Machine Learning

2022

Philippe Carré

philippe.carre@univ-poitiers.fr

institut de recherche

Université de Poitiers

FACULTÉ DES SCIENCES FONDAMENTALES ET APPLIQUÉES UNIVERSITÉ DE POITIERS

Practical matters

• Grading: Final grades are determined as follows

- 33% realization

- 33% report,

- 33% Presentation,

• Homework/classroom project :

– each project is realized by one student (Python, Framework Keras/TF)

– Each student have to send a report: explanation of the method, analysis of the results, proposition of modification

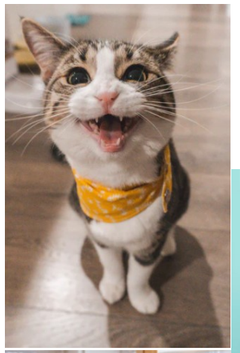
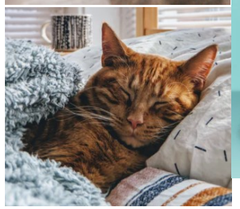
I Introduction

Prise de décision
à partir de la mesure

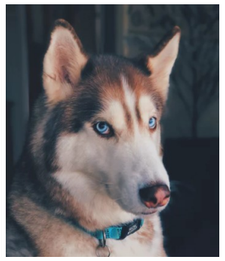
Le monde du numérique génère un grand volume de données, qui pourront être utilisées pour réaliser des gains en matière de production et de services mais pour améliorer les conditions de vie grâce à des solutions innovantes

L'analyse des données permet d'extraire des informations utiles des flux d'informations numériques. Machines ou objets connectés deviennent « intelligents ».

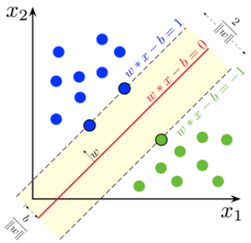
Supervised Learning


« Cat »

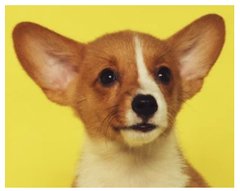
« Dog »



Annotated Data → Machine Learning

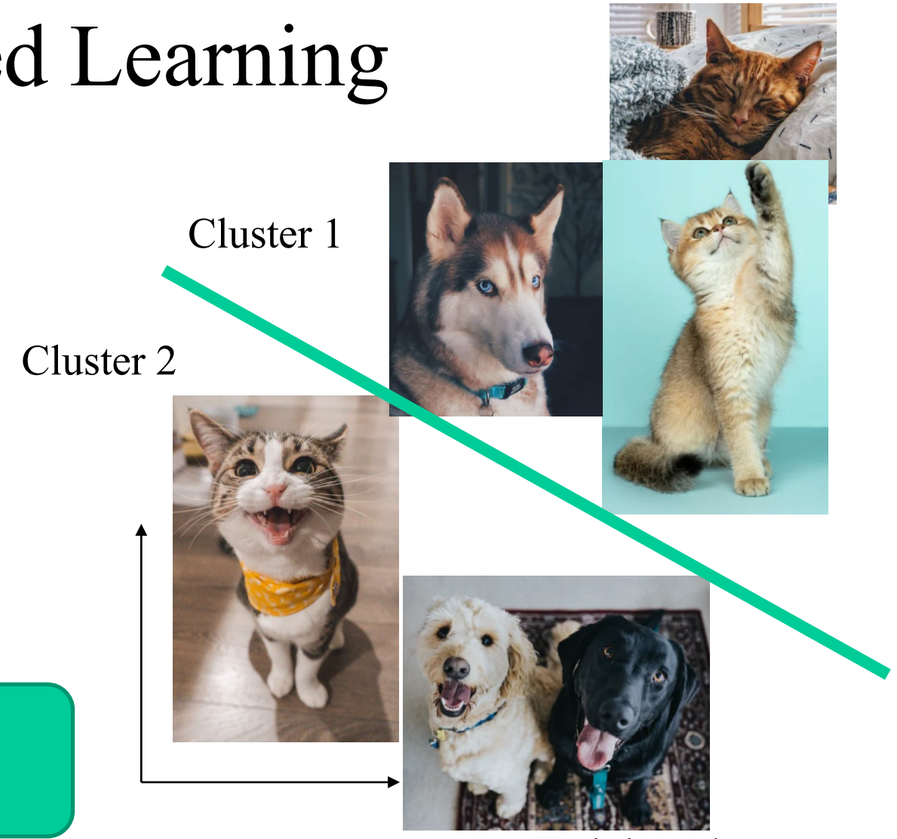
Training phase

New Data → Learned model → Prediction on Data



« Dog » (65%)
« Cat » (35%)

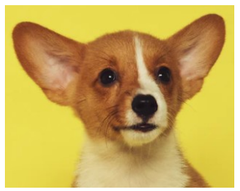
Unsupervised Learning



Unlabeled Data → Clustering algorithm

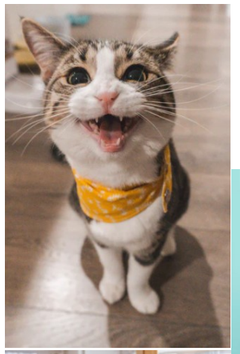
Training phase

New Data → Learned model → Prediction on Data



« cluster 1 » (100%)
« cluster 2 » (0%)

Reinforcement Learning



« Cat »

« Dog »

« Dog » (100%)
« Cat » (0%)

Annotated Data

Iterative learning algorithm

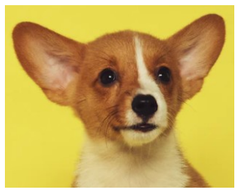
Feedback

Training phase

New Data

Learned model

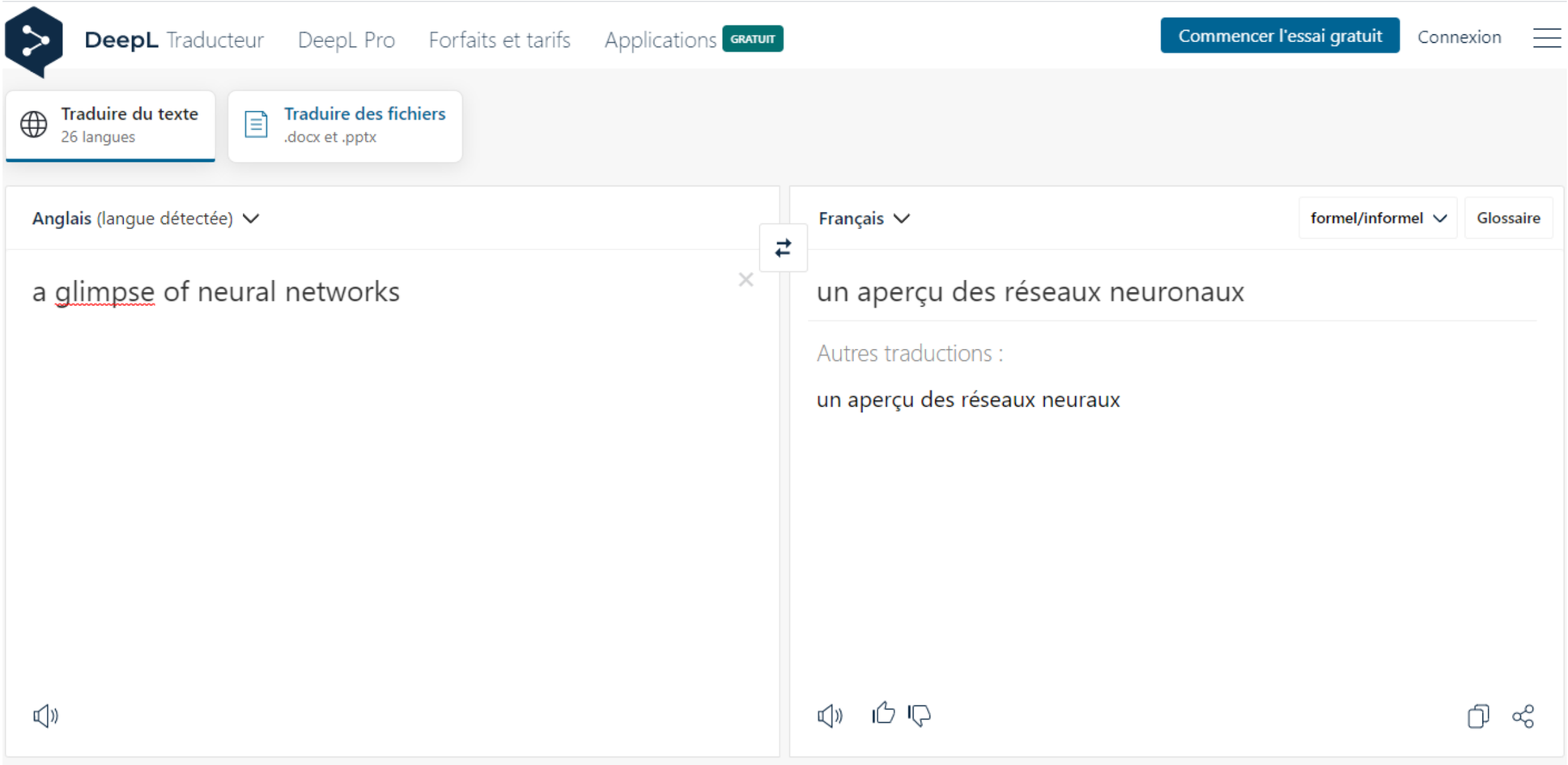
Prediction on Data



« Dog » (65%)
« Cat » (35%)

