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

- What are the **fundamentals for building and training a CNN using Keras**

- Understand the **most common applications** and **where to find the tools for your applications**

Outline :

I. Deep Learning : applications for image analysis

II. General introduction & definition of neural networks

III. Example #1 : single neuron application

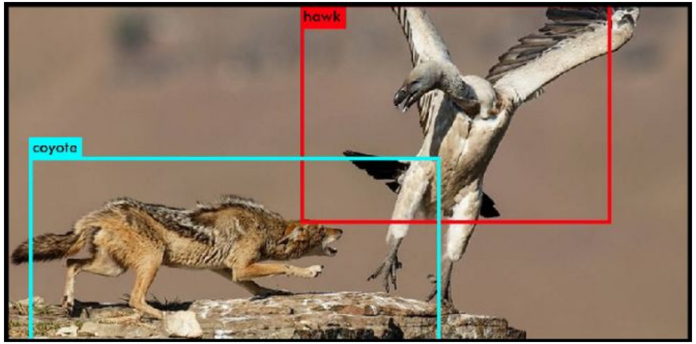
IV. Example #2: building a multi-layer network

Most common applications for image analysis:

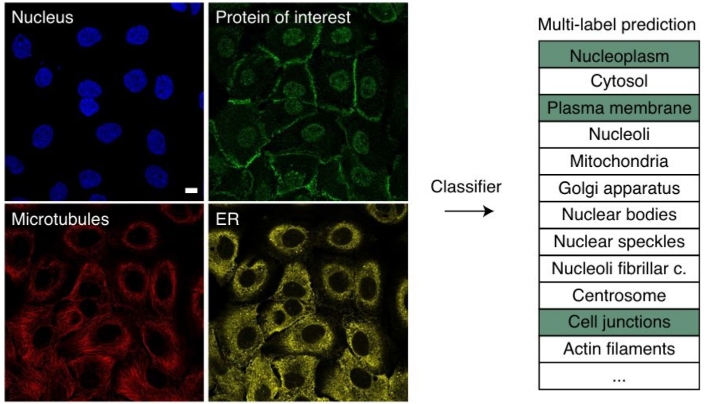
1- Image classification :

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

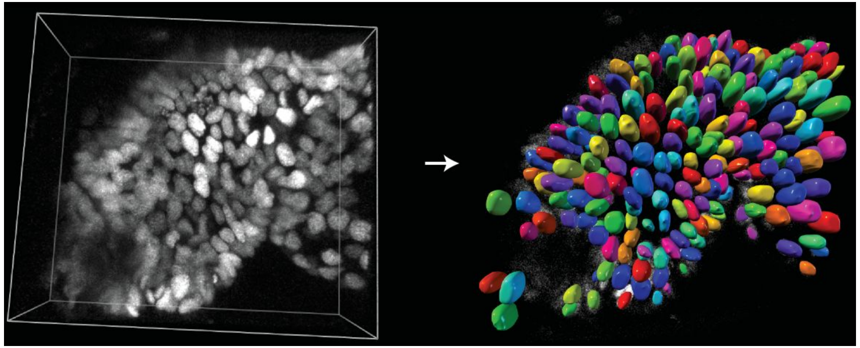


Ouyang et al. - 2019 Analysis of the Human Protein Atlas Image Classification competition.

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

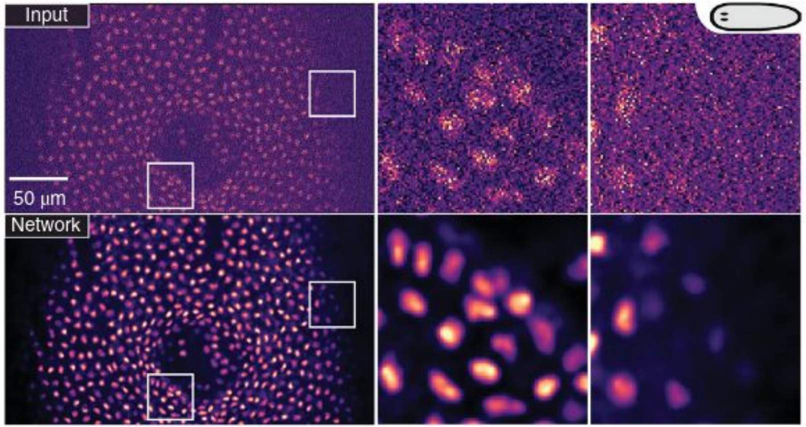
https://github.com/MouseLand/cellpose - Stringer et al. 2021 Cellpose: a generalist algorithm for cellular segmentation

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

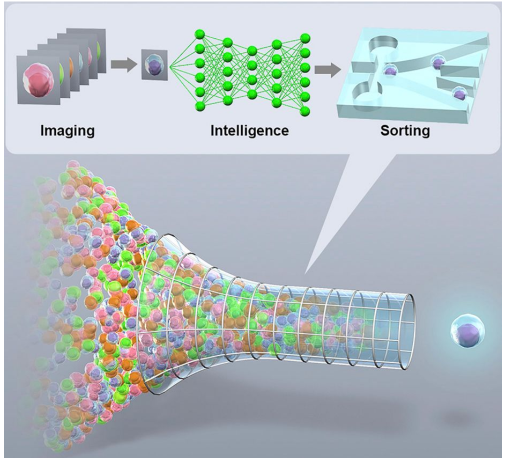
https://github.com/vanvalenlab/intro-to-deepcell - Greenwald at al. - 2022 Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning

Most common applications for image analysis:

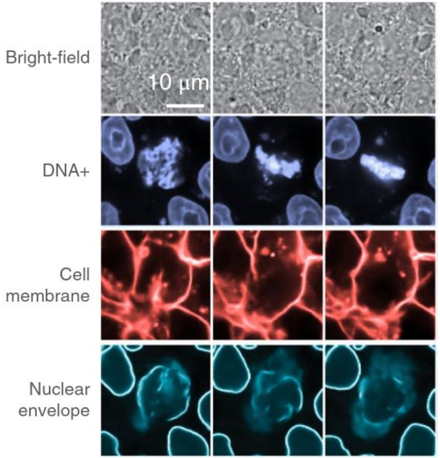
1- Image classification :
2- Image segmentation :
3- Augmented microscopy :



https://github.com/CSBDeep/CSBDeep - Weigert et al. 2017. Content-aware image restoration: pushing the limits of fluorescence microscopy



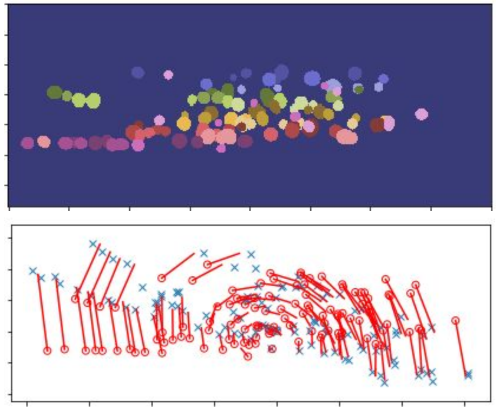
Nitta et al. 2018. Intelligent Image-Activated Cell Sorting



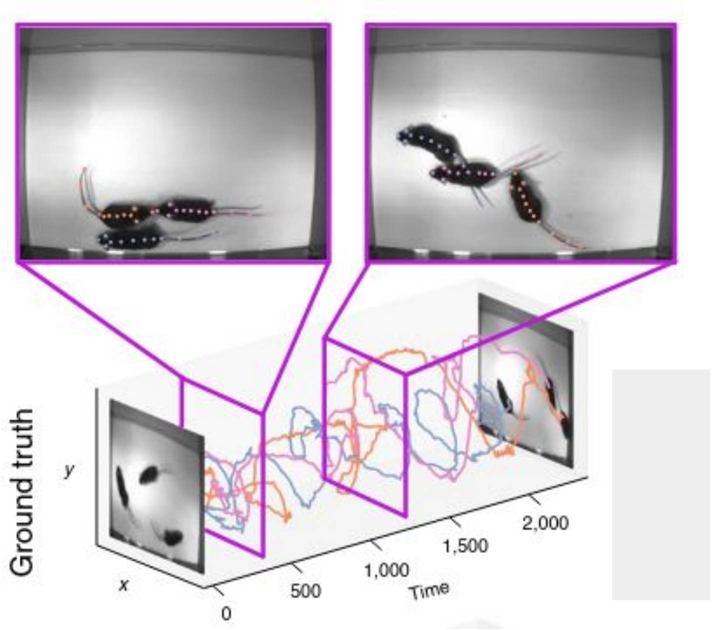
Ounkomol et al. 2018. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy

Most common applications for image analysis:

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

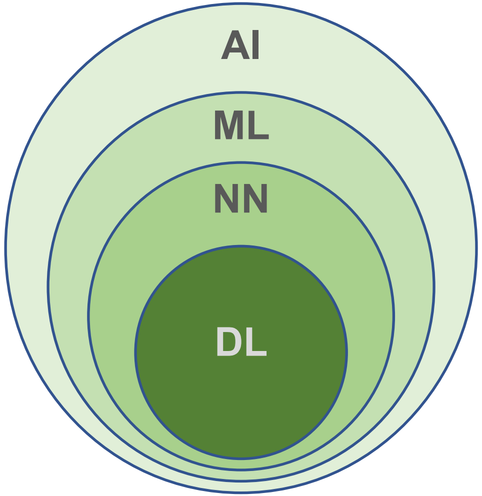


Wen et al. - 2021 3DeeCellTracker, a deep learning-based pipeline for segmenting and tracking cells in 3D time lapse images

