Deep Learning for image analysis
Part I - Fundamentals

JB Fiche, CBS-Montpellier & Plateforme MARS-MRI
Francesco Pedaci, CBS-Montpellier
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

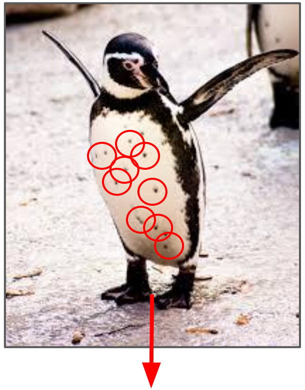
II. General introduction & definition of neural networks

III. Example #1 : application of a single neuron

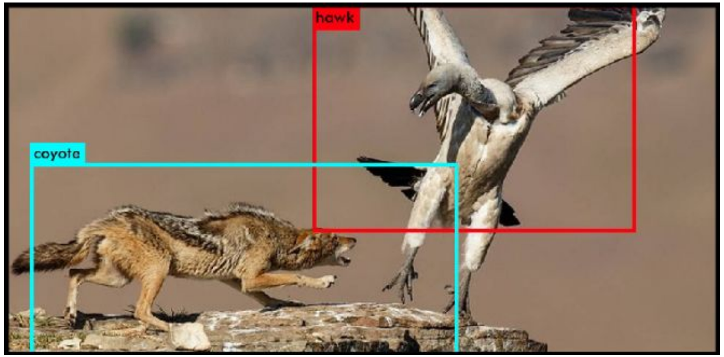
Most common applications for image analysis:

1- Image classification :

PlantNet



ID = ‘Skipper’



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

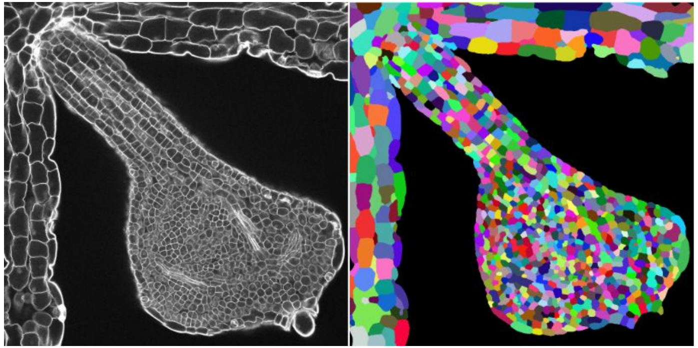
Moen et al. 2019. Deep learning for cellular image analysis

Most common applications for image analysis:

1- Image classification :

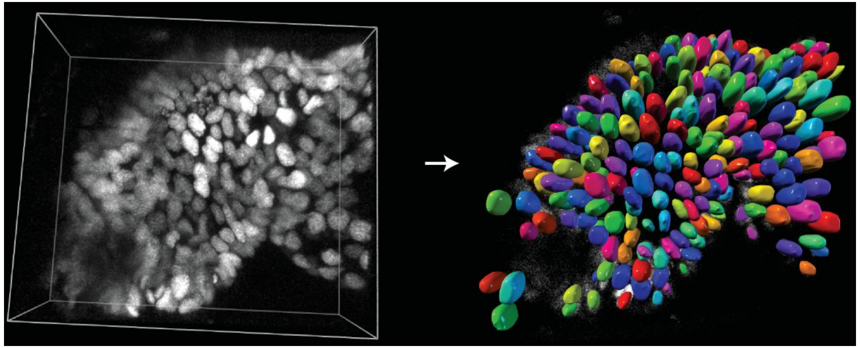
2- Image segmentation :

2D segmentation of plant cells using membrane staining



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

3D segmentation of nuclei in tissue

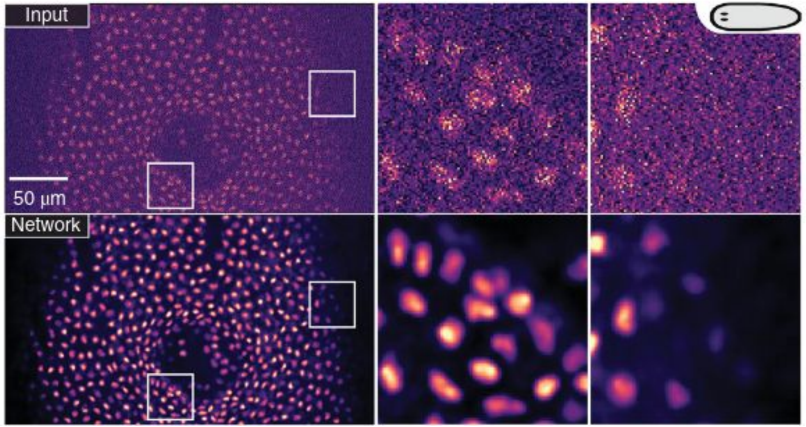


https://github.com/stardist/stardist - Schmidt et al. 2018. Cell Detection with Star-Convex Polygons

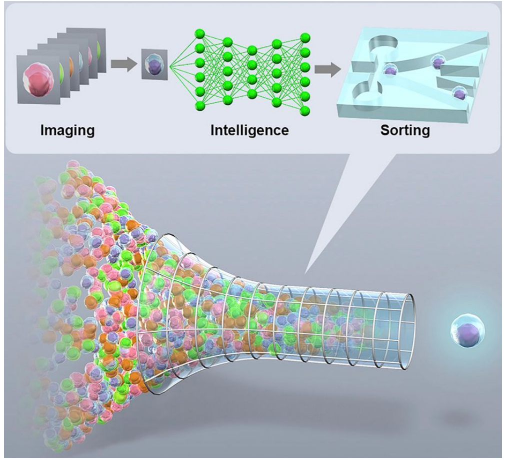
Minaee et al. 2020. Image segmentation using Deep Learning : a survey

Most common applications for image analysis:

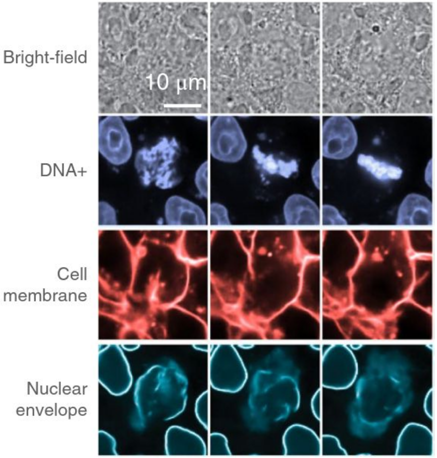
1- Image classification :
2- Image segmentation :
3- Augmented microscopy :



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

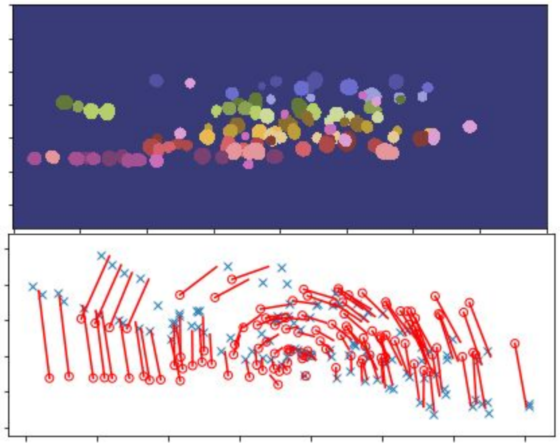


Ounkomol et al. 2018. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy

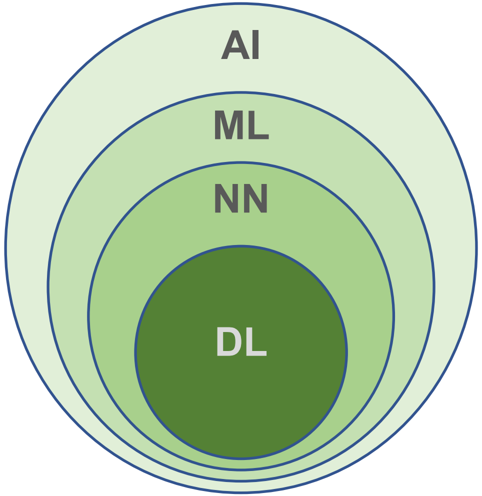
Most common applications for image analysis:

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

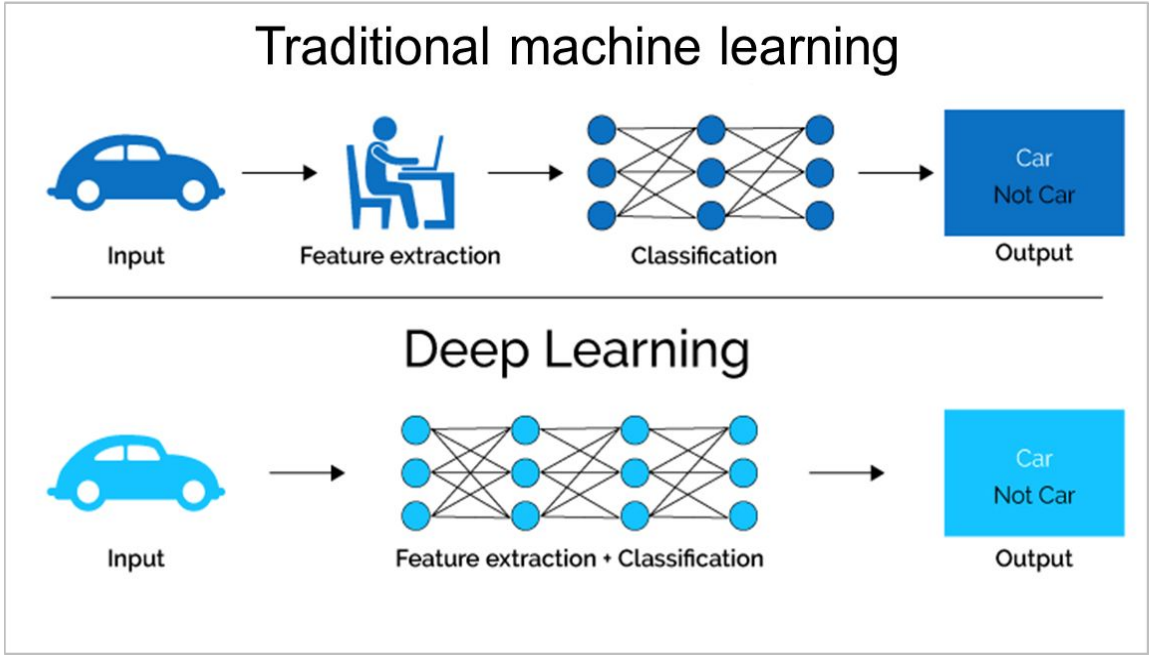
Moen et al
Wen et al (3DeeCellTracker)



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 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?

Matlab 2018 version and later

Python 3 – open source

For DL, the open-source **TensorFlow** library from Google is used.

TensorFlow

Colab (google)
free GPU
python jupyter

+

CSBDeep https://csbdeep.bioimagecomputing.com/

#ZeroCostDL4Mic https://github.com/HenriquesLab/ZeroCostDL4Mic

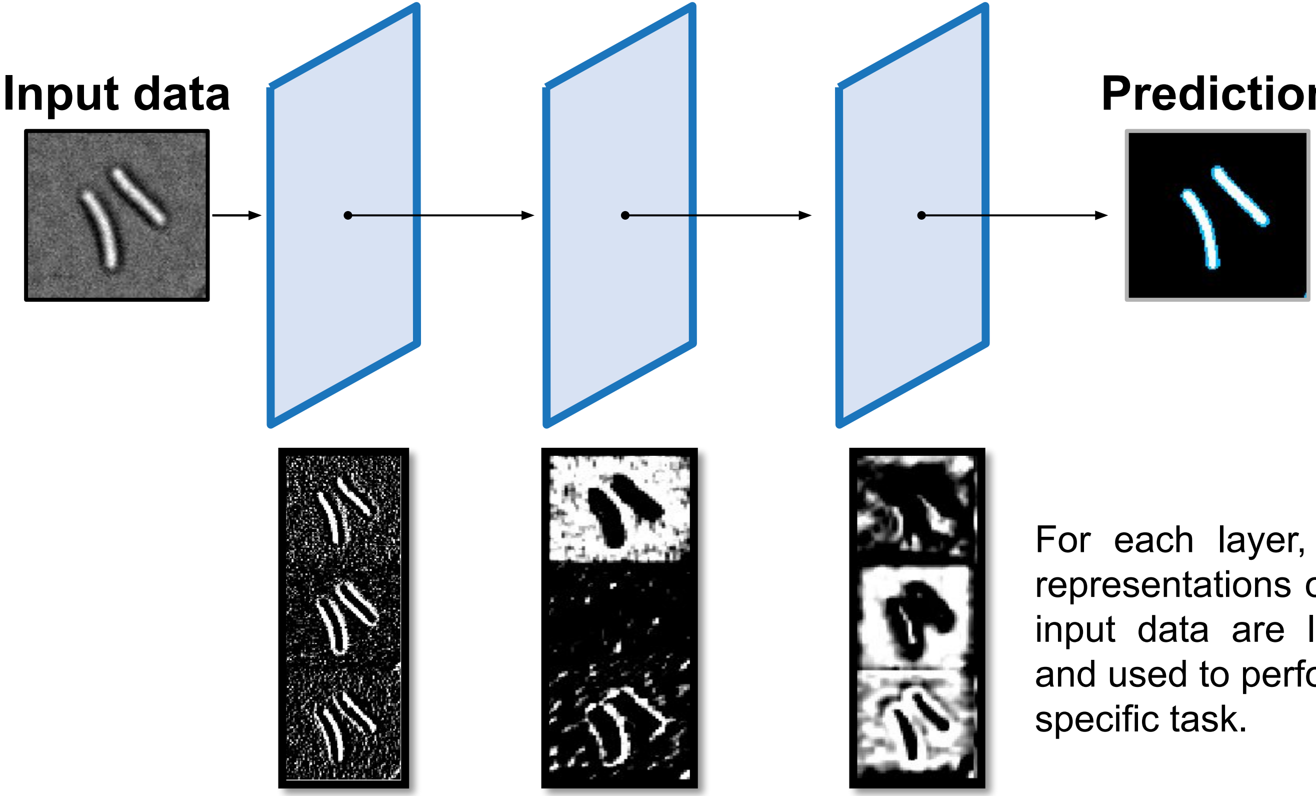
https://imjoy.io/

BioImage.IO https://bioimage.io/#/

Deep Learning Specialization
Become a Deep Learning expert. Master the fundamentals of deep learning and break into AI.
4.9 112,392 ratings
Andrew Ng +2 more instructors TOP INSTRUCTORS Most popular in Machine Learning