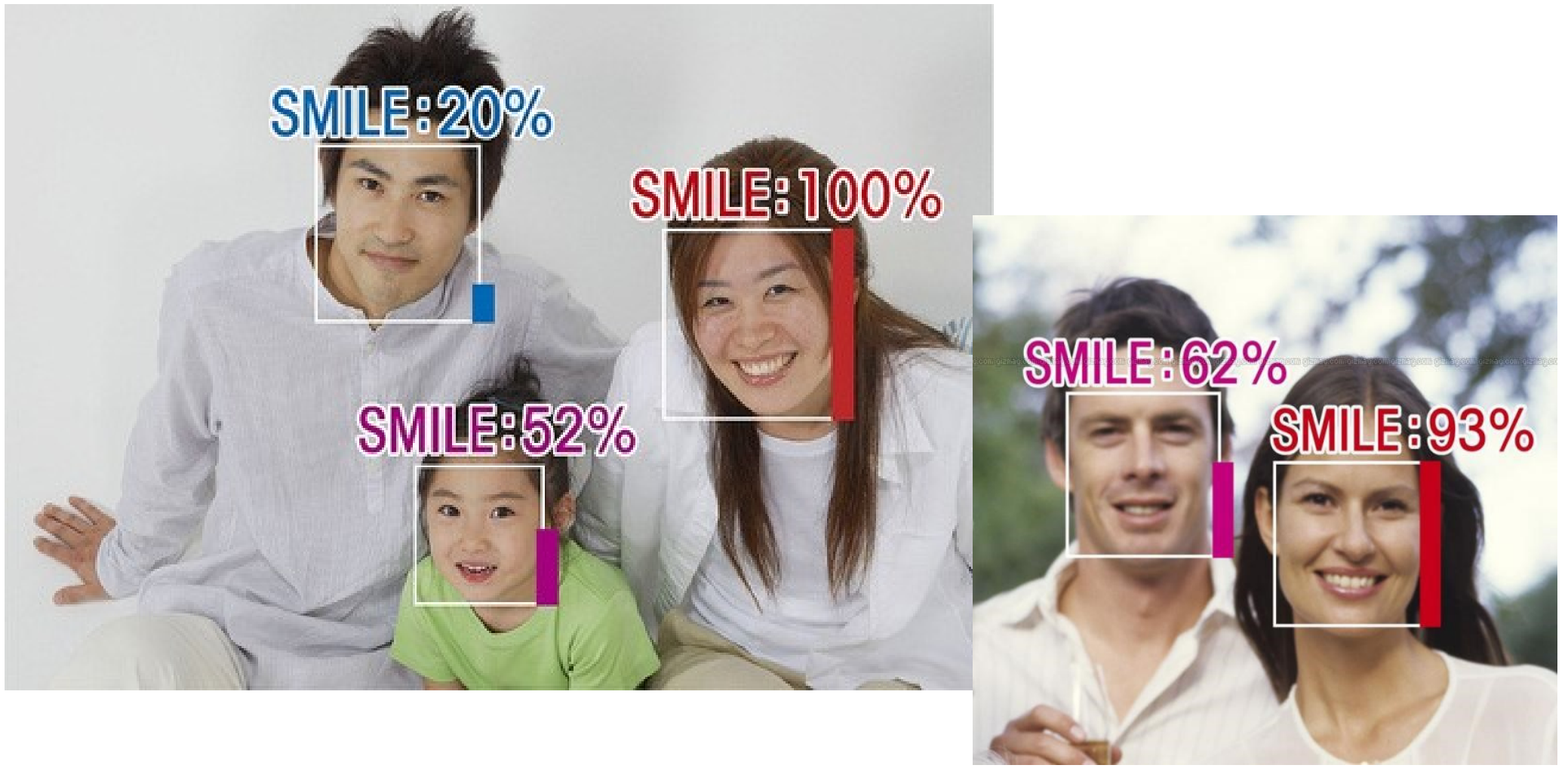
Deep learning : applications

Natural Language Processing : DeepL



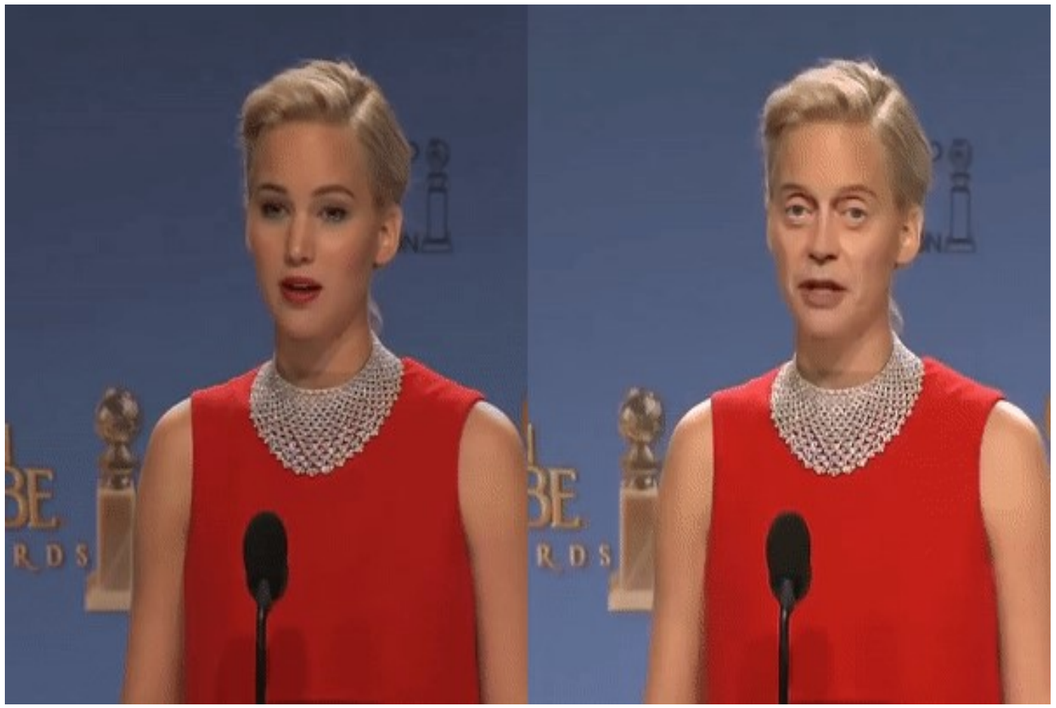
Deep learning : applications

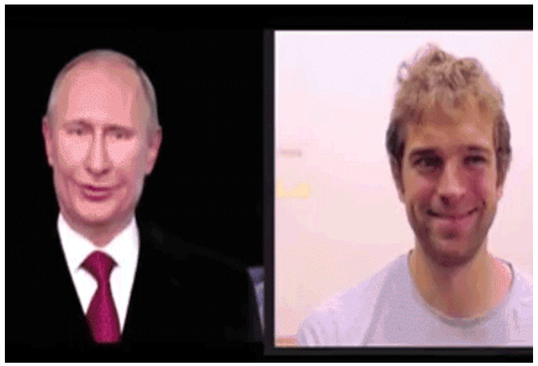
Face and emotion detection



Deep learning : applications

Deep fake



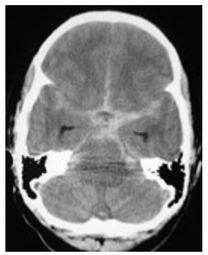


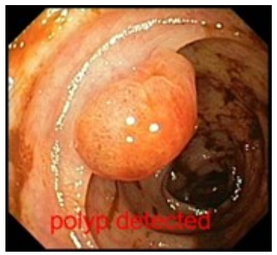
GAFA : **Google, Apple, Facebook, Amazon**

Applying machine learning science to Facebook products

Machine learning is essential to Facebook. It helps people discover new content and connect with the stories they care the most about. Our applied machine learning researchers and engineers develop machine learning algorithms that rank feeds, ads and search results, and create new text understanding algorithms that keep spam and misleading content at bay. New computer vision algorithms can “read” images and videos to the blind and display over 2 billion translated stories every day, speech recognition systems automatically caption the videos that play in your news feed, and we create new magical visual experiences such as turning panorama photos into fully interactive 360 photos.

Medical application





For example, how can we program a system in such a way that anomalous frames in an endoscopic video stream are detected automatically? In the machine learning paradigm, the solution is found by providing the system with a large number of examples from which, in a training stage, the system tries to generalize. Once trained, the system is then ready to be used on any individual image in the operational stage.