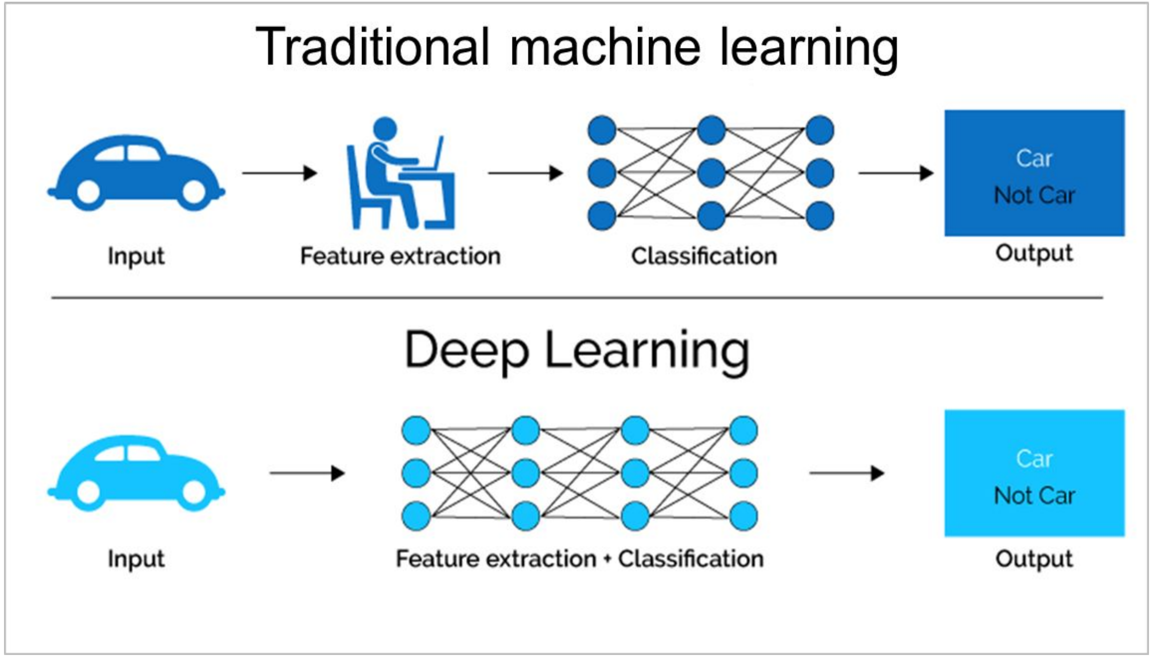


Lauer et al. - 2022 Multi-animal pose estimation, identification and tracking with DeepLabCut

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

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

Time consuming

Network are trained for a specific set of data. New type of data means new training.

Not (always) flexible

Deep Learning needs large computational resources for image analysis

Expensive

How to start with Deep Learning (for free)?

Python 3 – open source

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

Colab (google)
free GPU
python jupyter

#ZeroCostDL4Mic

https://csbdeep.bioimagecomputing.com/

https://github.com/HenriquesLab/ZeroCostDL4Mic

+

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

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

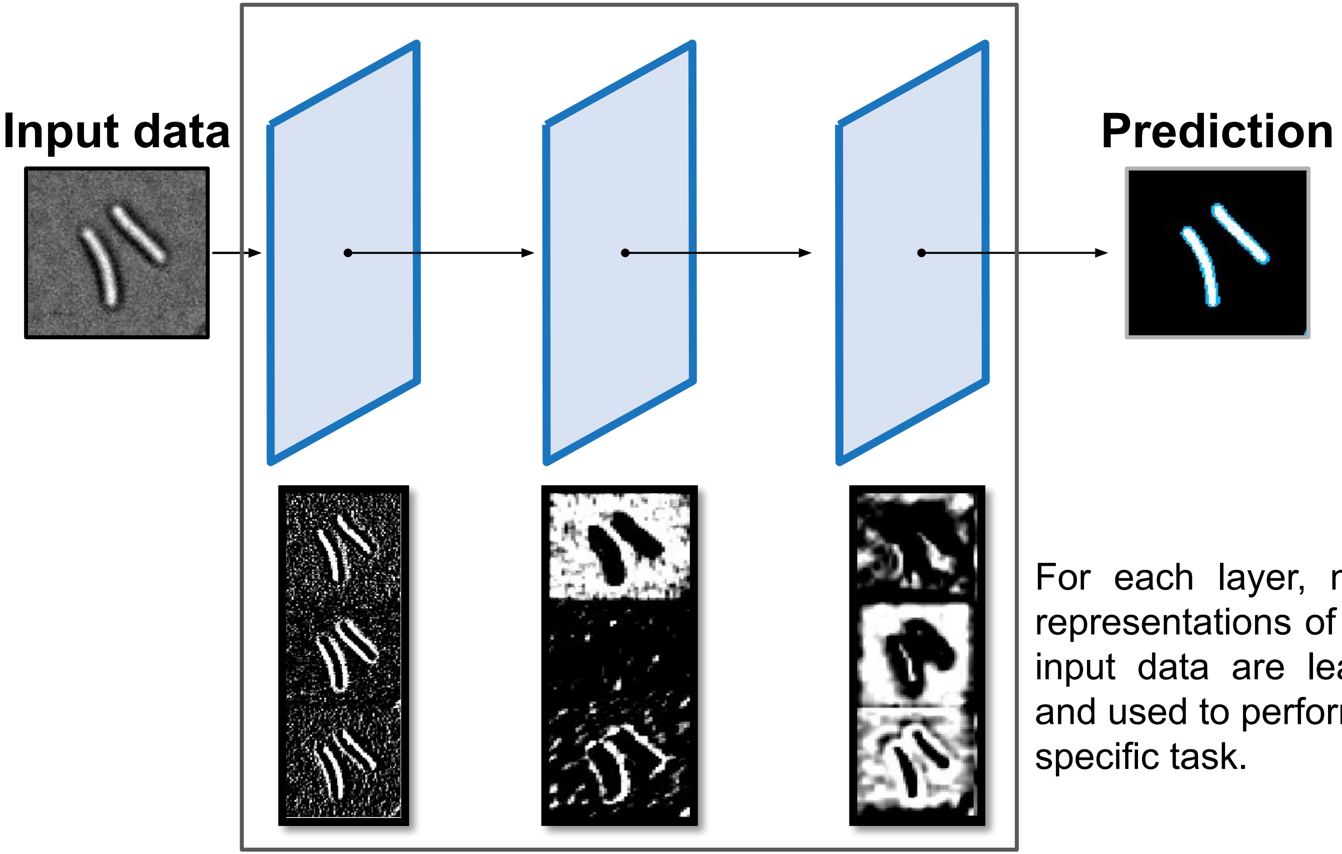
https://cs230.stanford.edu/lecture/

https://bioimage.io/#/

https://www.kaggle.com/

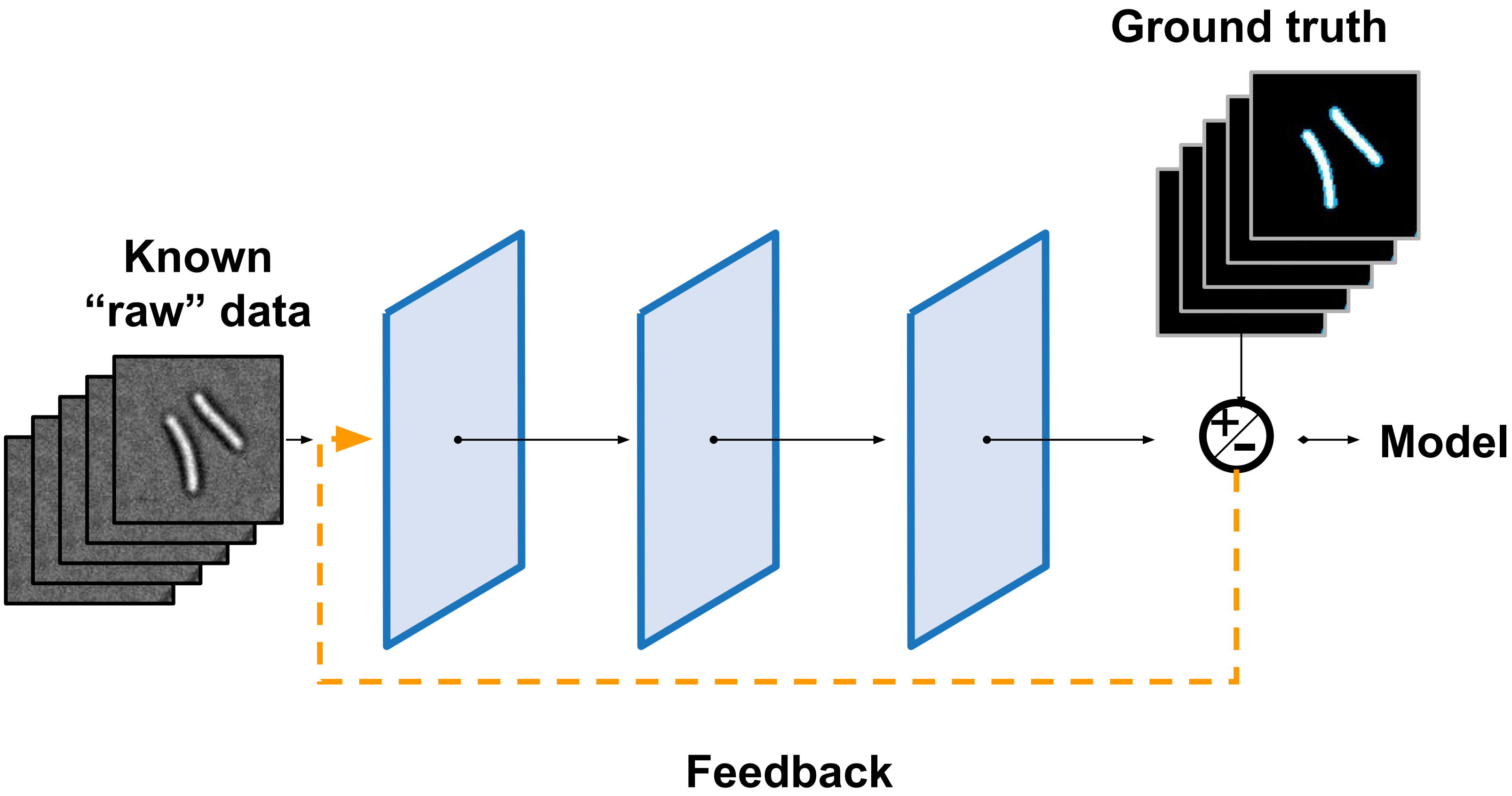
https://bbbc.broadinstitute.org/image_sets