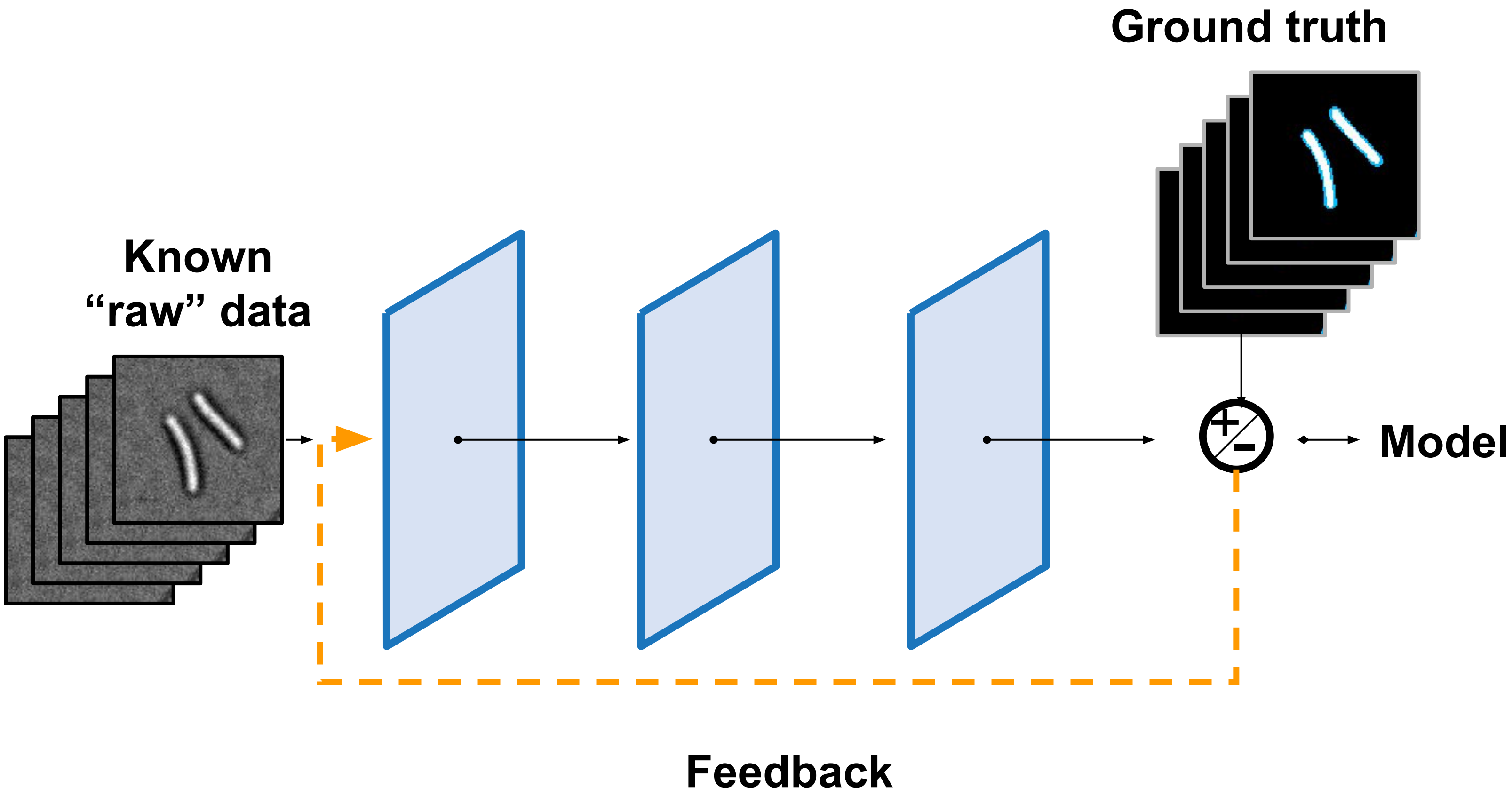
coursera

Deep Learning : why “Deep”?

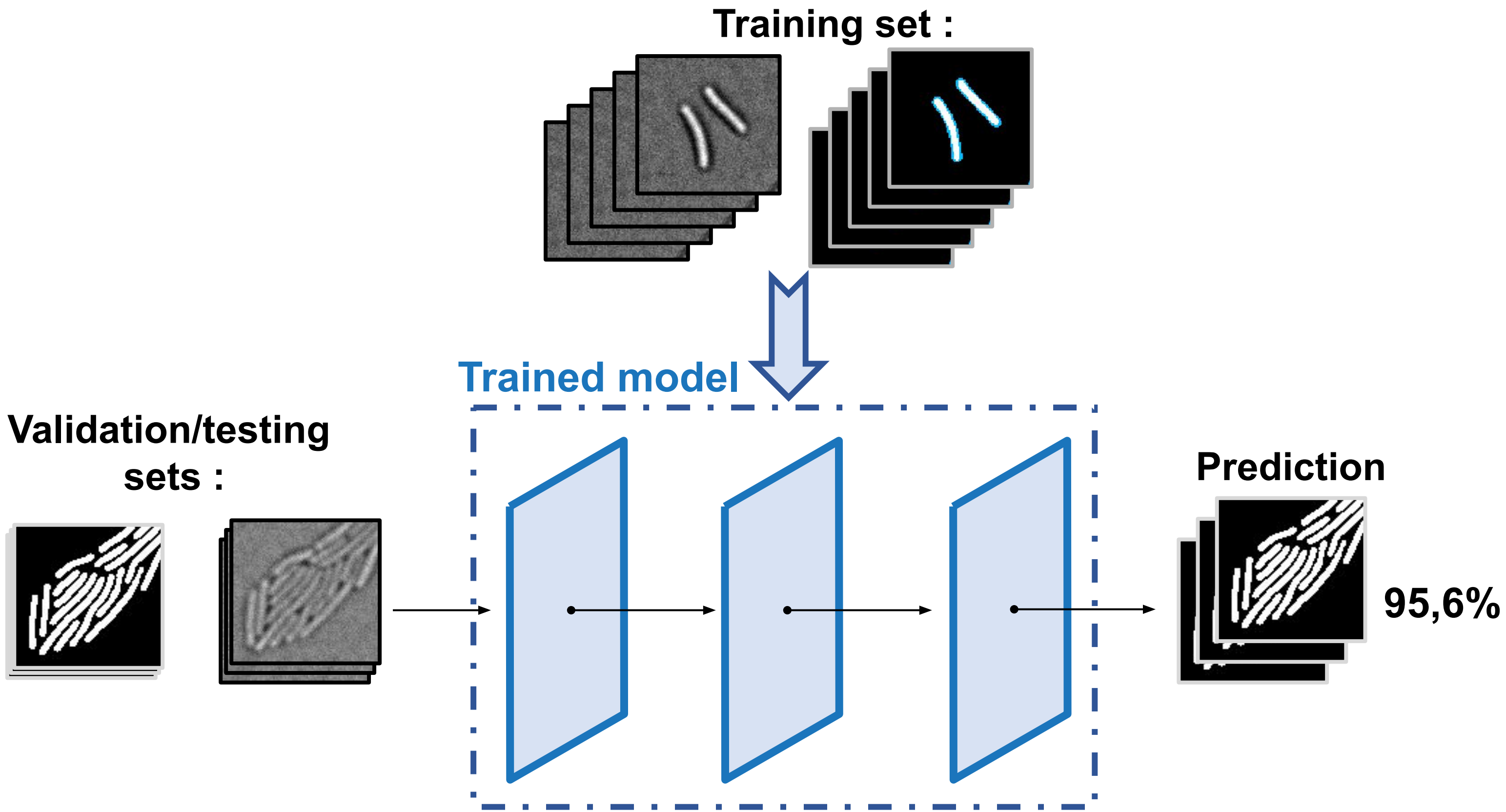


For each layer, new representations of the input data are learnt and used to perform a specific task.

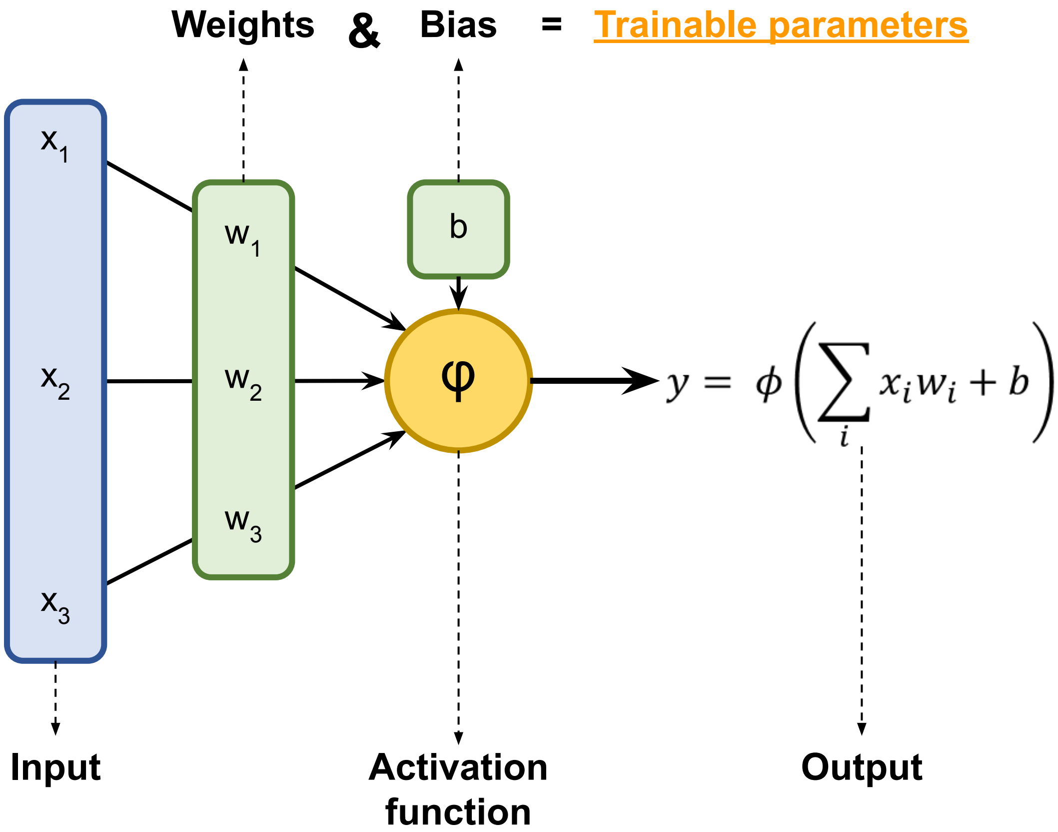
“Learning” under supervision :