- Detection of object **categories**
  - is there a … in this picture
- More generally: relevance of labels (action, place, ...)

person

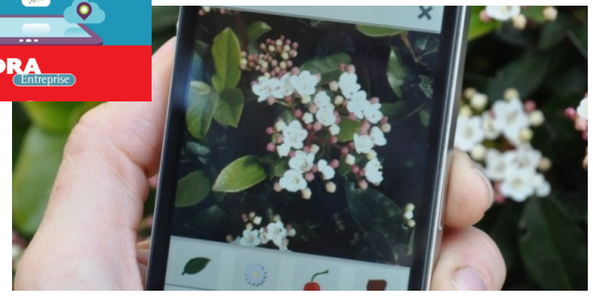
glass

drinking

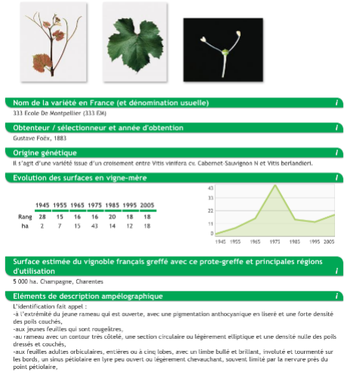
indoors

Example: Industrial experiences

Measures → Non-Destructive Testing

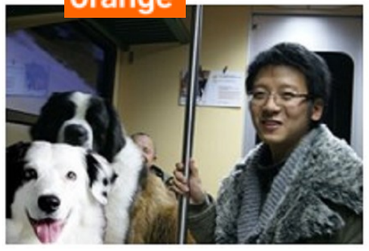


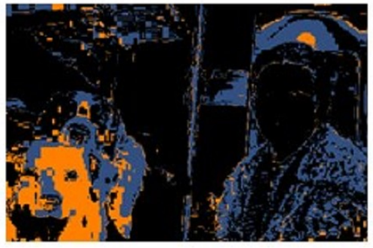
Notice Fiche Porte-Greffes



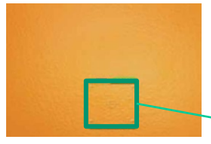
orange™

Security: Detection of modification

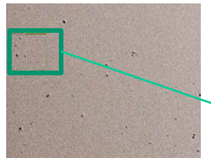




Industrial painting analysis

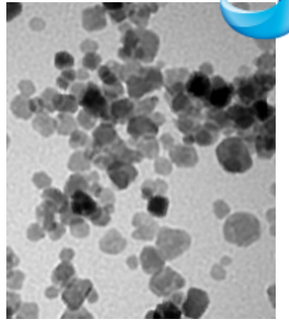


Surface Analysis

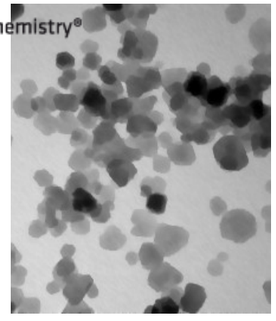


Default detection

SOLVAY
asking more from chemistry®



Pre-processing



Caracterisation
Classification

IA = intelligence ?

IA ? Le développement de l'intelligence artificielle : les algorithmes permettent de synthétiser et d'analyser la masse d'informations glanées par les objets connectés ou différents capteurs.

Les tâches relevant de l'IA sont parfois très simples pour les humains (par exemple reconnaître et localiser les objets dans une image…).

Elles requièrent parfois de la planification complexe.

En IA on associe presque toujours l'intelligence aux capacités d'apprentissage.

- C'est grâce à l'apprentissage qu'un système intelligent capable d'exécuter une tâche peut améliorer ses performances avec l'expérience.
- C'est grâce à l'apprentissage qu'il pourra apprendre à exécuter de nouvelles tâches et acquérir de nouvelles compétences.

Des neurones pour la modélisation et la décision

philippe.carre@univ-poitiers.fr

institut de recherche

Université de Poitiers

FACULTÉ DES SCIENCES FONDAMENTALES ET APPLIQUÉES UNIVERSITÉ DE POITIERS

Neuronal network approach

Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy.

The most popular approaches to machine learning are **artificial neural networks**.

Difficulties: within-class variations

Discrimination Process

Learning and Decision

Sensor

Modelisation

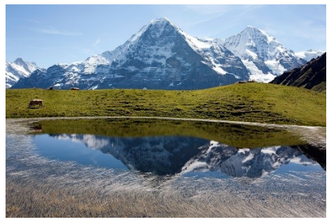
AI

Real Word

Data

Features

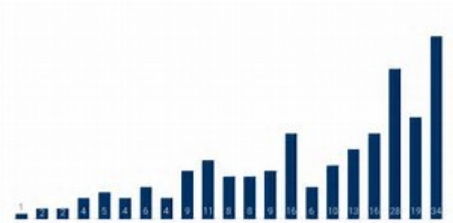
Decision



$$S = \{s_1, \ldots, s_k, \ldots, s_K\}$$

Signal

$$X = R^d$$



$$A = \{a_1, \ldots, a_l, \ldots, a_L\}$$

The input of the process

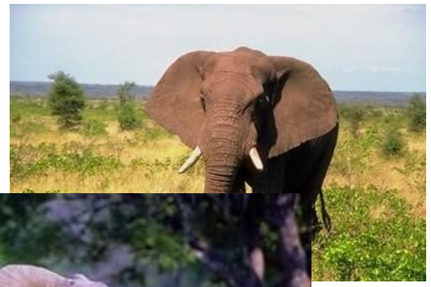
	individu 1	…	individu i	…	individu n
var 1	$x(1,1)$		$x(1,i)$		$x(1,n)$
…					
var j	$x(j,1)$		$x(j,i)$		$x(j,n)$
…					
var p	$x(p,1)$		$x(p,i)$		$x(p,n)$

Data tabular

$X = (x(j,i))_{1 \leq i \leq n,\ 1 \leq j \leq p}$ N examples and P measures/features

⟶ Space of measures is inhomogeneous

Learning



Training set

Learning algorithm

$\mathbf{x} \longrightarrow \Lambda(x) \longrightarrow$ predicted y

$x \in R^p$ $\omega_1, \ldots, \omega_K$

Our goal is, given a training set, to learn a function so that Λ(x) is a “good” predictor for the corresponding value of y

When the target variable can take on only a small number of discrete values we call it a classification problem.

Neural Networks

Discrimination algorithm

Measures vector x ⇨ **C : Algorithm Machine Learning** ⇨ Algorithm answer « $y=C(x)$ »

$x \in R^P$ measures space

$y \in \{1,2,\ldots,L\}$ Decision set

Machine learning $C: R^d \rightarrow \{1,\ldots,l,\ldots,L\}$

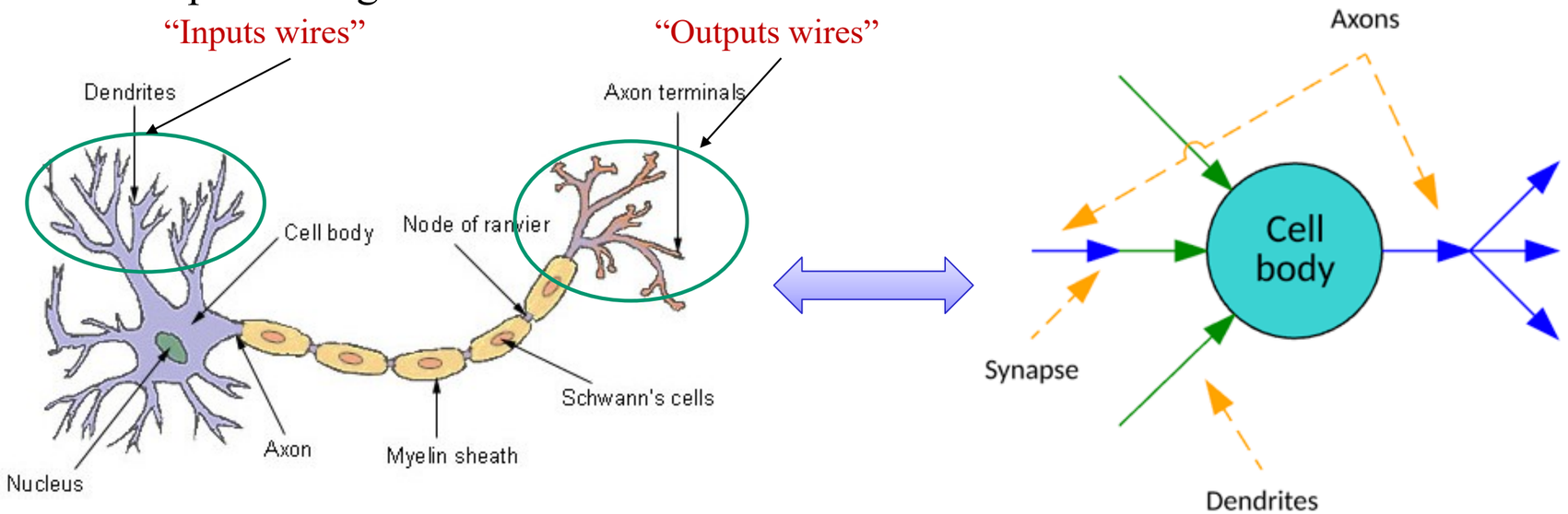
$x \mapsto C(x)$

The aim:

$\forall x \in R^d$, $C(x) =$ "the true decision"

Basics: Perceptron (Rosenblatt 1954)

- Inspired by neurobiology:
  - 85 billions neurons : 10^{15} synapses, 180 000 km of connections
  - Neuron receives signals through dendrites
  - Inputs are then summed up together inside the cell body
  - Neuron fires the if the output exceed a threshold
  - Neuron sends signal to other neurons through axon
- Artificial neuron: rough simulation of the neuron of humain brain
  - same basic components
  - same information processing