Deep Learning : why "Deep"?

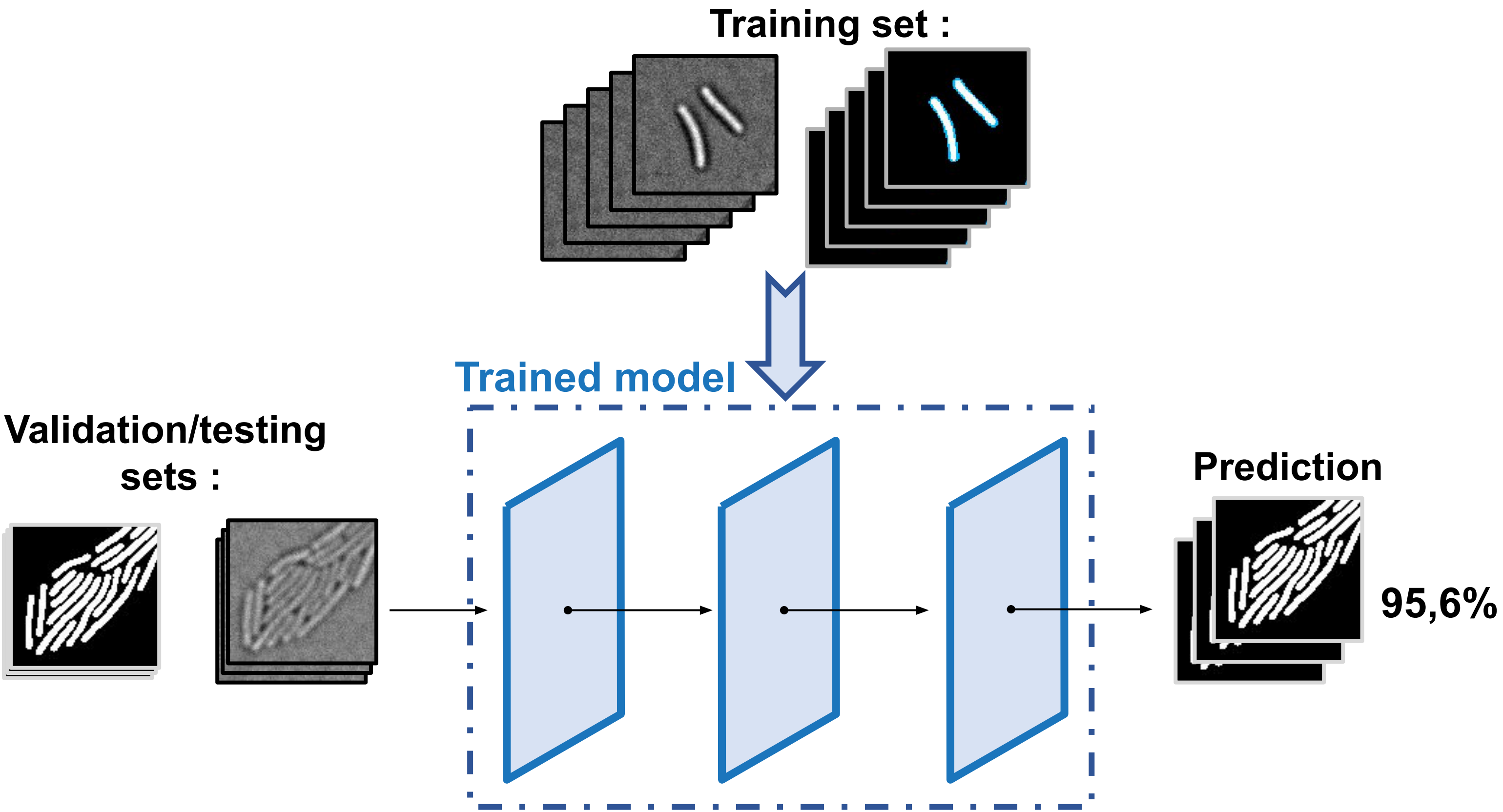


For each layer, new representations of the input data are learnt and used to perform a specific task.

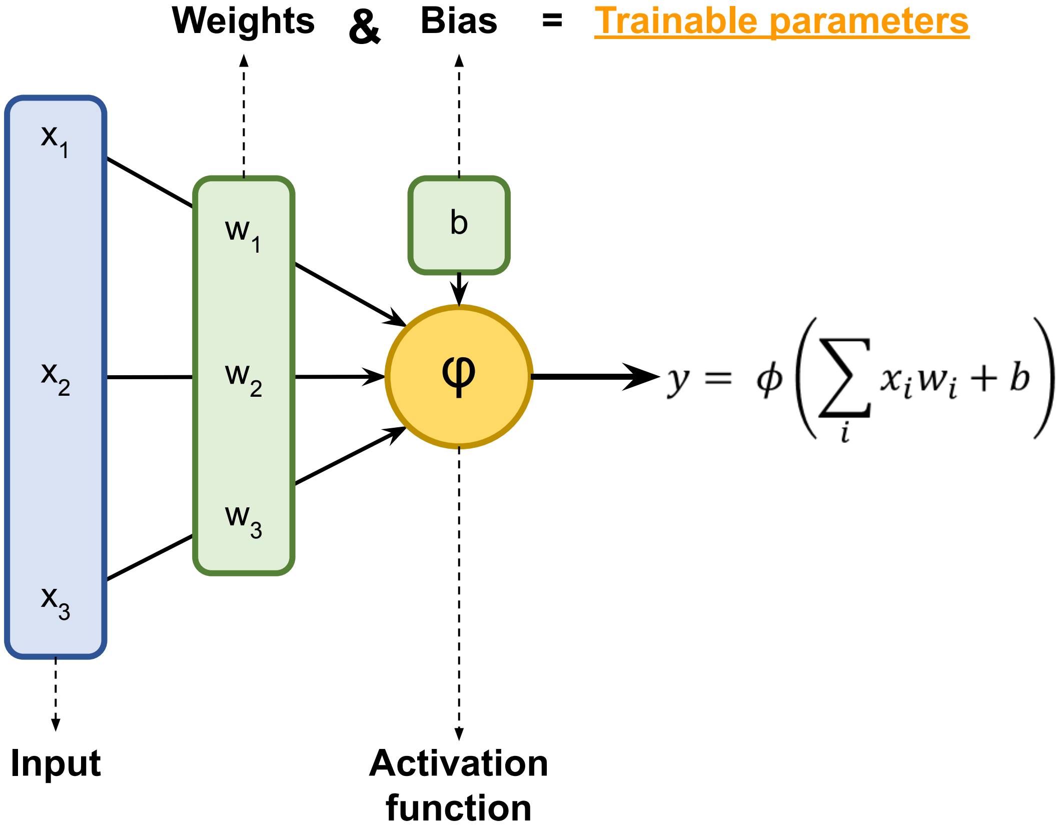
“Learning” under supervision :



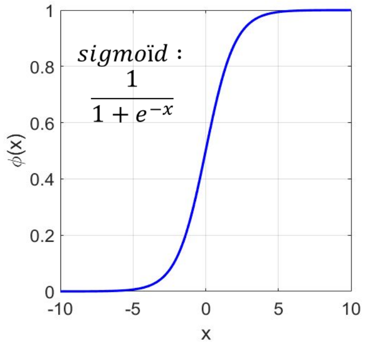
Supervised deep learning network :

