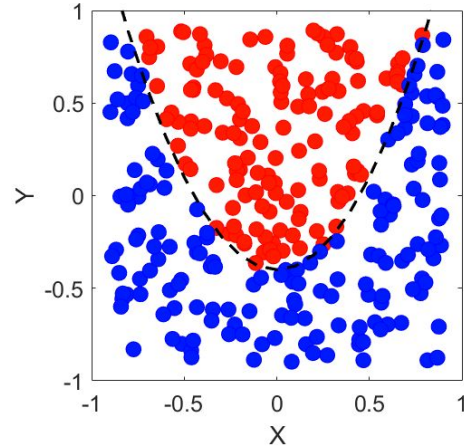
Multiple layers :

Activation function:
try sigmoid or relu
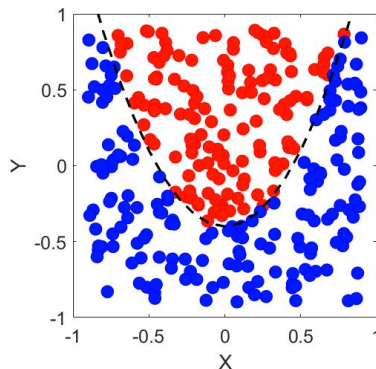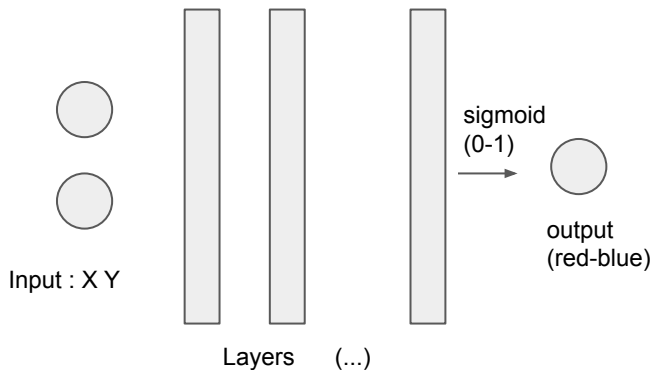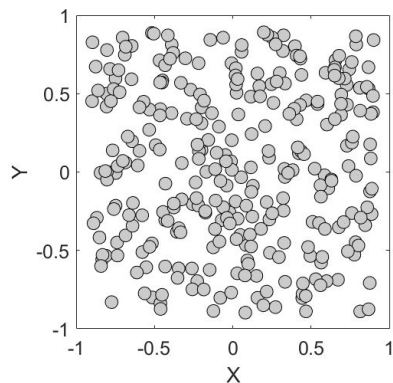
1st layer needs input specified
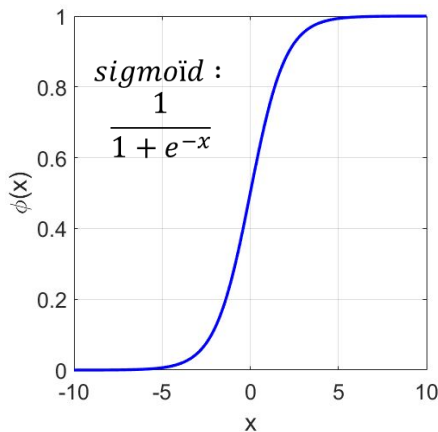
number of
neurons

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
Dense(32, activation='relu', input_shape=(2,)),
Dense(16, activation='relu'),
  Dense(1, activation='sigmoid')
])

model.compile(optimizer = 'adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```
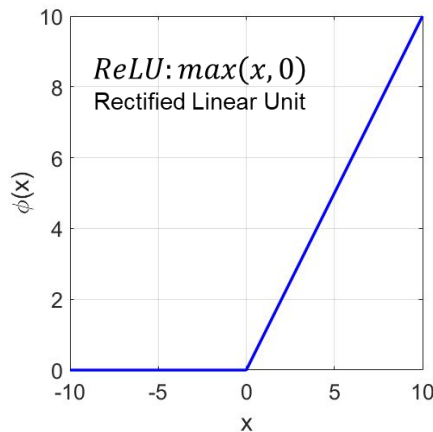
1st layer

2nd layer
…

Input : X Y

Layers   (...)

sigmoid
(0-1)

output
(red-blue)

The two most popular activation function

Plot of sigmoid function, $sigmo\ddot{\imath}d : \dfrac{1}{1 + e^{-x}}$, with axes $\phi(x)$ versus x.