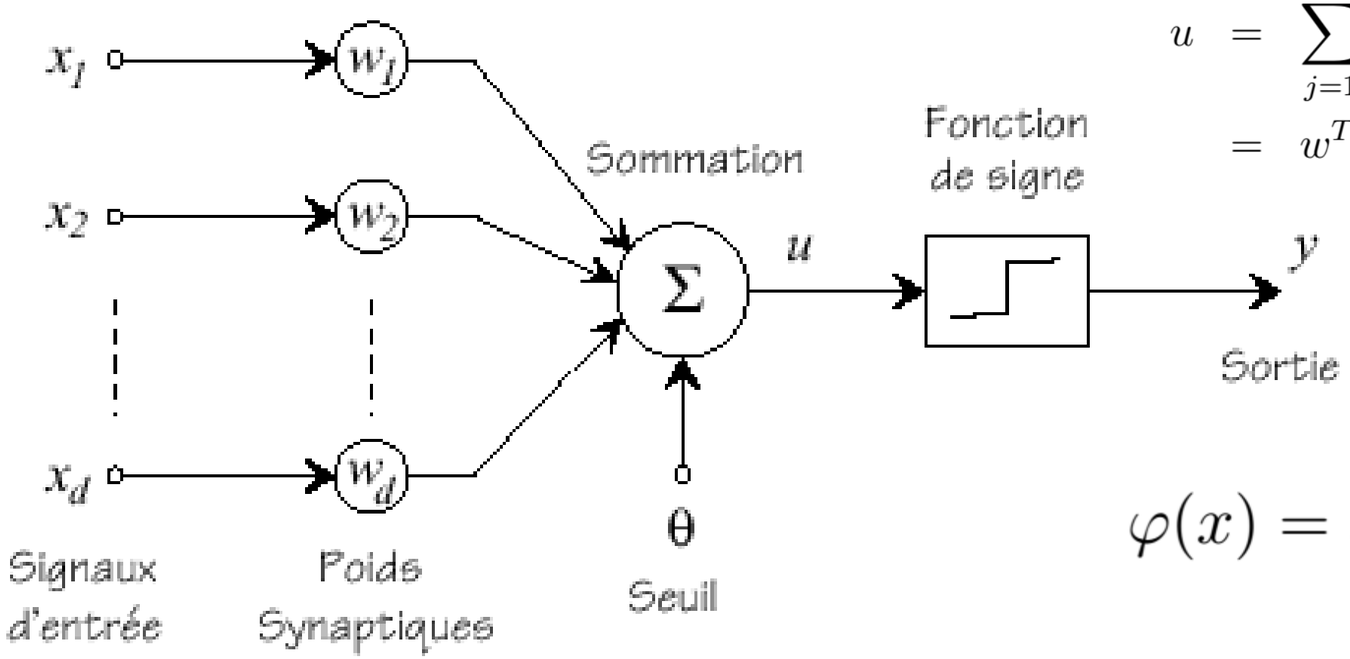
Biological neuron **Artificial neuron**

Perceptron: principle

The *Perceptron :* the simplest form of a neural network is able to classify data into two classes (*lineary separable)*.

Basically it consists of a single *neurone* with a number of adjustable weights.

It has three basic elements:
1. A set of connecting synapses; each link carries a weight w.
2. A summation sums the input signals after they are multiplied by their respective weights.
3. An activation function: the sign function



$$u = \sum_{j=1}^{d} w_j x_j + \theta = \sum_{j=0}^{d} w_j x_j \text{ avec } w_0 = \theta \text{ et } x_0 = 1$$
$$= w^T x$$

$$\varphi(x) = sign(x)$$

Perceptron: principle

We can use Perceptron neurons to implement the basic logic gates (e.g. AND, OR, NOT)

All we need to do is find the appropriate connection weights and neuron thresholds to produce the right outputs for each set of inputs.

We shall see explicitly how one can construct simple networks that perform NOT, AND, or OR.

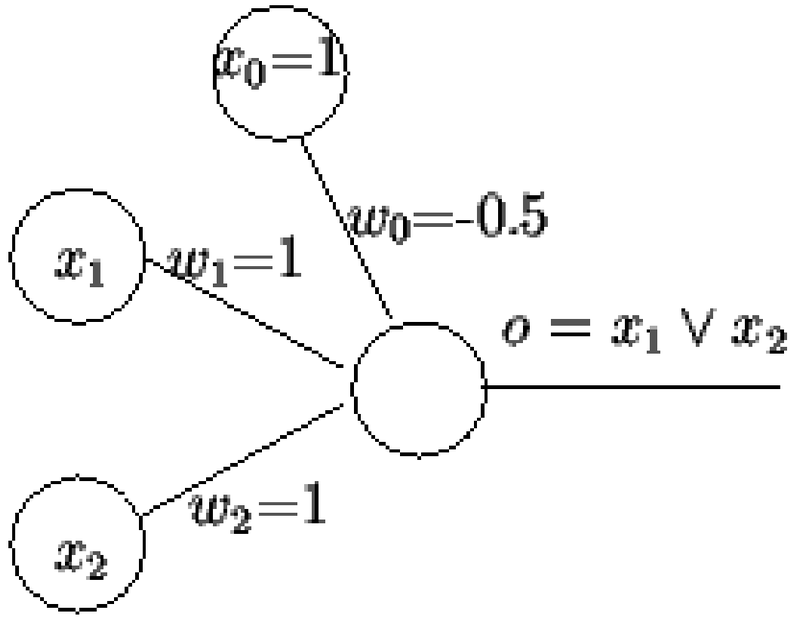
A binary classification

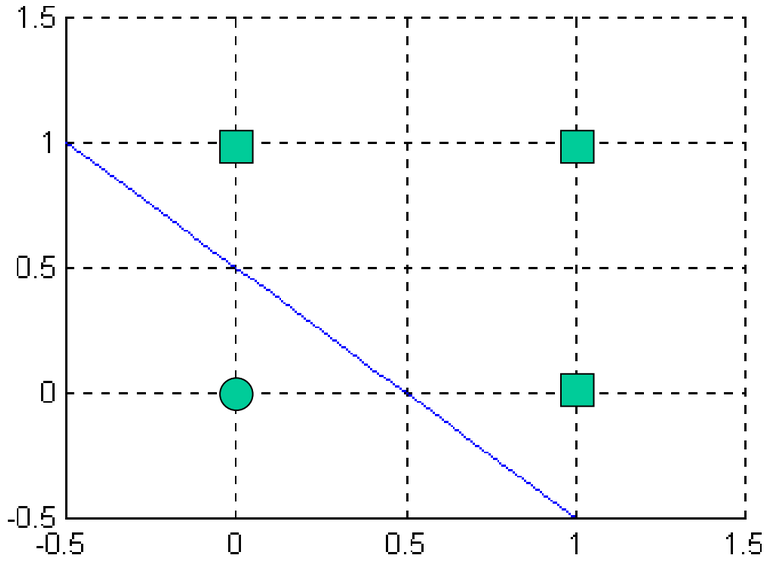
OR		
in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	1

Classification problem

$$\omega_0 : \{0\} \text{ et } \omega_1 : \{1\}.$$

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ x \to \omega_0 & x \to \omega_1 & x \to \omega_1 & x \to \omega_1 \end{array}$$





$\mathbf{w}^T\mathbf{u} \geq 0$ for every input vector $\mathbf{u}$ belonging to class C_1

$\mathbf{w}^T\mathbf{u} < 0$ for every input vector $\mathbf{u}$ belonging to class C_2

$$x \rightarrow \omega_0 \text{ ssi } \underbrace{w^T x} \geq 0\,;\; x \rightarrow \omega_1 \text{ otherwise}$$

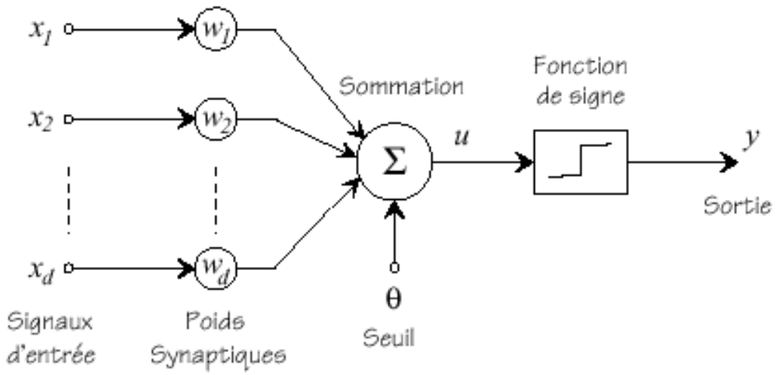
Linear decision rule

Assuming, to be general, that the perceptron has p inputs, then the equation

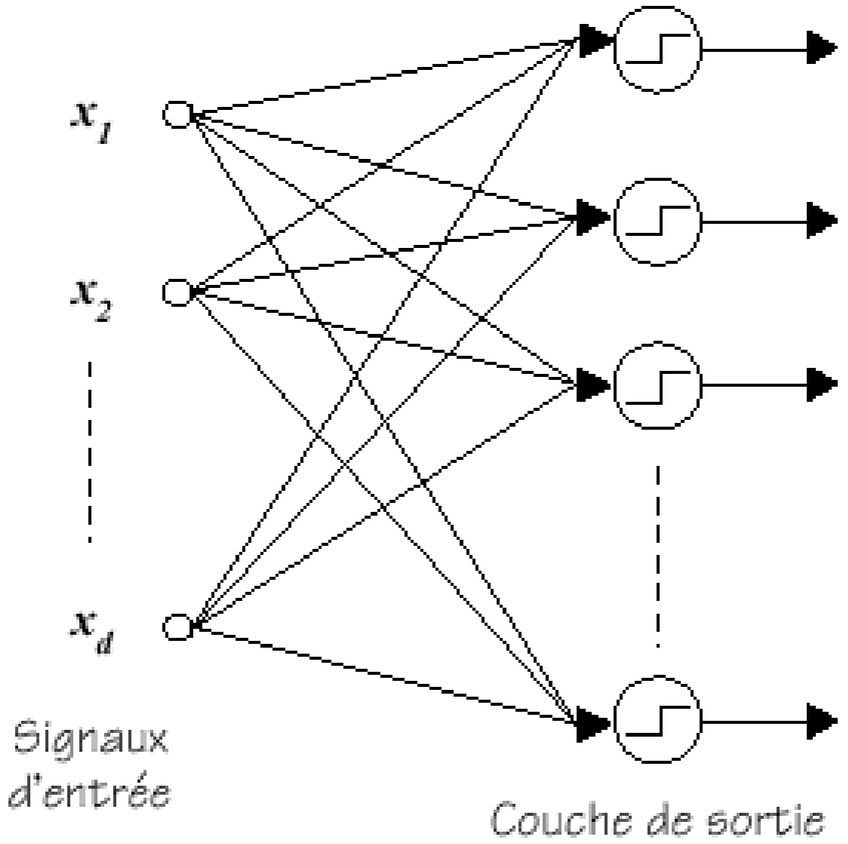
$$w_1 x_1 + \ldots + w_d x_d + w_0 = 0$$

in an p dimensional space with coordinates x1,x2…xd, defines a hyperplane as the switching surface between the two different classes of input.