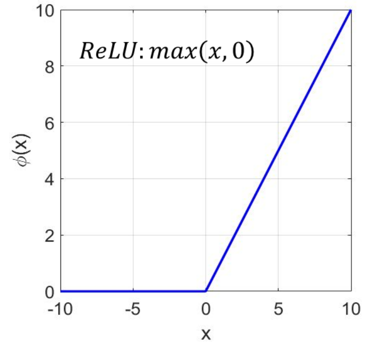


Definition of a single neuron



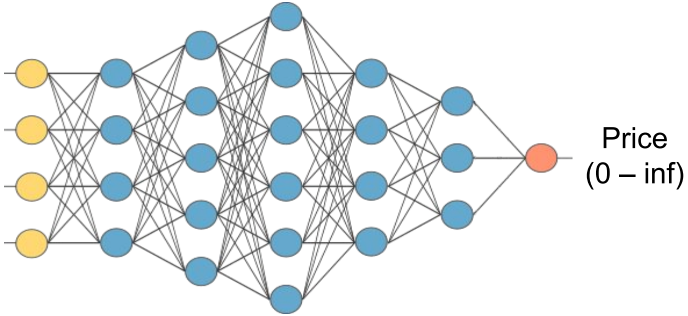
$$y = \phi\left(\sum_i x_i w_i + b\right)$$

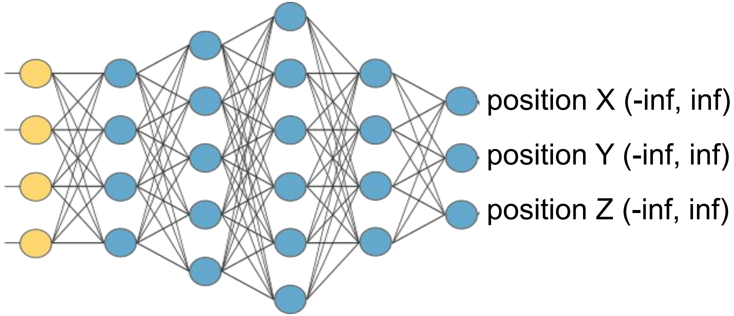




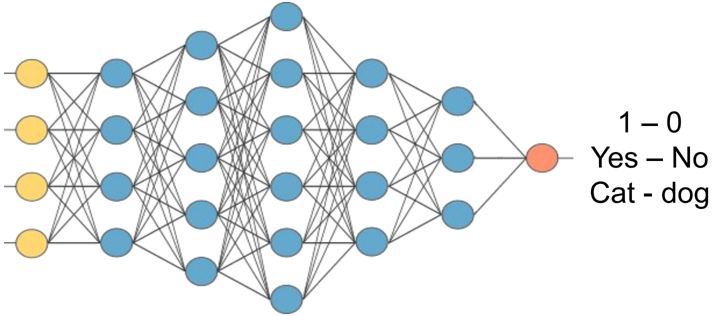
Regression vs. Classification

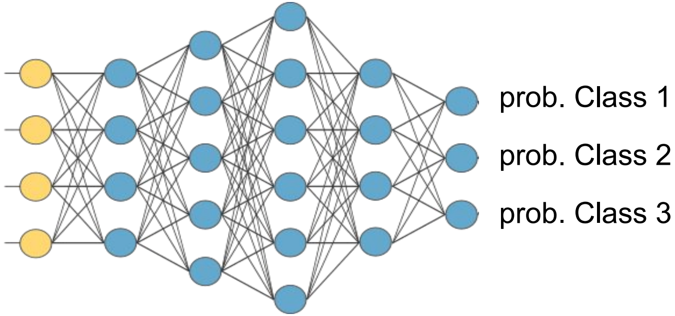
Regression : output is one or more real numbers



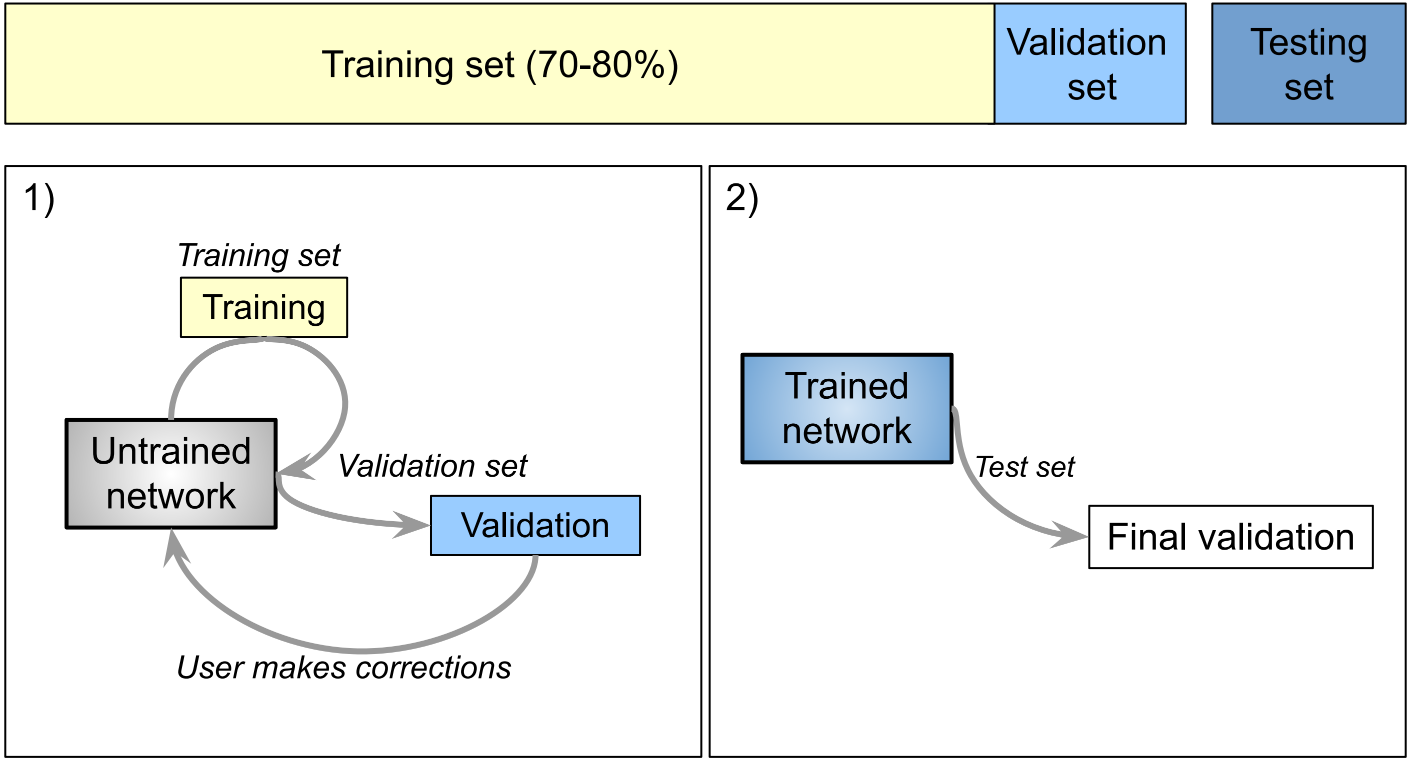


Classification : output is the probability that input belong to one or more classes