Plot of ReLU function, $ReLU : max(x, 0)$, Rectified Linear Unit, with axes $\phi(x)$ versus x.

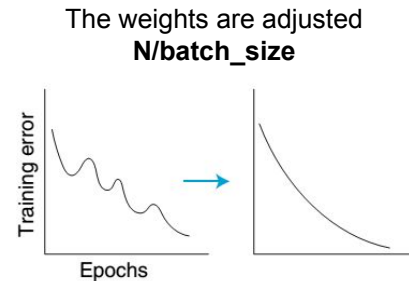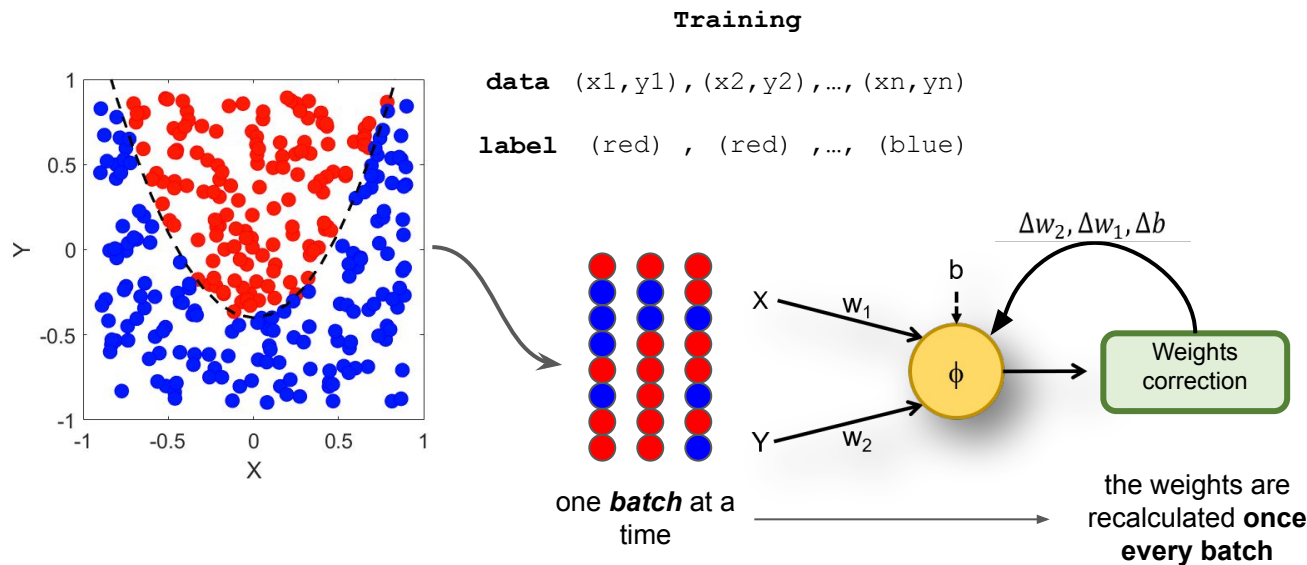Gradient is saturating for large output values and exp() is a computer-expensive operation

Non-saturating gradient and very fast operation. However, negative values are discarded.