Perceptron for a number of cluster >2



A binary classification

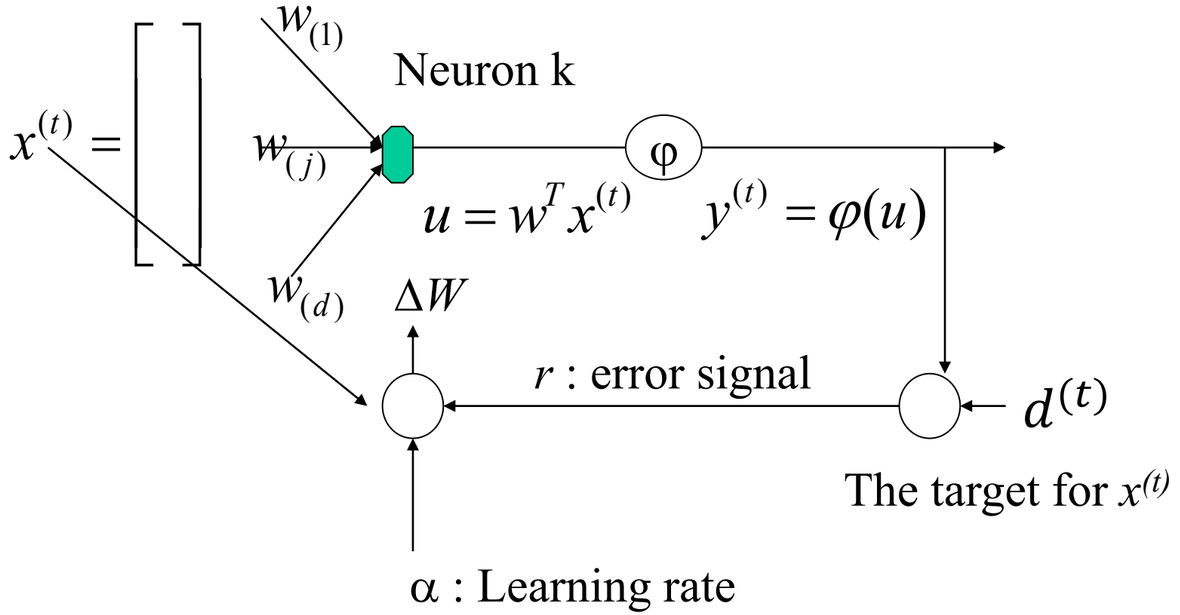


y_1 Cluster 1: yes or not

y_2 Cluster 2: yes or not

y_C Cluster C: yes or not

Training the neural Network



Generate a training pair or pattern:
- an input ***x*** = *[x_1 x_2 … x_n]*
- a target output *d* (known/given)

Initialize weights at random

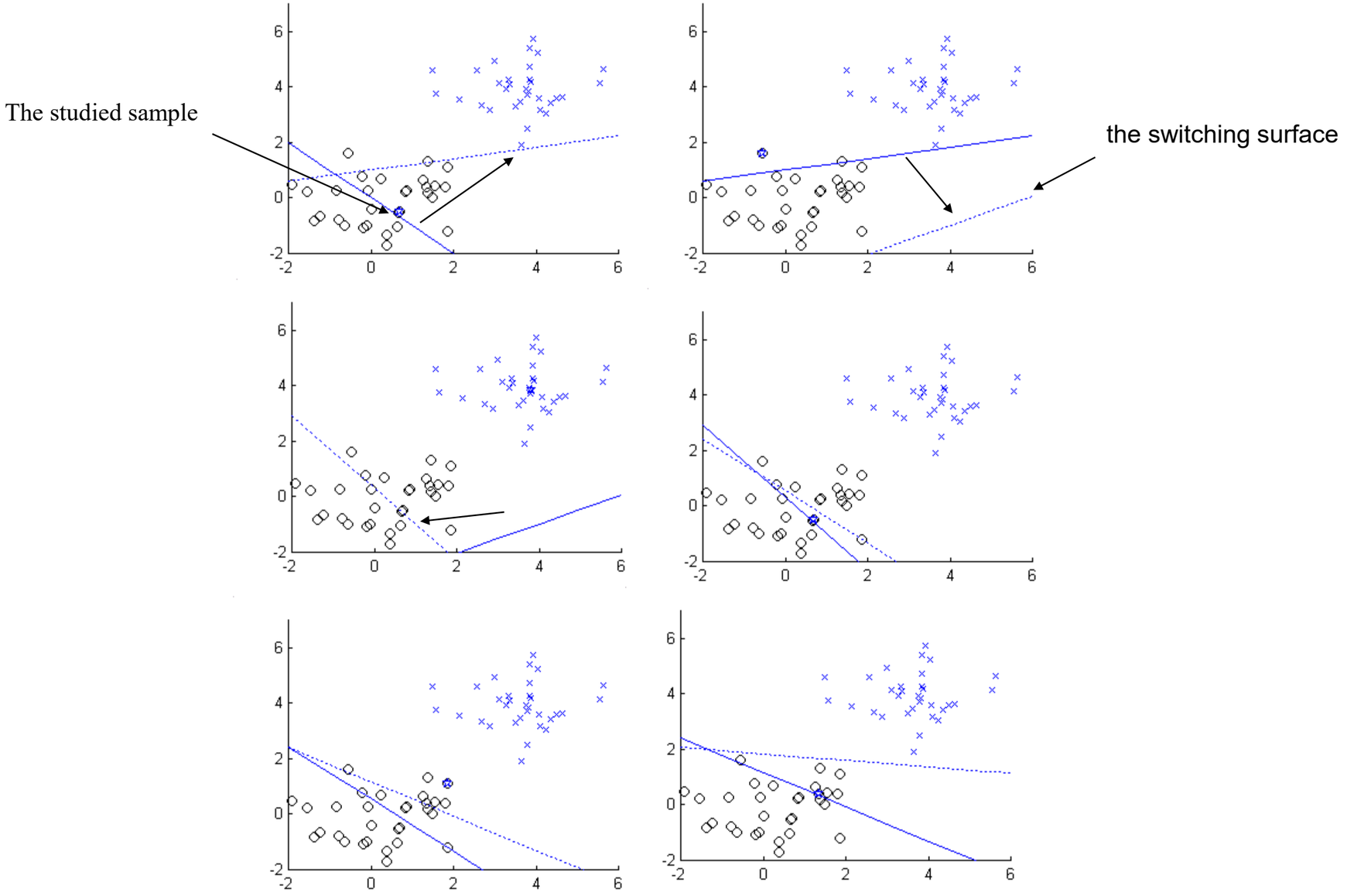
For each training pair/pattern ***(x, d)***

- Compute output *y*
- Compute error, *r=f(d – y)*
- Use the error to update weights as follows:

$$r = d - y$$

$$w^{n+1} = w^n \text{ +/- } \alpha r x$$

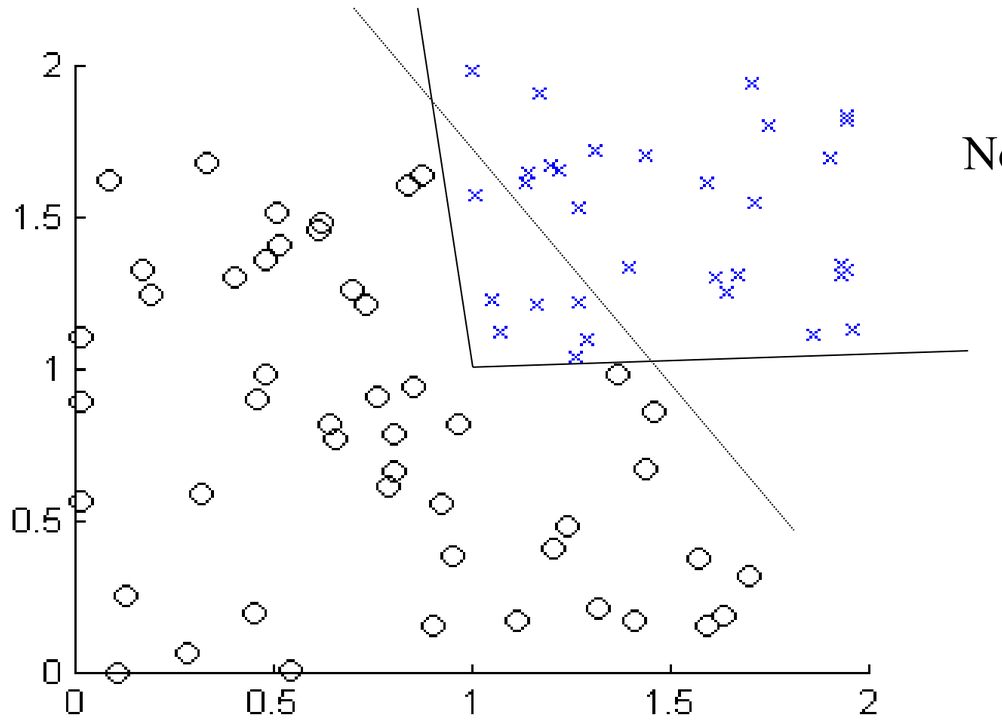
Repeat until “convergence”