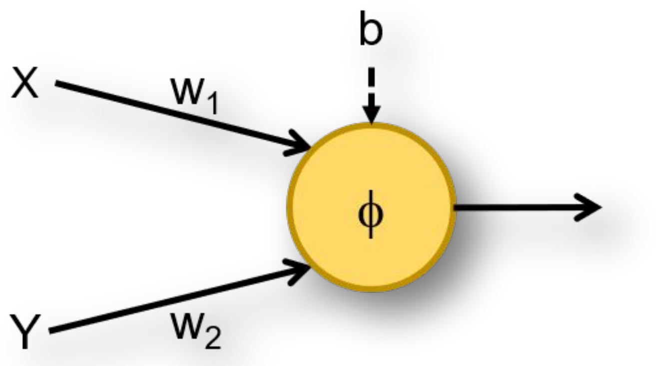
Training, testing and validation sets



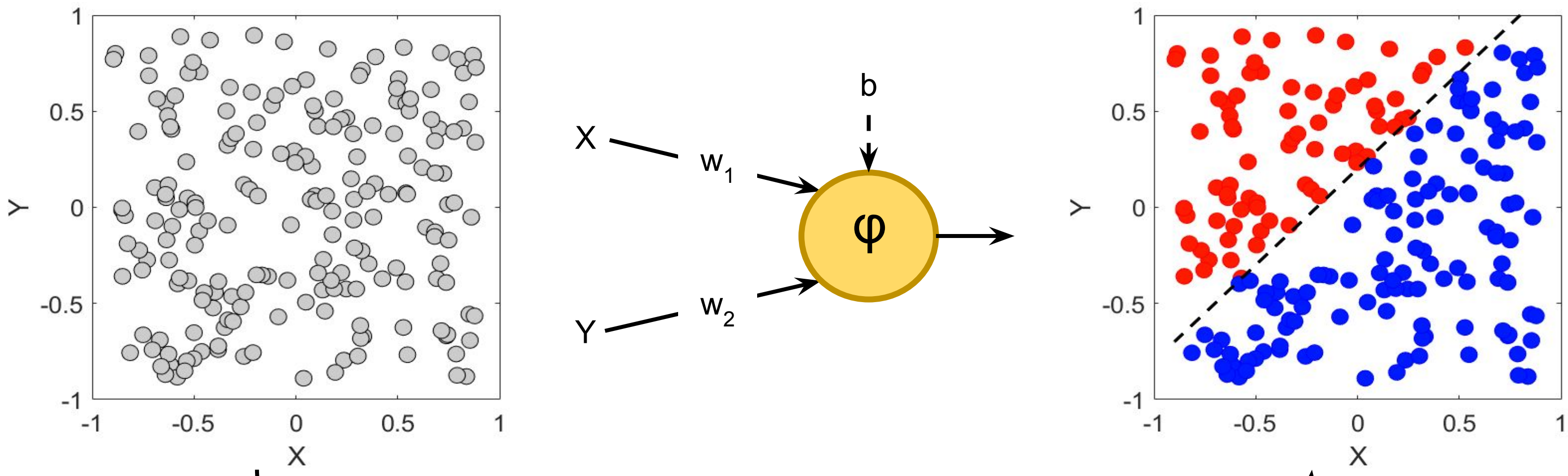
Train a single neuron classifier

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



<u>Input data</u>:
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

<u>Prediction :</u>
class red or
blue

● = 0
● = 1

Definition of a classifier with Keras

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'])
```

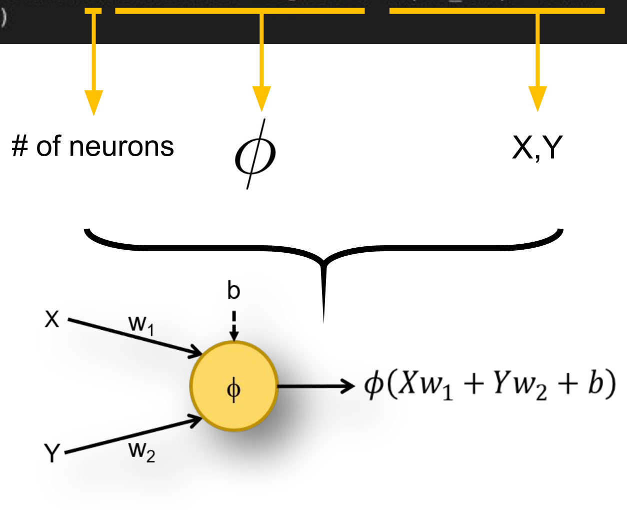
3- Training

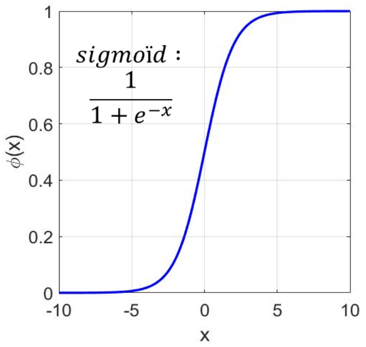
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```

Definition of a single neuron

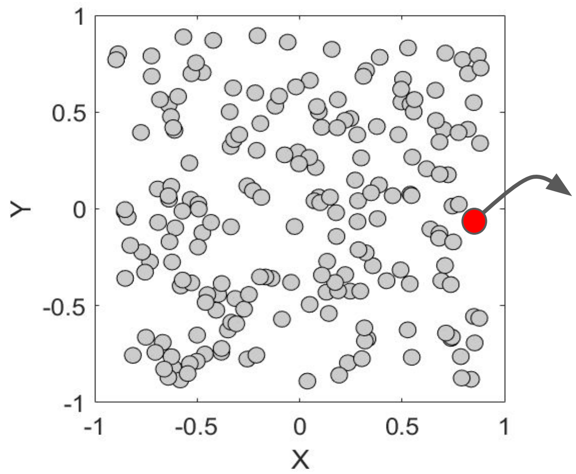
1- Definition of the network architecture

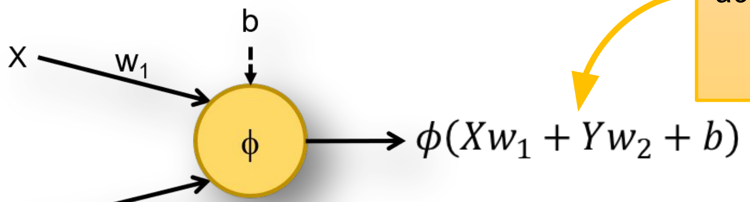
```python
from keras import models
from keras.models import Sequential
from keras.layers import Dense

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



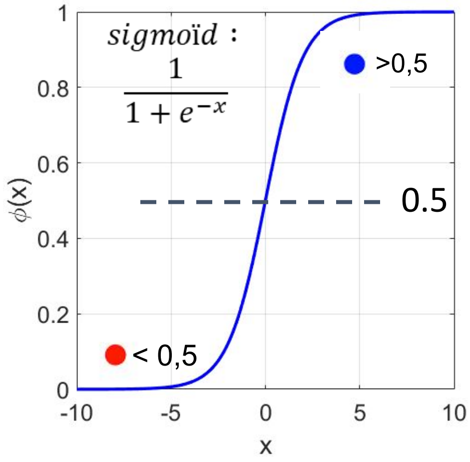


How the neuron works ?