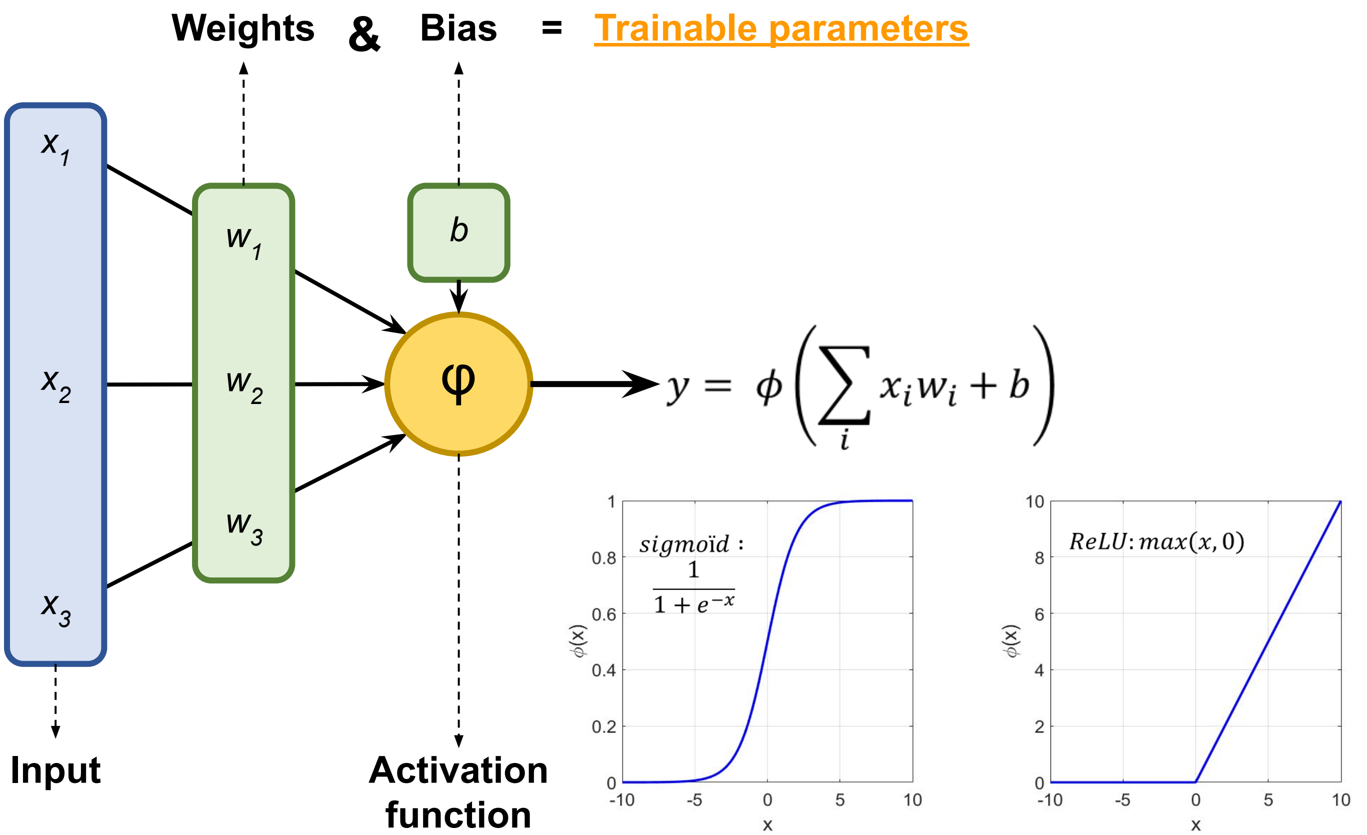
Supervised deep learning network :