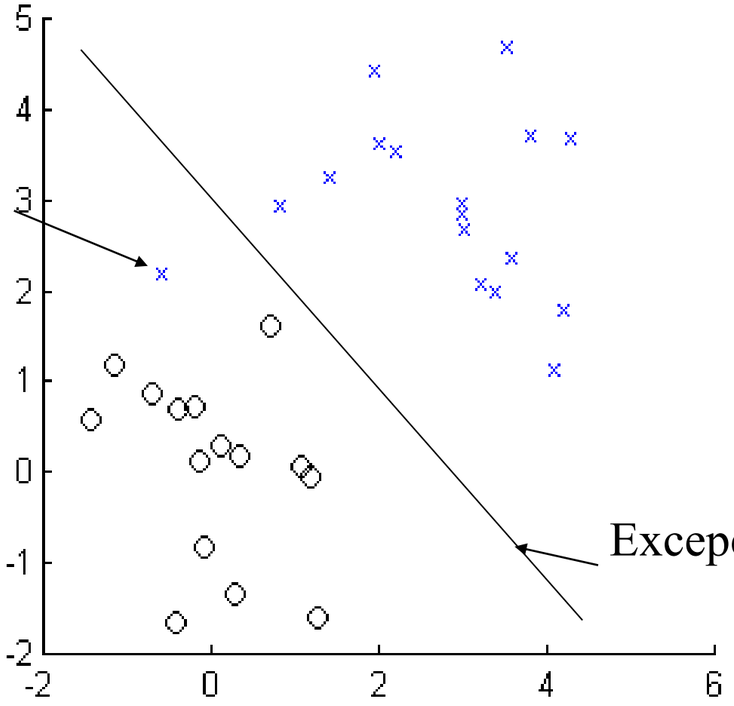
Training the Neural Network

Training the Neural Network: limit of the “simple” algorithm

Neuron defines two regions in input space where it outputs -1 and 1.
The regions are separated by a hyperplane **wx** = 0



Noise



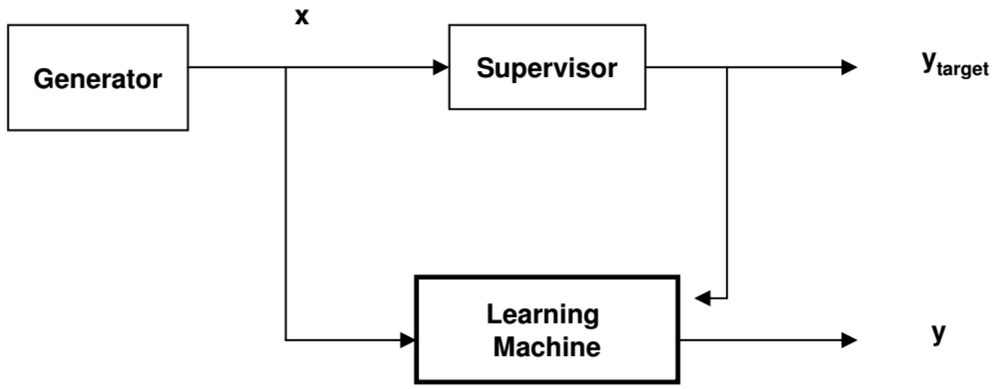
Excepected solution

Repeat 2 until convergence (i.e. error r is zero for each training pair)

Not possible

It is not possible to stop the training algorithm

Training the Neural Network: Widrow-Hoff algorithm



Training: Learn from training pairs (**x**, $\mathbf{y}_{target}$)

Testing: Given **x**, output a value y close to the supervisor's output y_{target}

The Perceptron Learning Rule is an algorithm for adjusting the network weights **w** to minimize the difference between the actual and the desired outputs.

We can define a **Cost Function** to quantify this difference:

$$E = \frac{1}{2}\sum_{t=1}^{n}(d^{(t)} - \varphi(w^T s^{(t)})^2$$

Square makes error positive and penalises large errors more
- ½ just makes the math easier
- Need to change the weights to minimize the error – How?

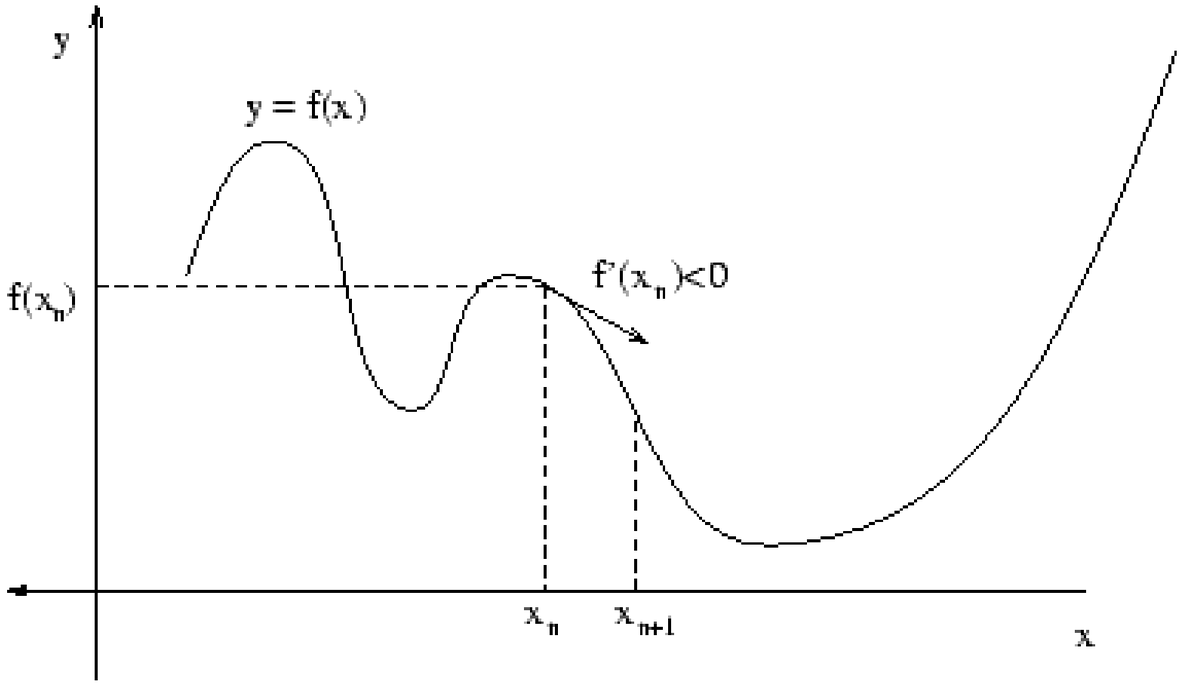
:Use principle of **Gradient Descent**

Training the Neural Network: Widrow-Hoff algorithm

$$E = \frac{1}{2}\sum_{t=1}^{n}(d^{(t)} - \varphi(w^T s^{(t)})^2$$

We use *gradient descent* to search for a good set of weights

Problem formulation: we don ’t know the formel expression of *f* but if *x* is setting it is possible to compute *f(x)*



Initialize the initial position x0 at random

$$x_{n+1} = x_n - \alpha f'(x_n)$$

Repeat until convergence

α is called the **learning rate** or **step size** and it determines how smoothly the learning process is taking place

Training the Neural Network: Widrow-Hoff algorithm

Calculate the derivative (gradient) of the Cost Function with respect to the weights, and then change each weight by a small increment in the negative (opposite) direction to the gradient

The derivative of a sum → the sum of the derivatives

$$E = \frac{1}{2}\sum_{t=1}^{n}(d^{(t)} - \varphi(w^T s^{(t)})^2$$

Sum on training data samples

To simplify the computation

$$E = \tfrac{1}{2}(d - \varphi(w^t x))^2$$

Error defined for one training data samples

For each weight

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial w_j}$$

$$\frac{\partial E}{\partial y} = \frac{\partial\left[\frac{1}{2}(d-y)^2\right]}{\partial y} = -(d-y)$$

$$\frac{\partial y}{\partial v} = \frac{\partial \varphi(v)}{\partial v} = \varphi'(v)$$

$$\frac{\partial v}{\partial w_j} = \frac{\partial\left[\sum_{j=1}^{d} w_j x(j)\right]}{\partial w_j} = x(j)$$

To reduce E by gradient descent, move/increment weights in the negative direction to the gradient

$$\frac{\partial E}{\partial w_j} = -(d - \varphi(w^t x)).\varphi'(w^t x).x(j)$$

$$x_{n+1} = x_n - \alpha f'(x_n)$$

For each weight

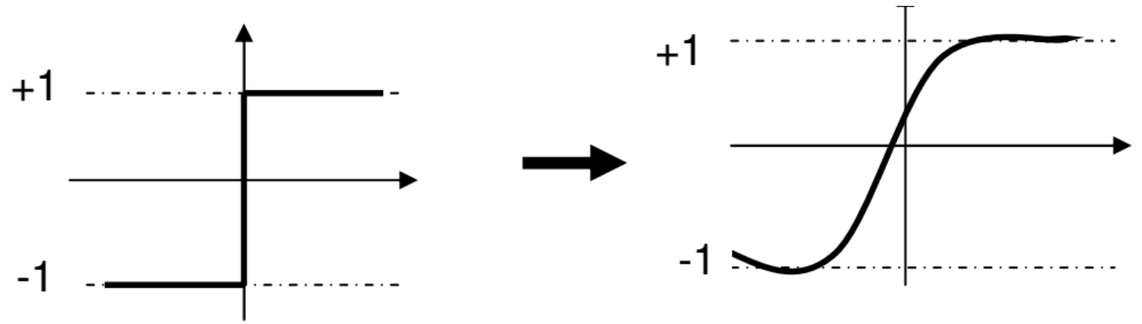
$$w_j^{n+1} = w_j^n - \alpha \left\{ -(d - \varphi(w^t x)).\varphi'(w^t x).x(j) \right\}$$