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



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

Definition of the training options

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,))
])
```

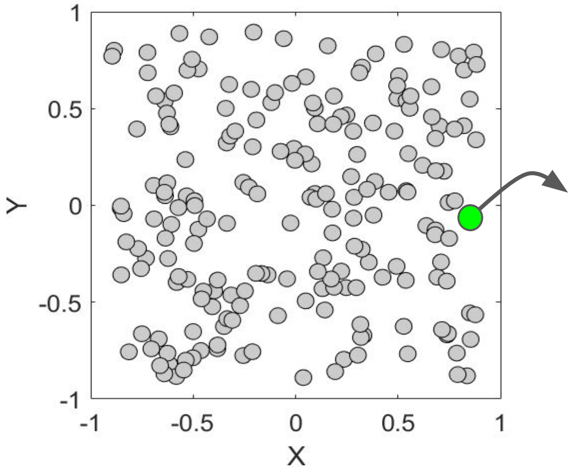
2- Definition of the training options

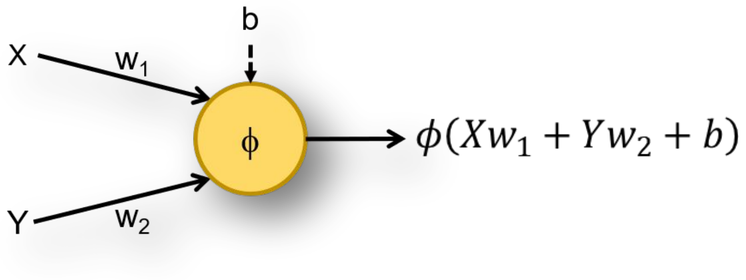
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

3- Training

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

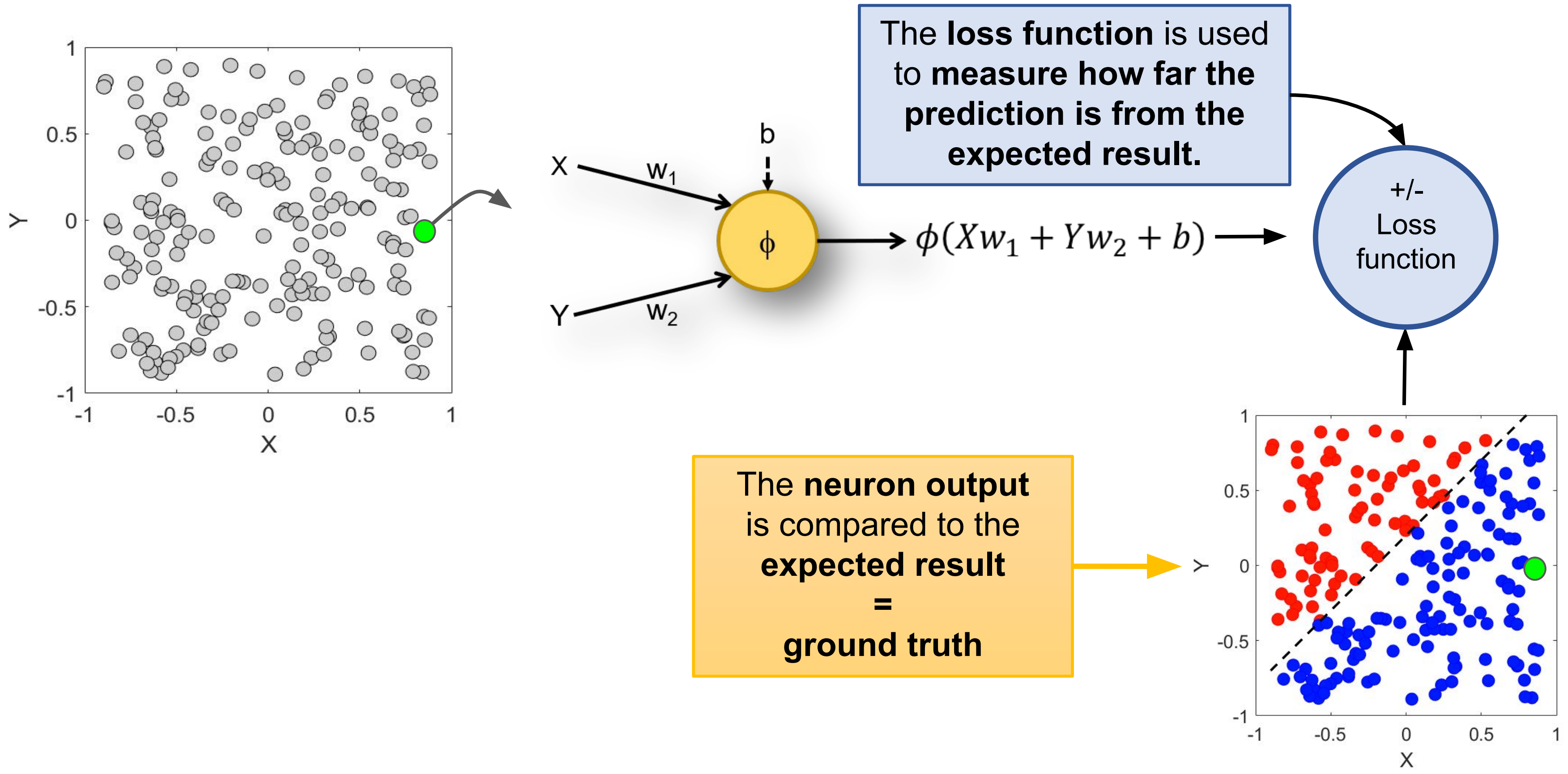
Compiling : defining training process



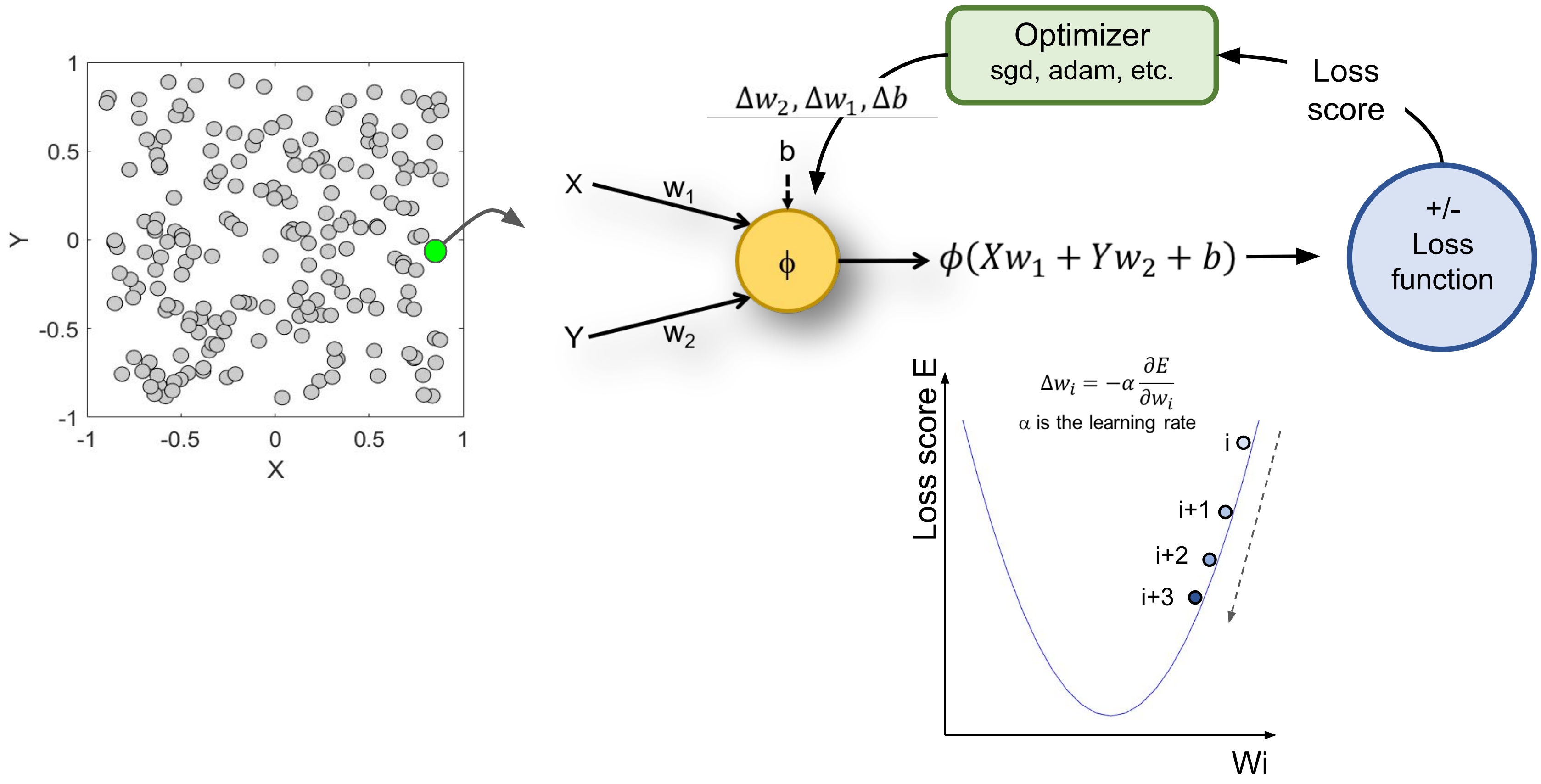


Training set

Compiling : defining training process



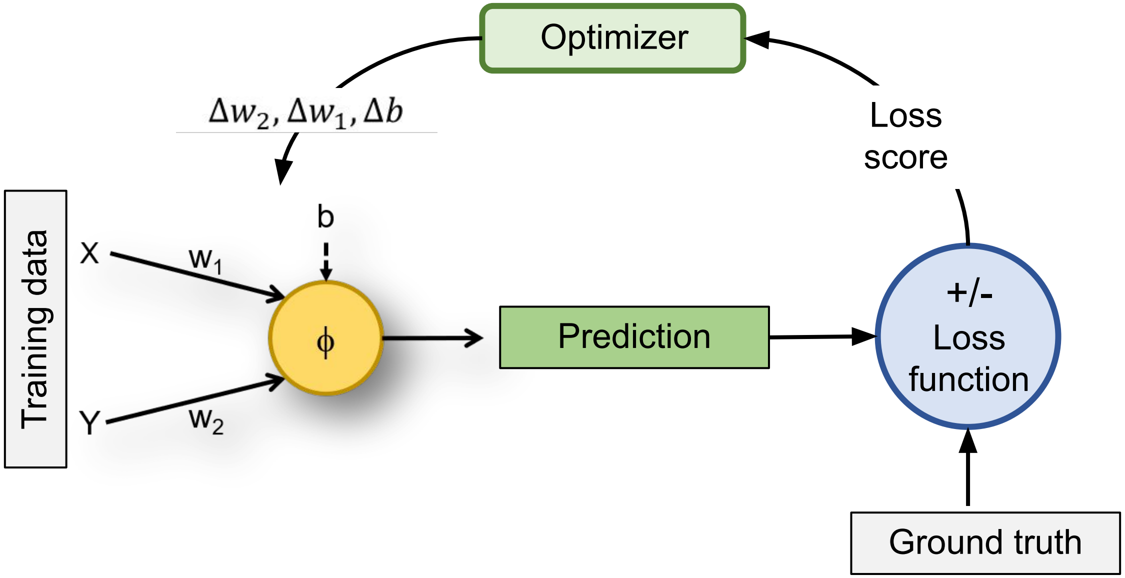
Compiling : defining training process



Model compiling

2- Definition of the training options

```
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

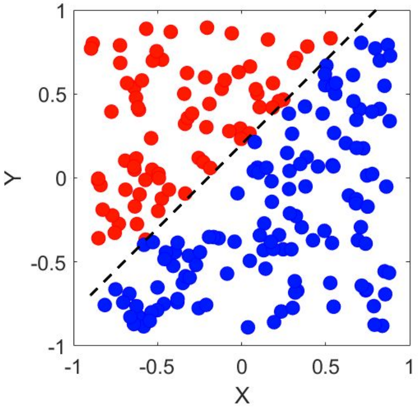
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```