Definition of a single neuron

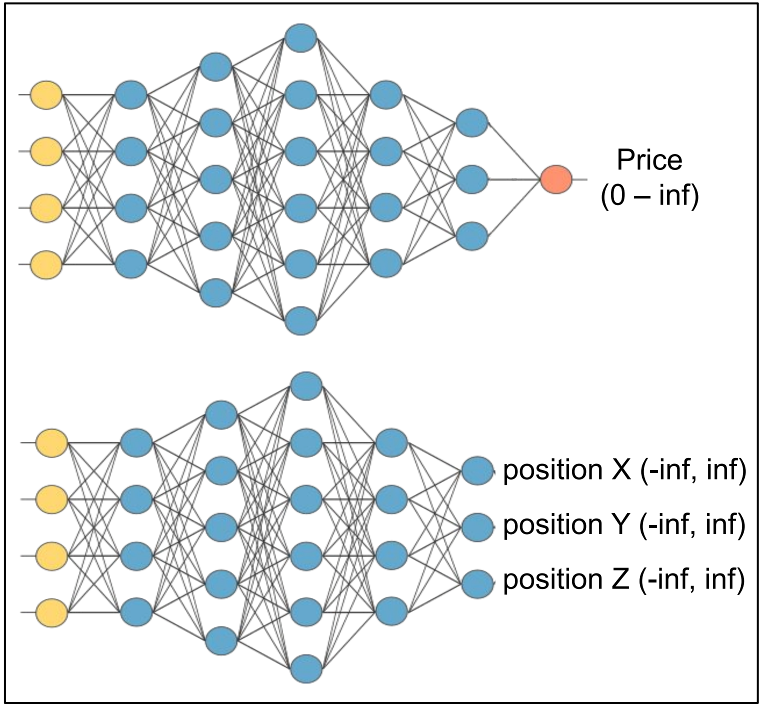


Definition of a single neuron

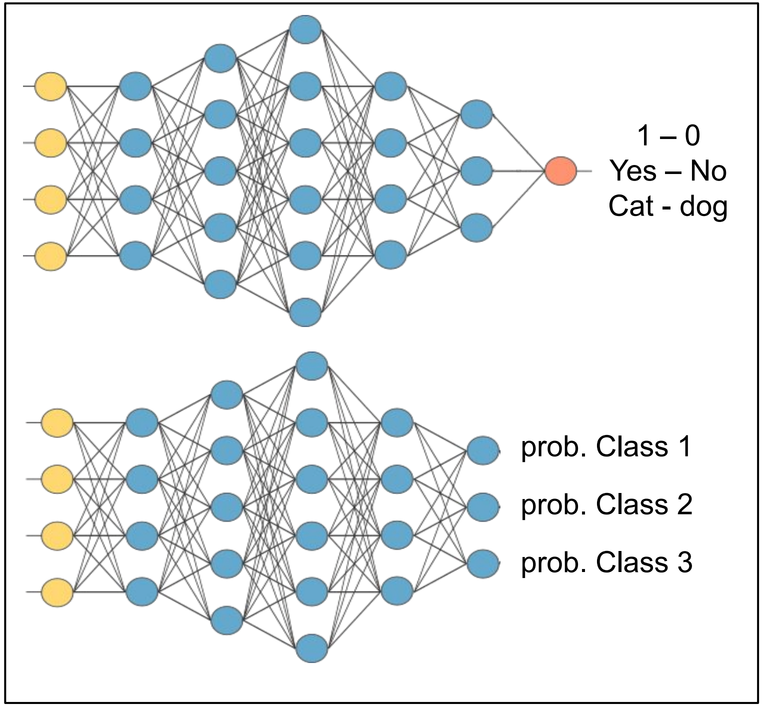


Regression vs. Classification

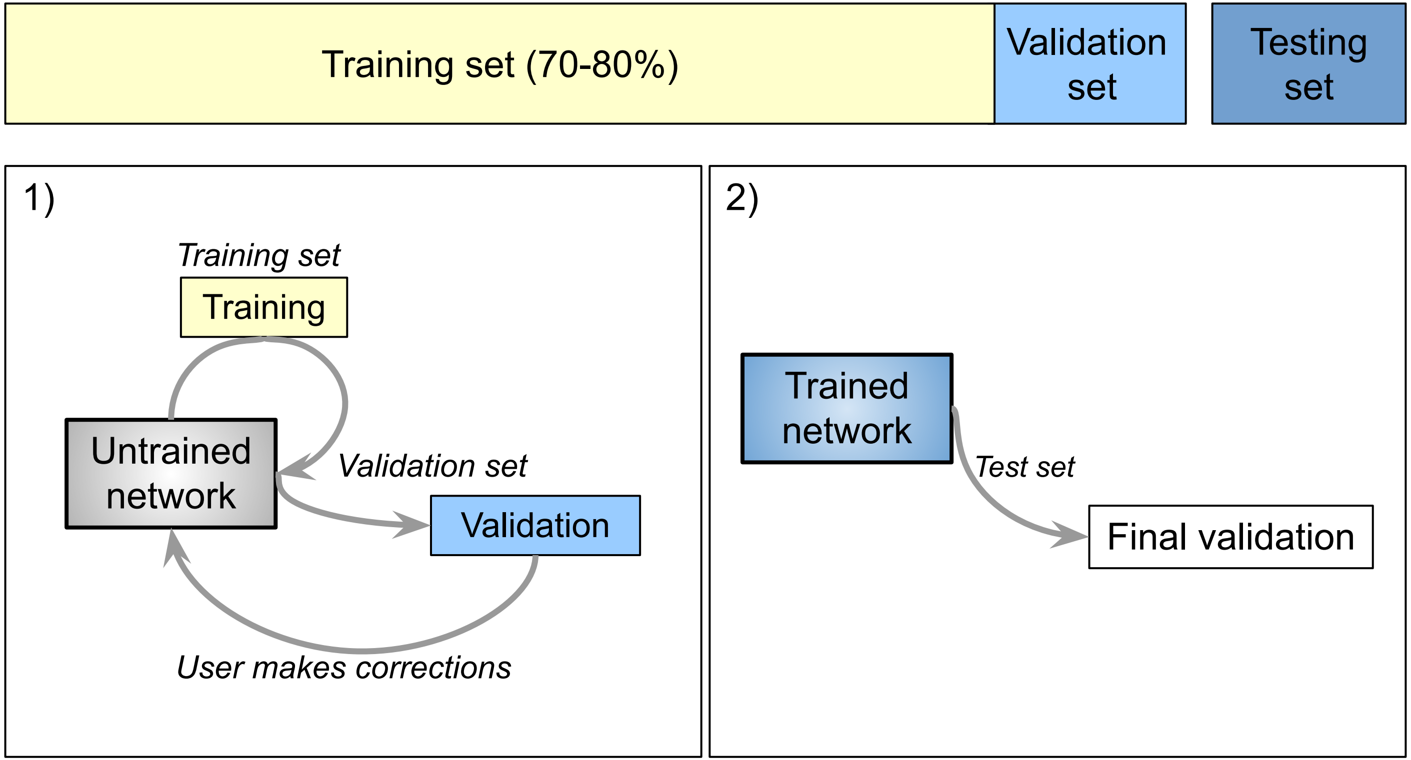
Regression : output is one or more real numbers



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



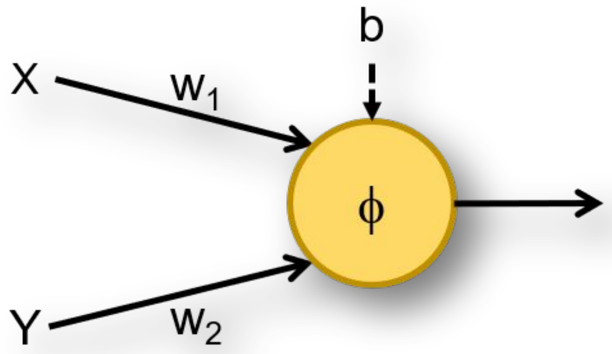
Training, testing and validation sets



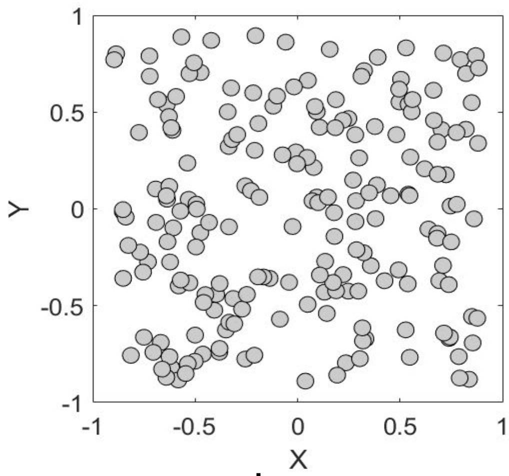
Train a single neuron classifier

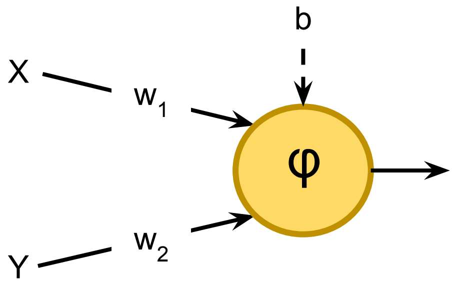
Example n°1 : Ex1_Clusterization_linearly_separated.ipynb

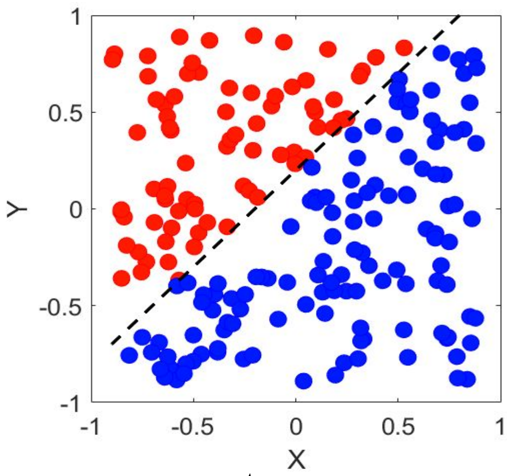
1. Understand the principle of the training
2. Train the classifier and test its accuracy
3. First step with Keras/TensorFlow



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 the classifier with Keras

1- Definition of the network architecture

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(1,activation='sigmoid', input_shape=(2,)))
```

2- Definition of the training options

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

3- Training

```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 50,
                    validation_data = (Validation_data, Validation_label))
```

Definition of the classifier with Keras

1- Definition of the network architecture

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(1,activation='sigmoid', input_shape=(2,)))
```

2- Definition of the training options

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

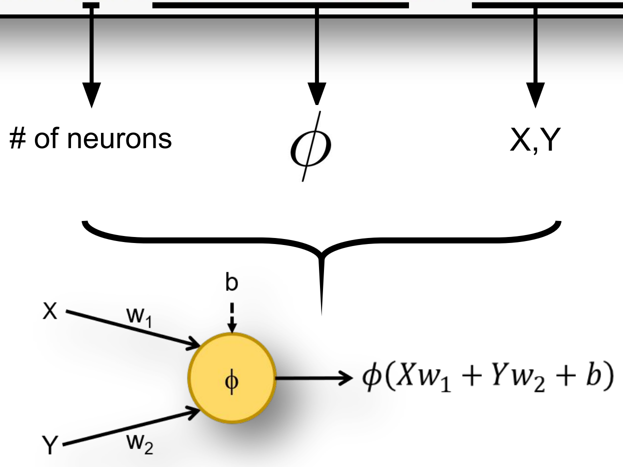
3- Training

```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 50,
                    validation_data = (Validation_data, Validation_label))
```