Training with mini-batch
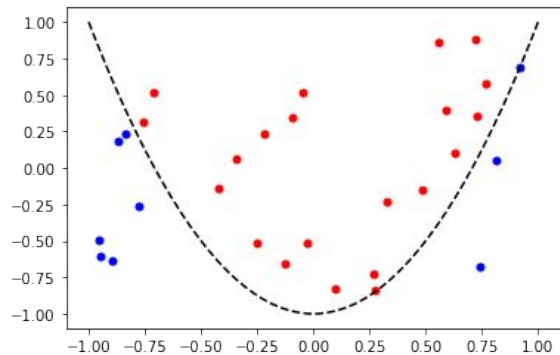
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 150,
                    batch_size = 8,
                    validation_data = (Validation_data, Validation_label))
```
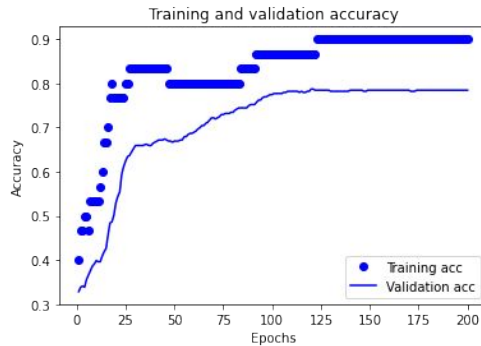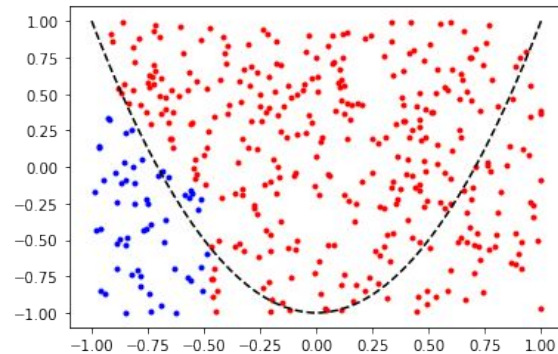
Good practice to use a power of 2

Training

data (x1,y1),(x2,y2),…,(xn,yn)

label (red) , (red) ,…, (blue)

$$\Delta w_2, \Delta w_1, \Delta b$$

b

X w_1

ϕ

Weights correction

Y w_2

one *batch* at a time

the weights are recalculated **once every batch**

The weights are adjusted **N/batch_size**

Pic credit : Moen et al. 2019. Nat. Meth.

Effect of imbalanced classes
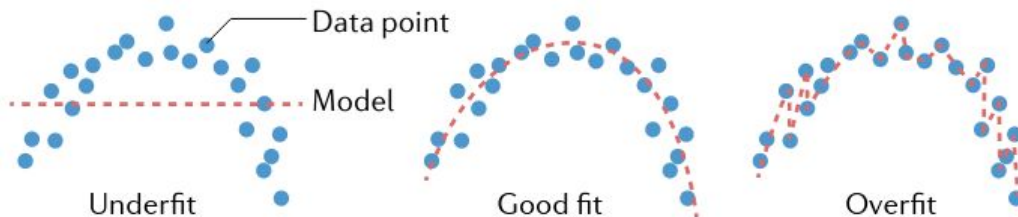

Training set

**Accuracy ~ 0.8**

Prediction on test set

Overfitting

The production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably
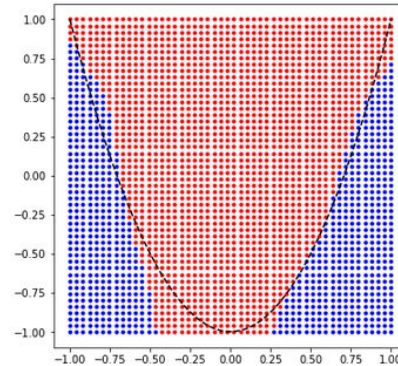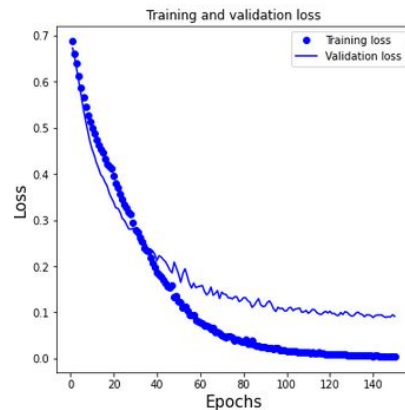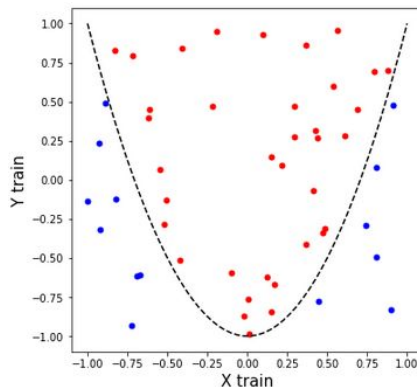
Possible problem when
- too many layers
- too many neurons
- too **few** training data

Greener et al. - 2021 A guide to machine learning for biologists
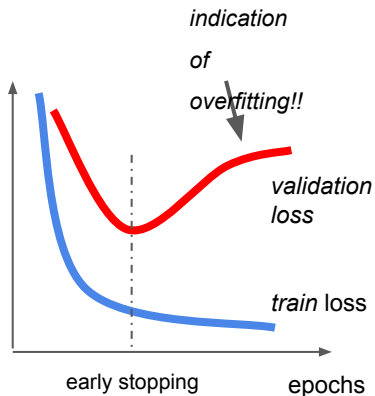
In our example:

What did we learn ?

- How a single neuron works :