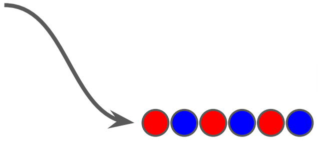
Start the training

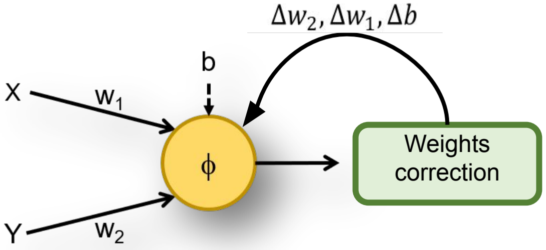
3- Training

```
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)







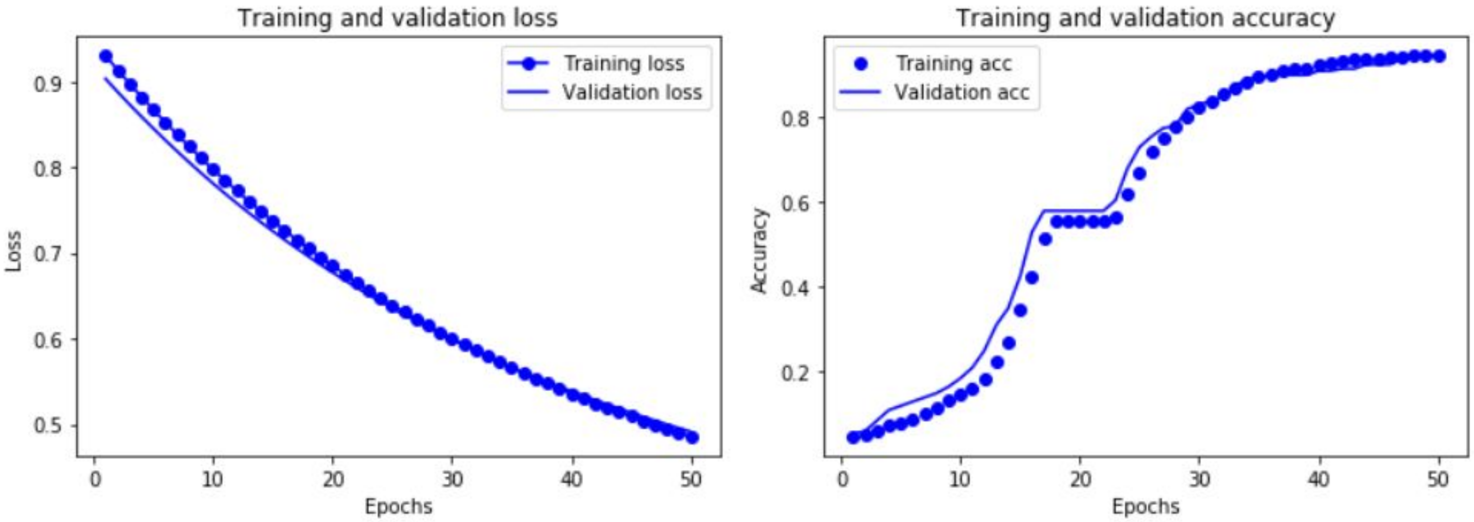
1 *epoch*
=
all points used once

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))
```



Summary

1- Definition of the network architecture

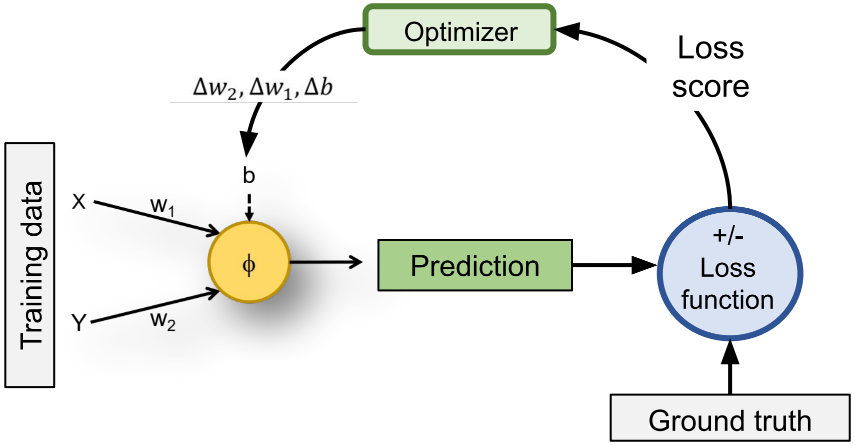
```
from keras import models
from keras.models import Sequential
from keras.layers import Dense

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

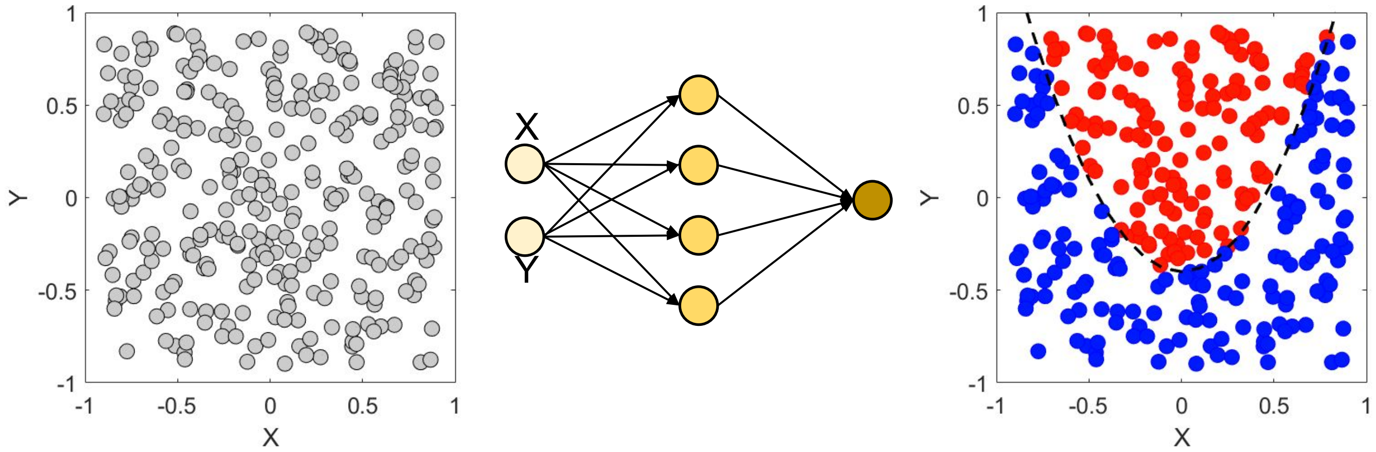
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 100,
                    validation_data = (Validation_data, Validation_label))
```



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

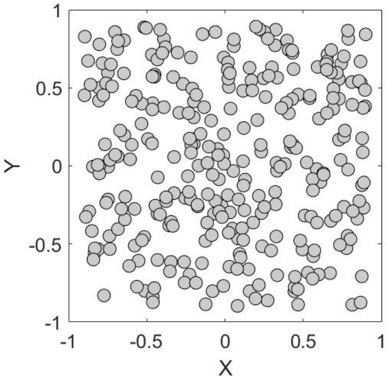
1st layer needs input specified

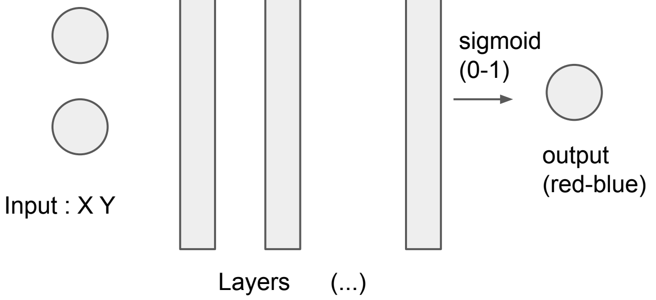
number of
neurons

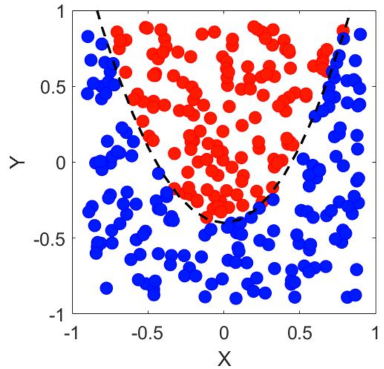
1st layer

2nd layer
…

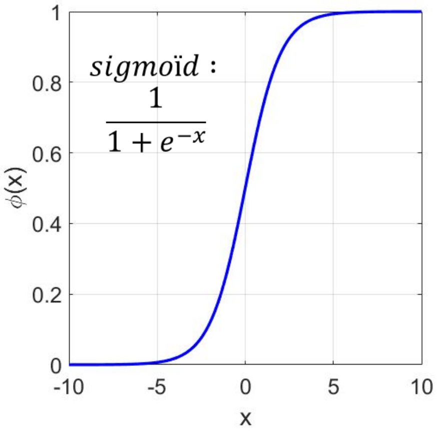
```python
from keras.models import Sequential
from keras.layers import Dense

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'])
```