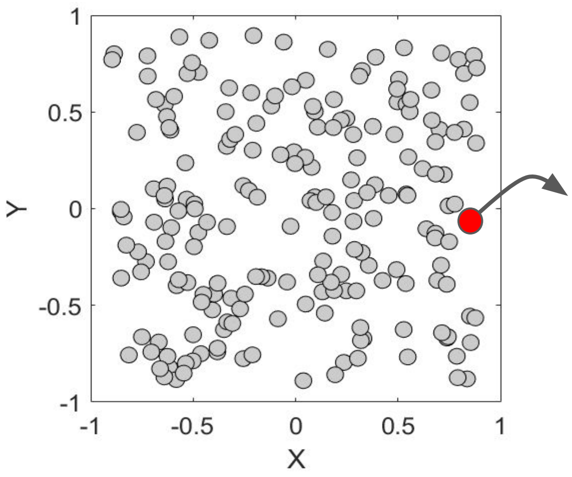
Definition of the classifier with Keras

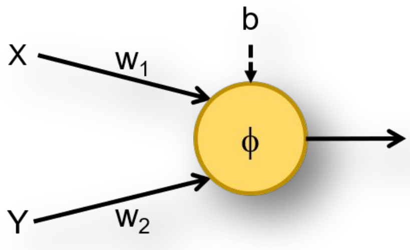
1- Definition of the network architecture

```python
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(1,activation='sigmoid', input_shape=(2,)))
```



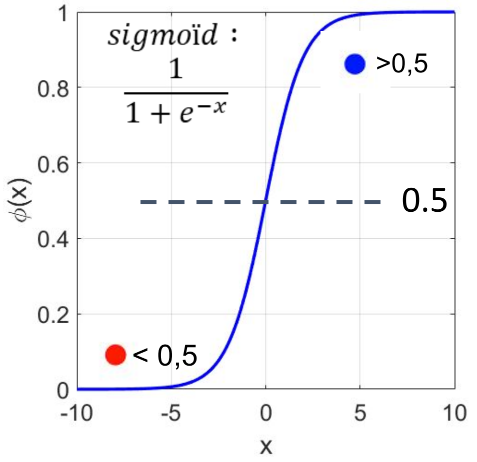
Training





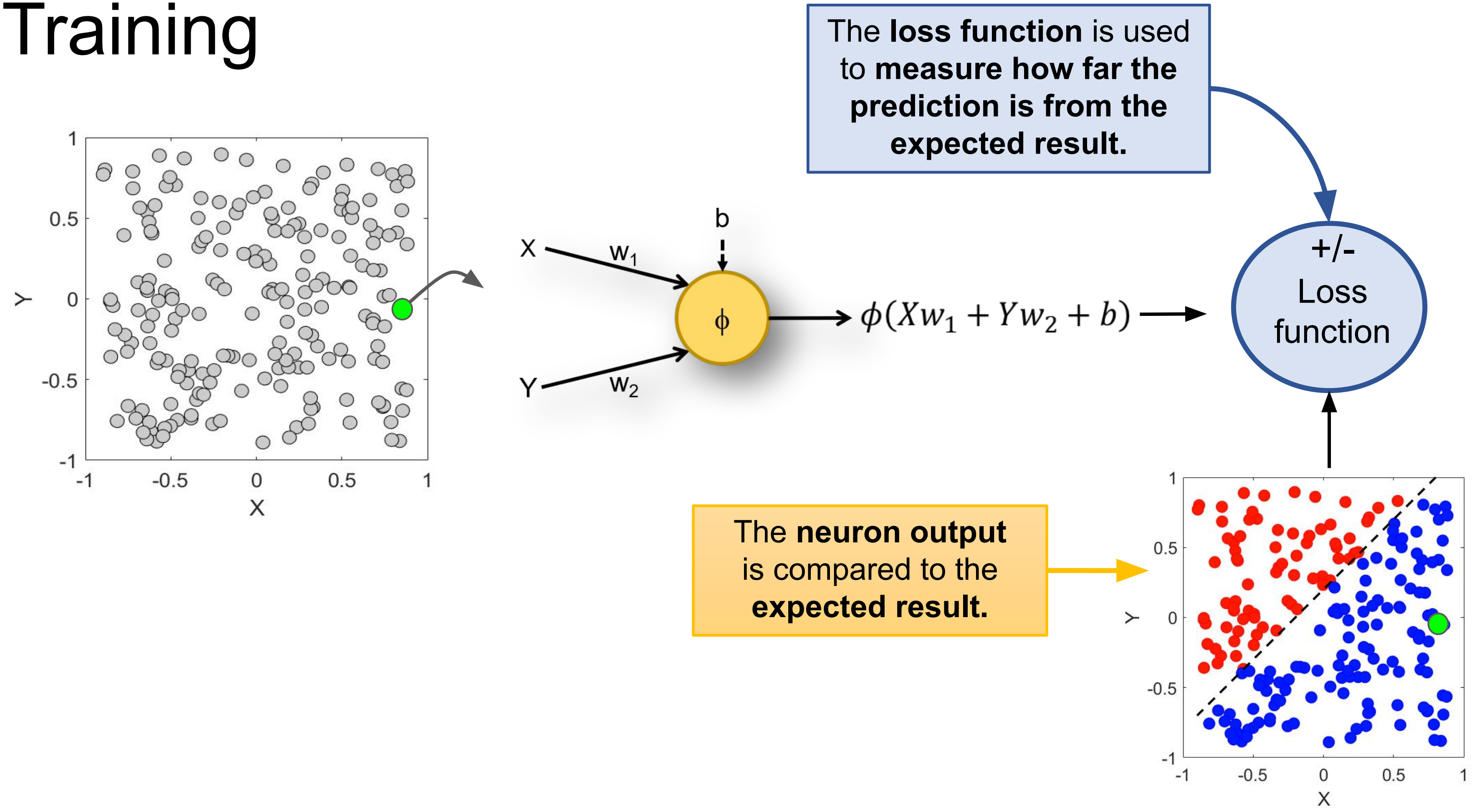
$$\phi(Xw_1 + Yw_2 + b)$$

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



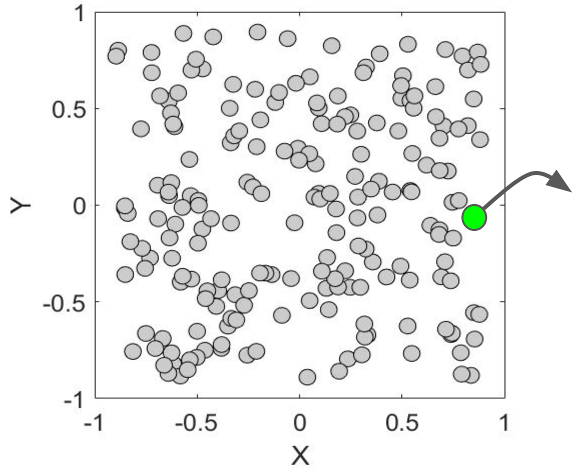
Initially the weights are *randomly* initialized and the bias set to zero.

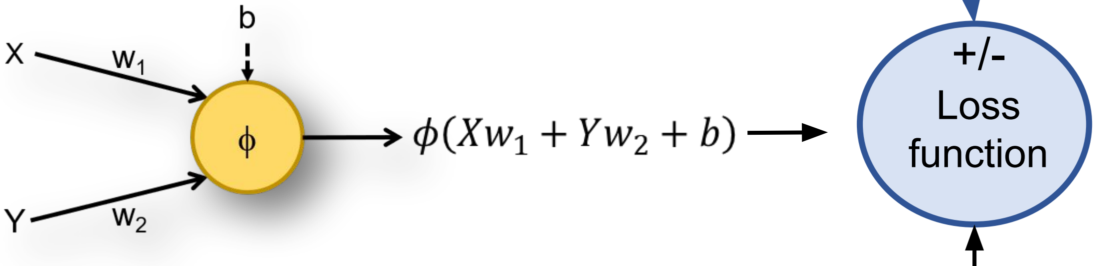
Training



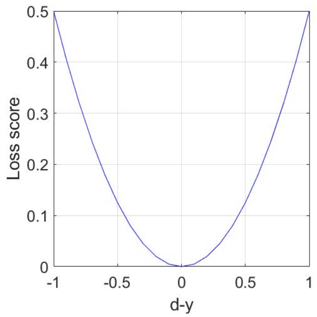
The **loss function** is used to **measure how far the prediction is from the expected result.**
1
0.5
0
-0.5
-1
Y
-1
-0.5
0
0.5
1
X
X
Y
w_1
w_2
b
ϕ
$\phi(Xw_1 + Yw_2 + b)$
+/-
Loss function
The **neuron output** is compared to the **expected result.**