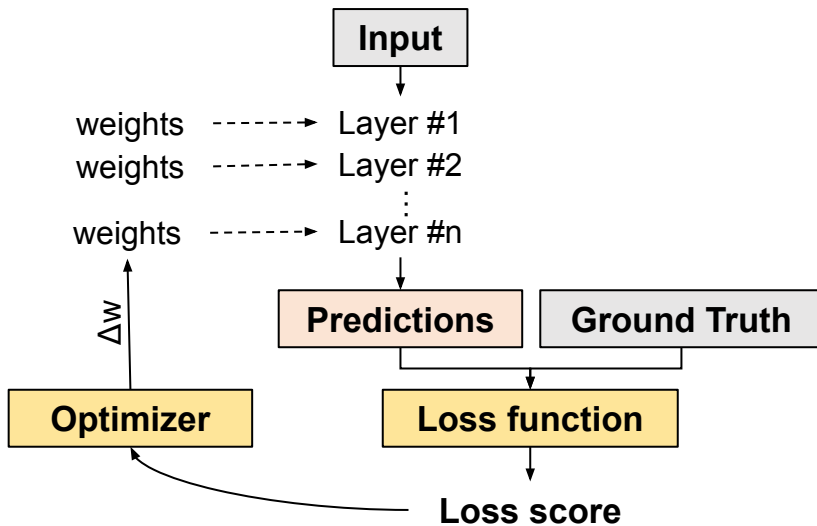
    - **Activation function** ⎤ To be defined by
    - **Input / output** ⎥ the user
    - Weights and bias

$$\phi(Xw_1 + Yw_2 + b)$$
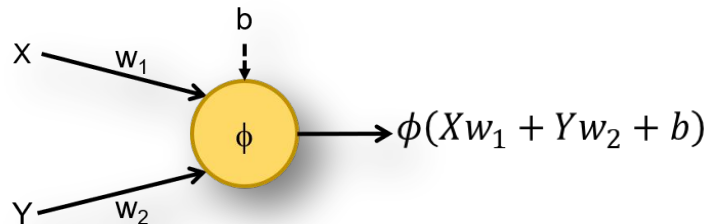
- How the training works :

    - **Loss function**
    - **Optimizer**
    - **Learning rate**

When & why using Deep Learning?

When **classic image processing/analysis tools** are not efficient or do not exist for the task we want to perform (e.g. high throughput segmentation)

Need to have enough **analyzed & good-quality** data to train the network

Need to label the data in order to get database large enough for the training	Network are trained for a specific set of data. New type of data means new training.	Deep Learning needs large computational resources for image analysis
Time consuming	**Not (always) flexible**	**Expensive (money, energy, ecology, …)**

How to start with Deep Learning (for free)?

Python 3 – open source

For DL, the open-source
TensorFlow and **PyTorch**
libraries are used.

TensorFlow

PyTorch

+

FIDLE

https://www.youtube.com/c/DigitalSreeni

https://www.youtube.com/c/CNRSFormationFIDLE?app=desktop

https://cs230.stanford.edu/lecture/

https://www.youtube.com/c/machinelearnia

jupyter CO

Colab (google)
free GPU
python jupyter