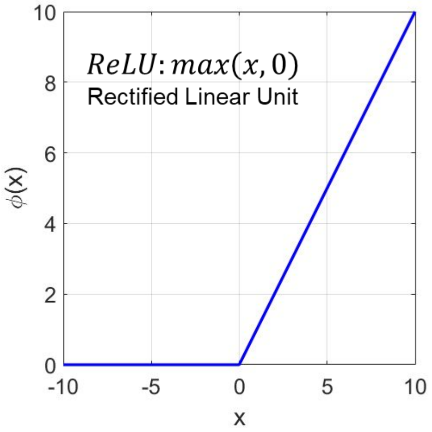






The two most popular activation function





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

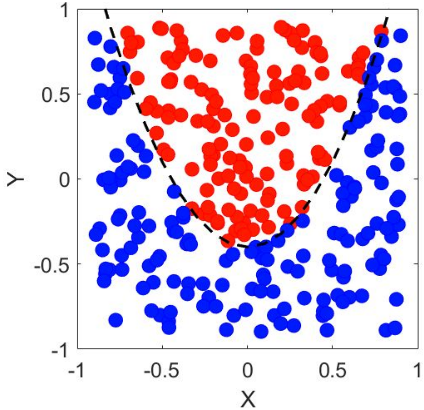
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 150,
                    batch_size = 8,
                    validation_data = (Validation_data, Validation_label))
```

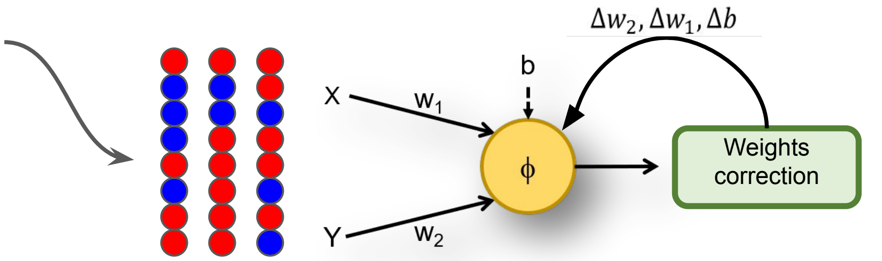
Good practice to use a power of 2

Training

data (x1,y1),(x2,y2),…,(xn,yn)

label (red) , (red) ,…, (blue)

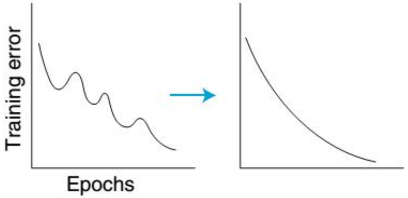




one ***batch*** at a time

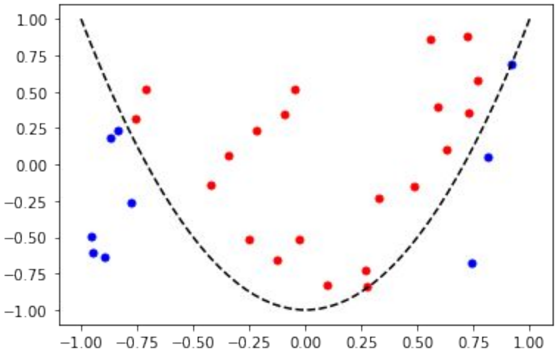
the weights are recalculated **once every batch**

The weights are adjusted **N/batch_size**

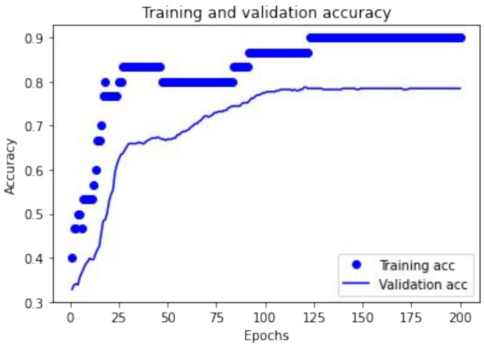


Pic credit : Moen et al. 2019. Nat. Meth.

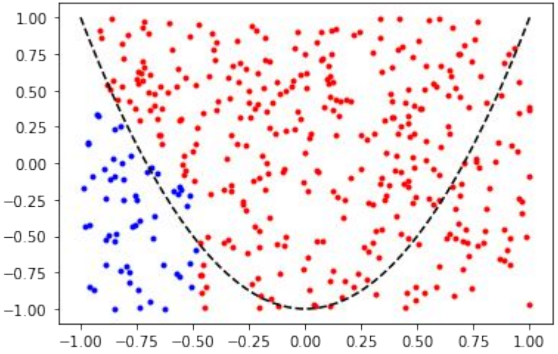
Effect of imbalanced classes



Training set

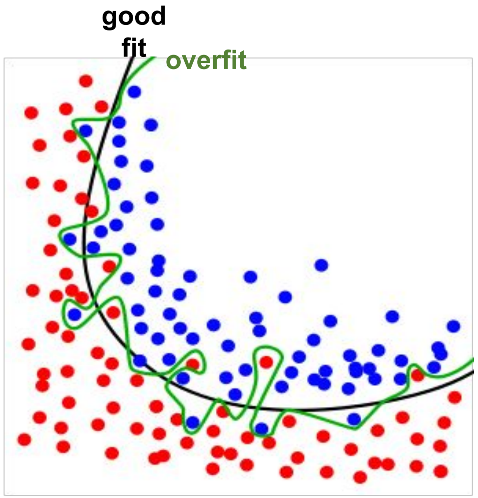


Accuracy ~ 0.8



Prediction on test set

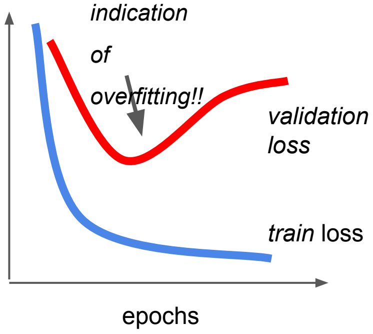
Overfitting



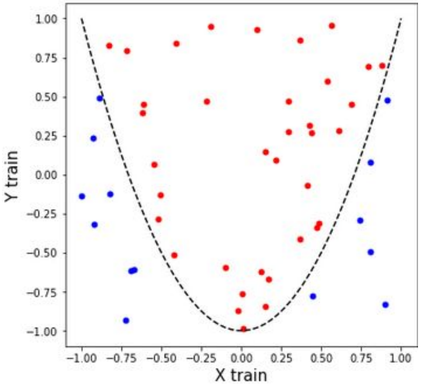
the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably

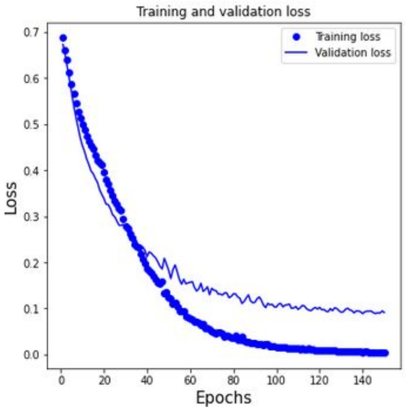
Possible problem when

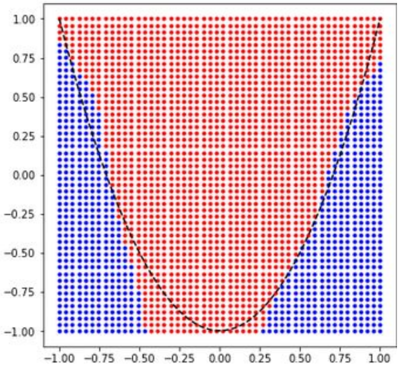
- too many layers
- too many neurons
- too **few** training data



In our example:



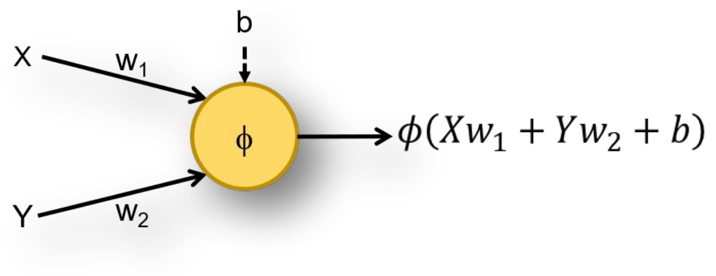




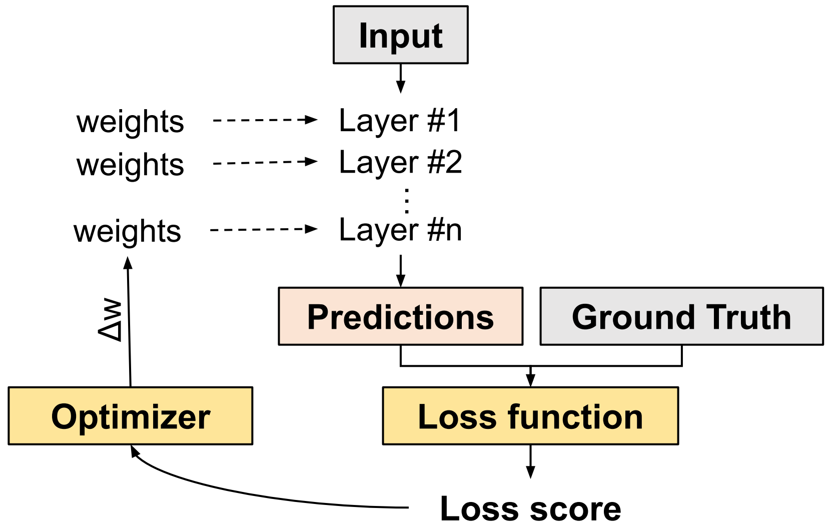
What did we learn ?

- How a single neuron works :
  - **Activation function**
  - **Input / output**
  - Weights and bias

To be defined by the user



- How the training works :
  - **Loss function**
  - **Optimizer**
  - **Learning rate**



How to choose the activation and loss functions ?

Problem type	Last-layer activation	Loss function	Number of neurons in the last layer
Binary classification	‘sigmoid’	‘binary-crossentropy’	1
Multiclass, single-label classification	‘softmax’	‘categorical_crossentropy’	As many as the number of classes
Regression to values between 0 and 1	‘sigmoid’ or ‘none’	‘mse’	1

As a rule-of-thumbs, <u>use ‘relu’ everywhere else as activation function.</u>