Training





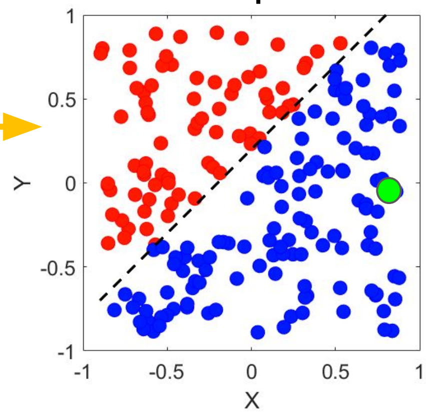
The **loss function** is used to **measure how far the prediction is from the expected result.**



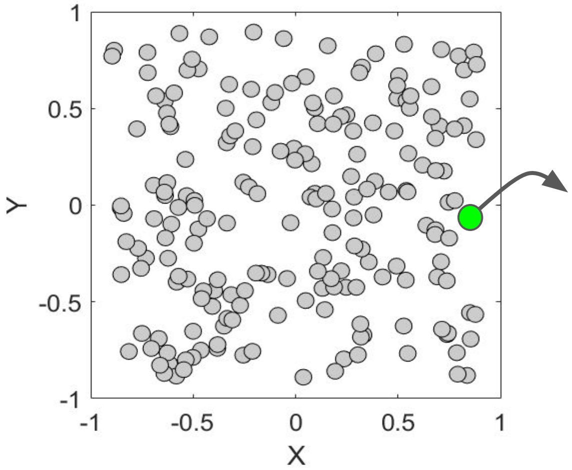
$$J = \frac{1}{2}(d_i - y_i)^2$$

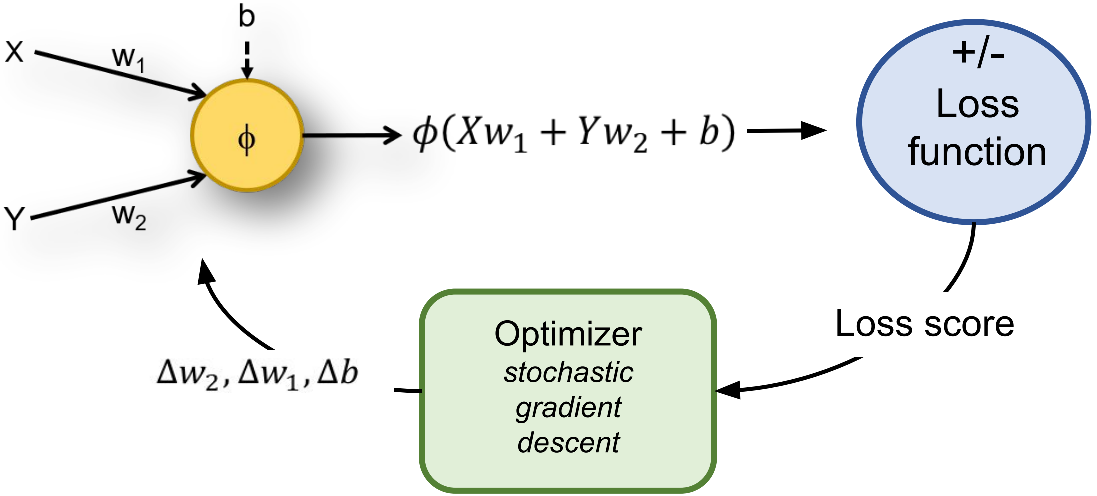
Squared error function, mainly used for regression problems.
d_i = prediction
y_i = true label

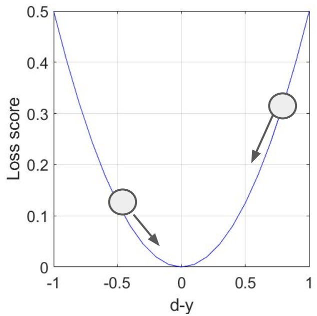
The **neuron output** is compared to the **expected result.**



Training







$$J = \frac{1}{2}(d_i - y_i)^2$$

Squared error function, mainly used for regression problems.

Definition of the classifier with Keras

1- Definition of the network architecture

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(1,activation='sigmoid', input_shape=(2,)))
```

2- Definition of the training options

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

3- Training

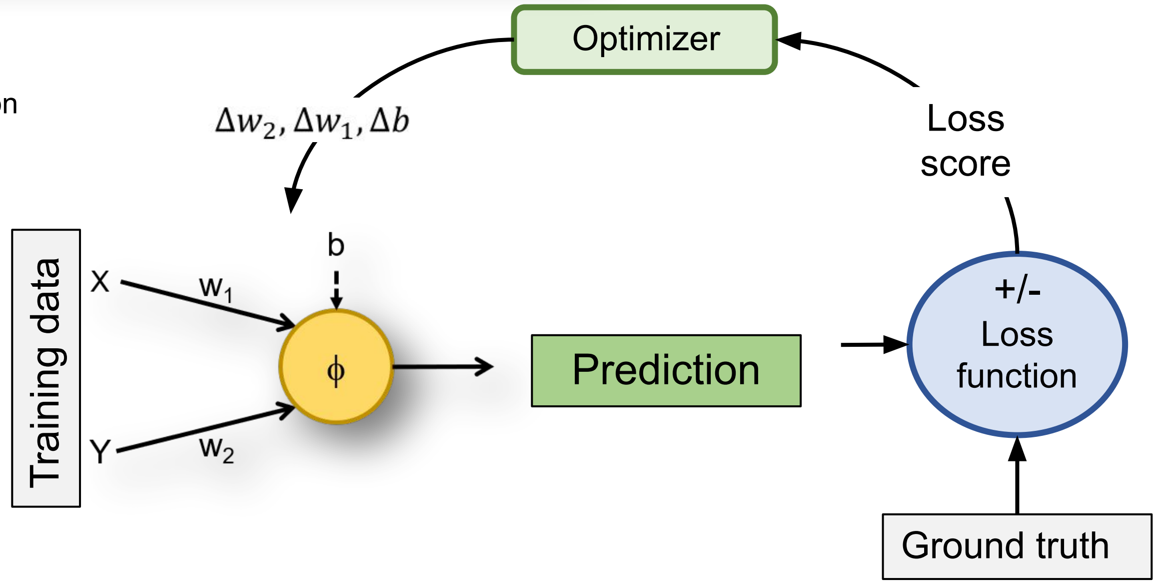
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 50,
                    validation_data = (Validation_data, Validation_label))
```

Model compiling

2- Definition of the training options

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

'sgd' : stochastic gradient descent
'binary_crossentropy' : loss funct. for classification
'accuracy' : to add to log



Definition of the classifier with Keras

1- Definition of the network architecture

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(1,activation='sigmoid', input_shape=(2,)))
```

2- Definition of the training options

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

3- Training

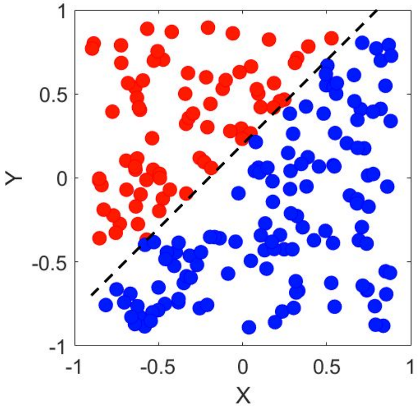
```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 50,
                    validation_data = (Validation_data, Validation_label))
```

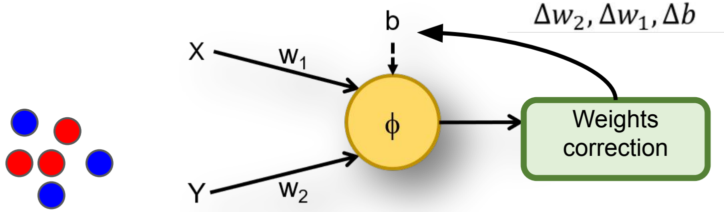
Start the training

3- Training