Delta rule DR or Widrow-Hoff rule is similar to the simple Perceptron Learning Rule (PLR), with some differences:

1. Error in DR is not restricted to having values of 0, 1, or -1 (as in PLR), but may have any value
2. DR can be derived for any *differentiable* output/activation function *f*, whereas in PLR only works for threshold output function

$$\varphi'(x)$$

The activation function have to be differentiable



Background of the backpropagation

$$w^{n+1} = w^n - \alpha r x$$

$$w_j^{n+1} = w_j^n - \alpha \left\{ -(d - \varphi(w^t x)).\varphi'(w^t x).x(j) \right\}$$

$$r = \frac{\partial E}{\partial v} = -(d - y)\varphi'(v)$$

$$w^{n+1} = w^n - \alpha r x$$

Training Strategy

$$w^{n+1} = w^n - \alpha r x$$

On-line Training (or Sequential Training): update all the weights immediately after processing each training pattern

First definition of the error

$$E = \frac{1}{2}\sum_{t=1}^{n}(d^{(t)} - \varphi(w^T s^{(t)})^2$$

Sum on training data samples

Batch Training: update the weights after all training patterns have been presented

$$\Delta w_j = \sum_{t=1}^{n} \Delta w_j^{(t)}$$

Notion de traitement par mini-lots

- Epoch: the number of times the model is exposed to the training set
- Batch_size: the number of training instances observed before the optimizer performs a weight update

Activation function

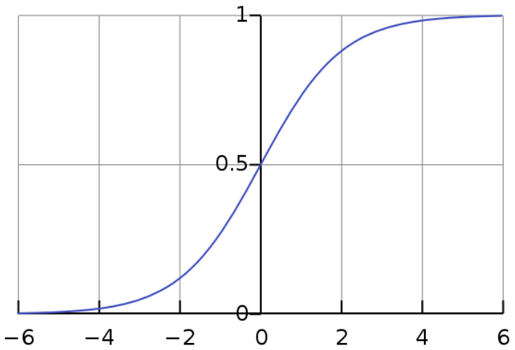
A differentiable transfer/activation function is necessary for the gradient descent algorithm to work.

For example the standard sigmoid (i.e. logistic function) is a particularly convenient replacement for the step function of the Simple Perceptron

y in [0,1]

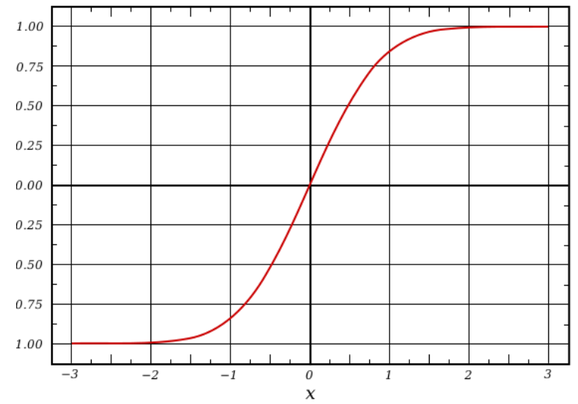
Sigmoid Unit (Neuron)

$$\varphi(x) = \frac{1}{1+e^{-x}} \quad \Rightarrow \quad \varphi'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \varphi(x) - (\varphi(x))^2$$



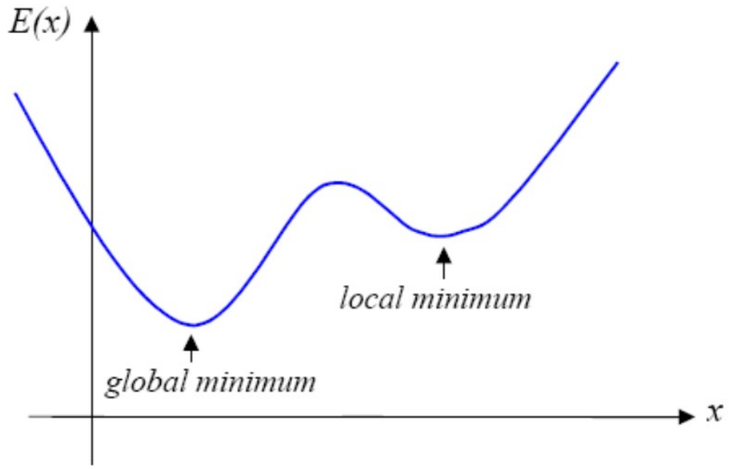
$$\varphi(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad \Rightarrow \quad \varphi'(x) = 1 - (\varphi(x))^2$$

y in [-1,1]



Choosing the Initial Weight Values

Cost functions can quite easily have more than one minimum:



If we start off in the vicinity of the local minimum, we may end up at the local minimum rather than the global minimum.

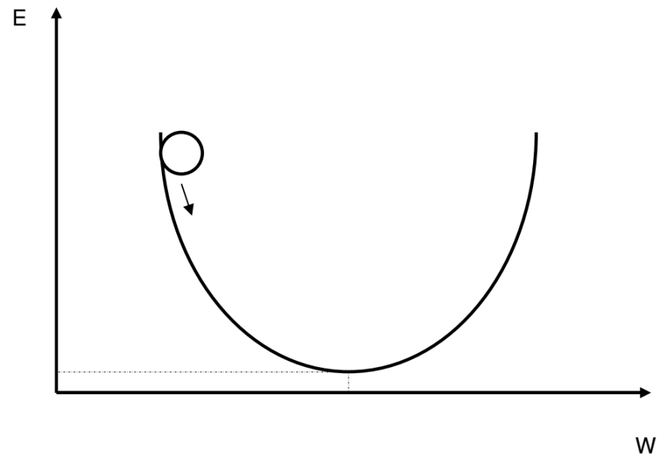
Starting with a range of different initial weight sets increases our chances of finding the global minimum.

Choosing the Learning Rate

$$w_j^{n+1} = w_j^n - \alpha \left\{ -(d - \varphi(w^t x)).\varphi'(w^t x).x(j) \right\}$$

Choosing a good value for the learning rate is constrained by two opposing facts:

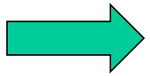
1. If α is too small, it will take too long to get anywhere near the minimum of the error function.
2. If α is too large, the weight updates will over-shoot the error minimum and the weights will oscillate, or even diverge.



Unfortunately, the optimal value is very problem and network dependent, so one cannot formulate reliable general prescriptions.

Gradient Stochastique

Choix aléatoire à chaque itération d'un individu de l'ensemble d'apprentissage



Permet de sauter hors de minima locaux
Comportement irrégulier lorsque l'on approche de la solution (ne décroit qu'en moyenne)

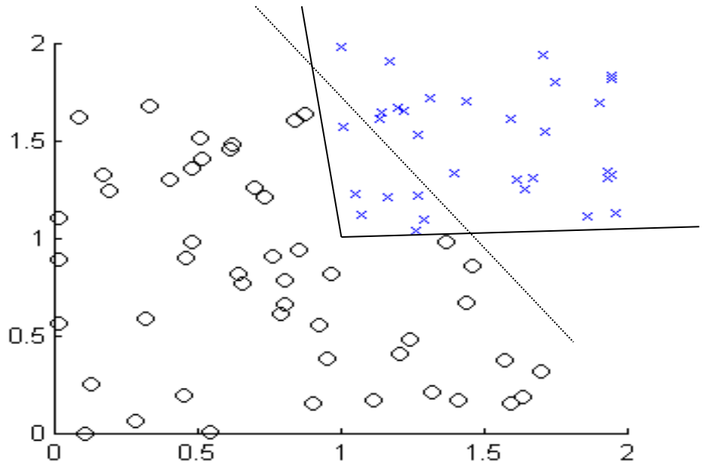
Réduire le coefficient d'apprentissage au fur et à mesure des itérations

Introduire la notion de mini-lots : régularise la descente mais risque de minima locaux

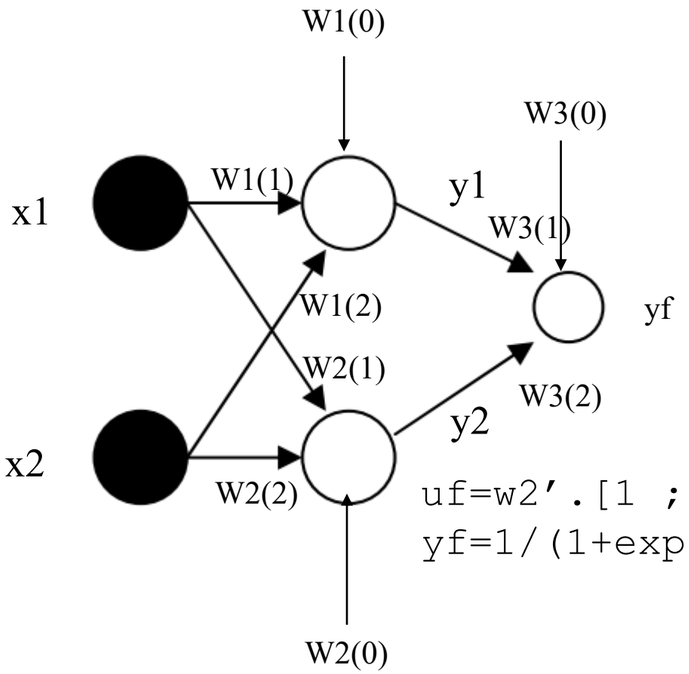
Multilayer neural network: why ?