```
history = model.fit(Training_data,
                    Training_label,      batch_size = 4,
                    epochs = 50,
                    validation_data = (Validation_data, Validation_label))
```

	Training	**validation**
data	(x1,y1),(x2,y2),…,(xn,yn)	(xn+1,yn+1),…,(xm,ym)
label	(red) , (red) ,…, (blue)	(red) ,…,(blue)





one *batch* at time

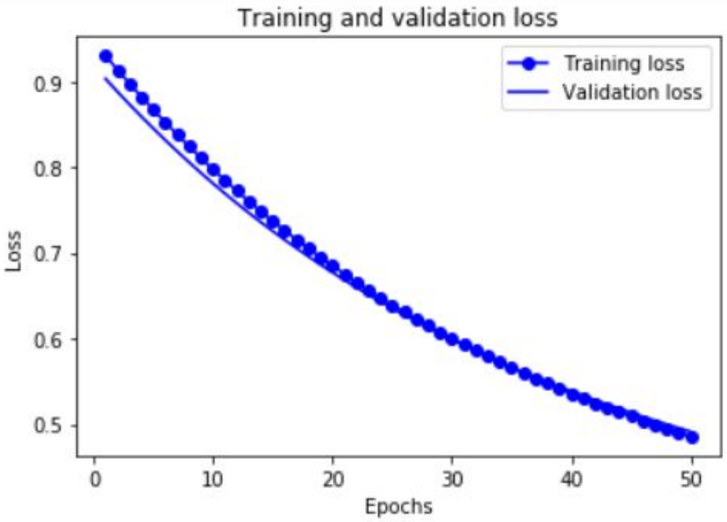
1 *epoch* : all points used once

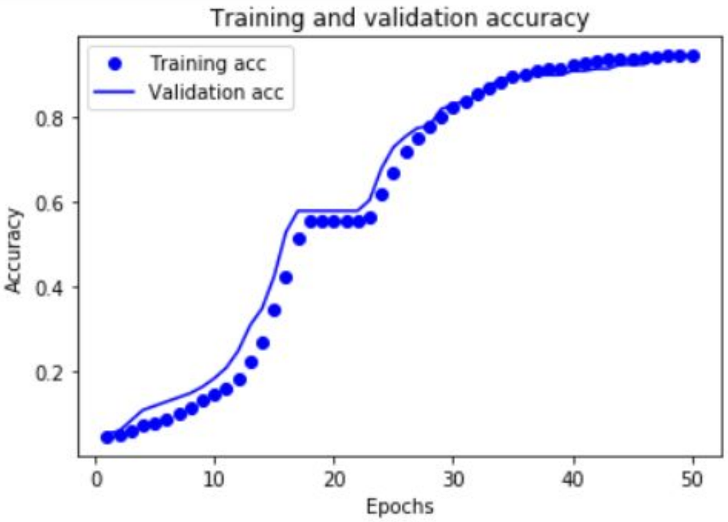
repeat *n epochs*

Training results

3- Training

```
history = model.fit(Training_data,
                    Training_label,
                    epochs = 50,
                    validation_data = (Validation_data, Validation_label))
```

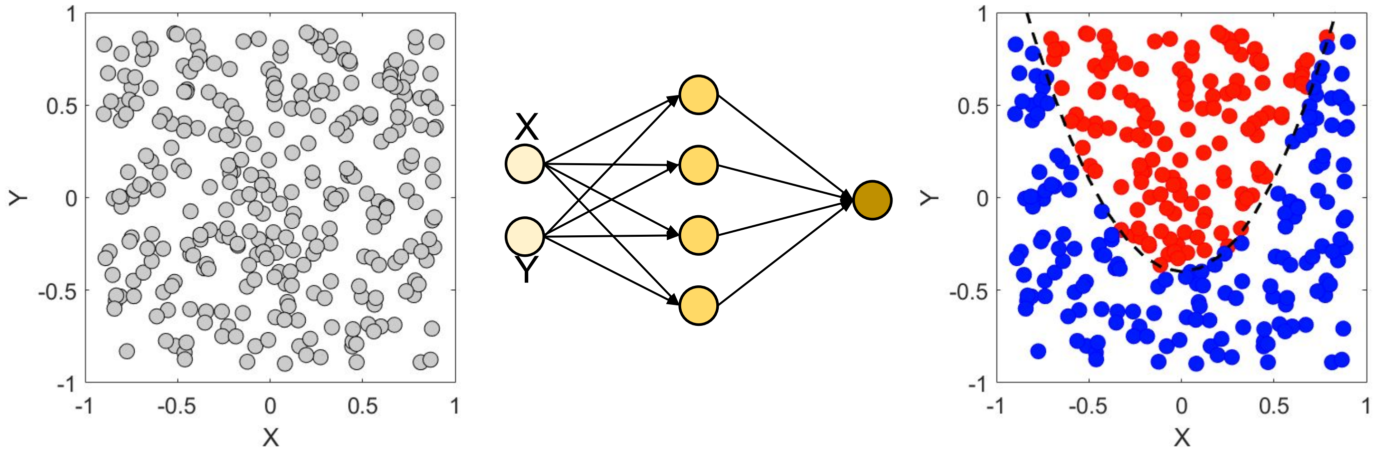




Example 2 : classify non-linearly separable data

<u>Example n°2</u> : 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

n.of neurons

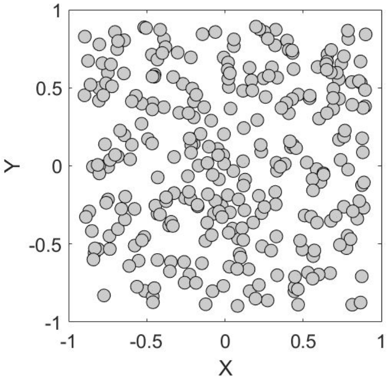
try sigmoid or relu

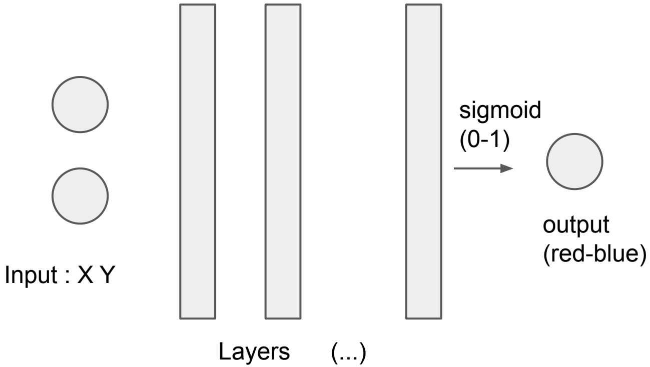
1st layer needs input specified

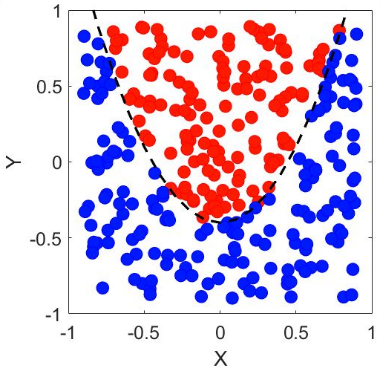
1st layer

2nd layer etc...

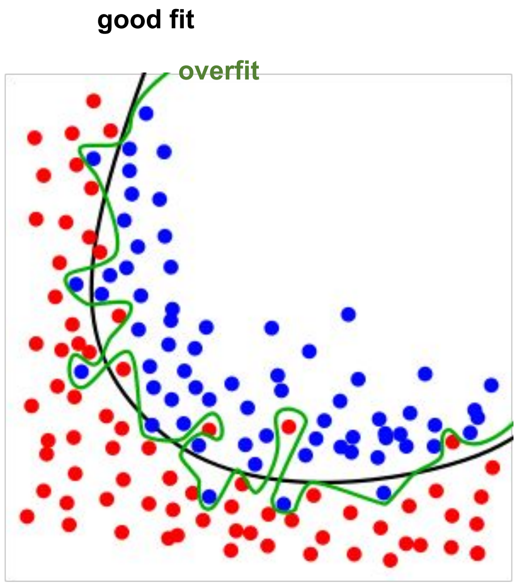
```
model = models.Sequential()
model.add(layers.Dense(4,activation='sigmoid', input_shape=(2,)))
model.add(layers.Dense(1,activation='sigmoid'))

model.compile(optimizer = optimizers.sgd(lr=5e-2),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```





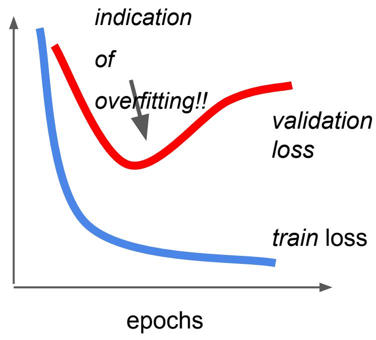


Overfitting



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 input data



in our example
try with small Ntrain
and a large model