Neuron defines two regions in input space where it outputs 0 and 1.
The regions are separated by a hyperplane

$$w_1 x_1 + \ldots + w_d x_d + w_0 = 0$$

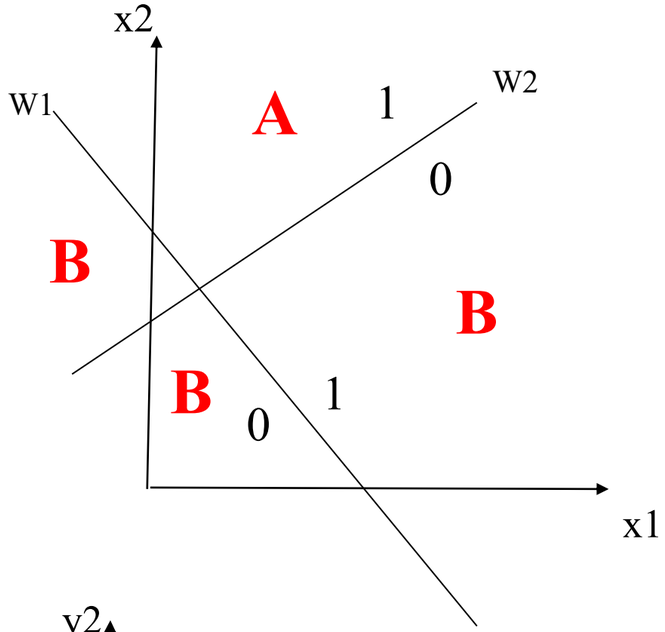


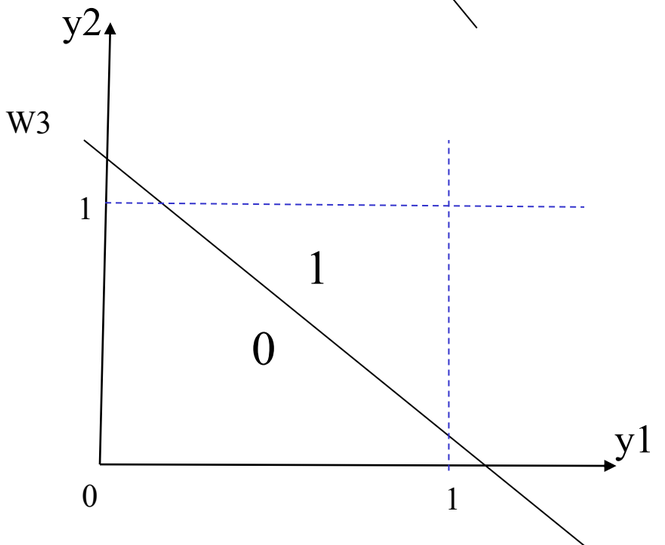

```
u1=w1'.[1 ; x]
y1=1/(1+exp(-1*u1))
```




```
uf=w2'.[1 ; y1 ; y2]
yf=1/(1+exp(-1*uf))
```

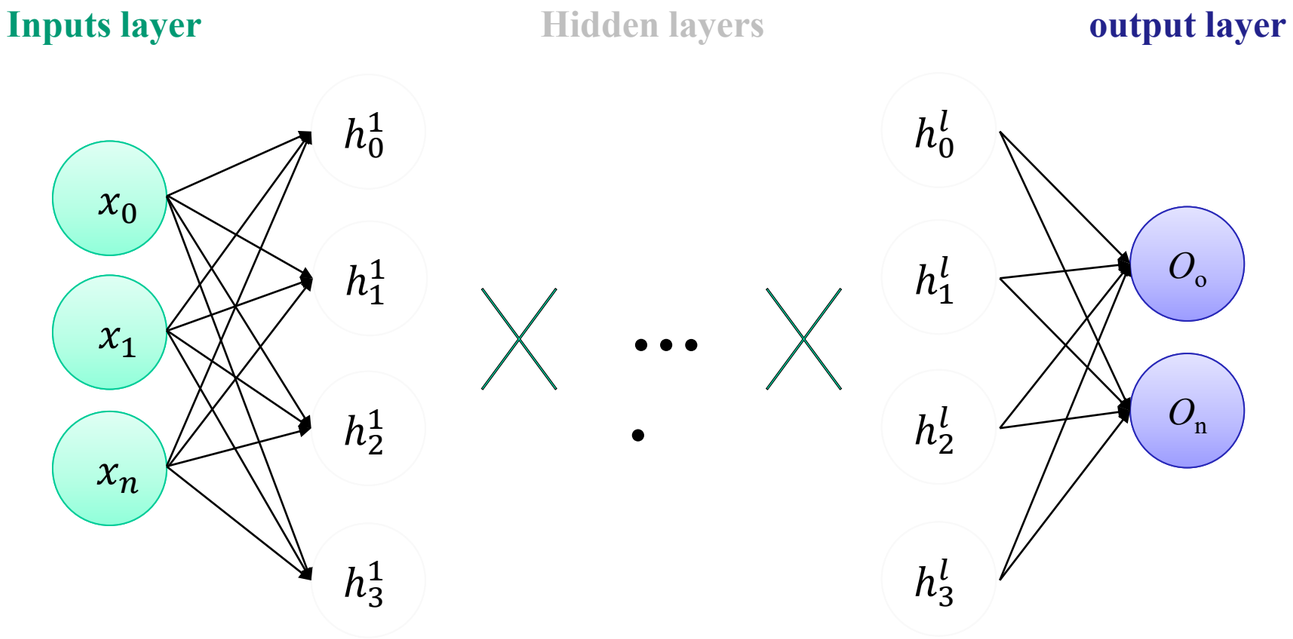
```
u2=w2'.[1 ; x]
y2=1/(1+exp(-1*u2))
```





“Deep” Neural Network

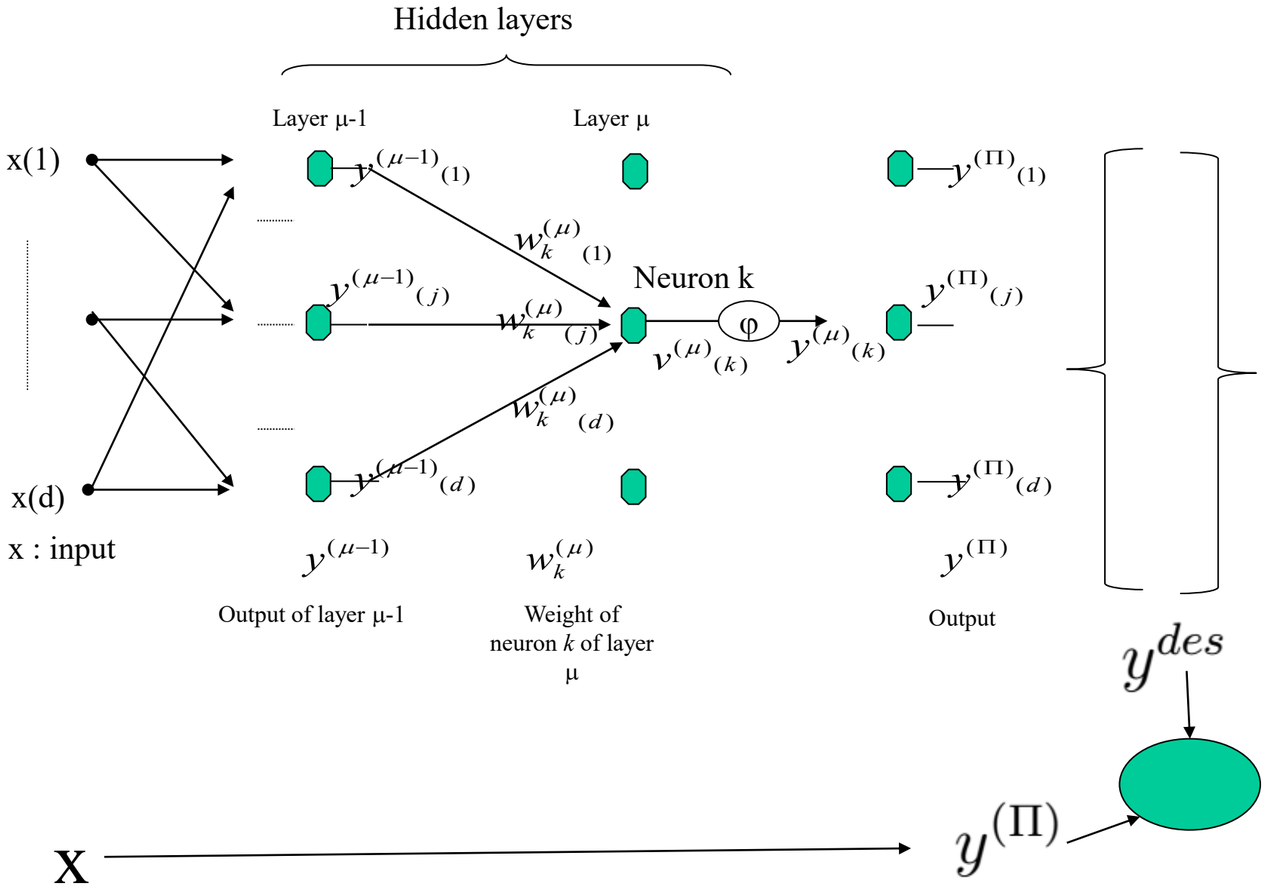
- Go deeper, add more layers…
- Fully connect the weights between layers



Training the Multilayer Neural Network: Widrow-Hoff algorithm

The Multilayer Perceptron Learning Rule is an algorithm for adjusting the network weights **w** to minimize the difference between the actual and the desired outputs.

We can define a **Cost Function** to quantify this difference



Error defined for one training data samples

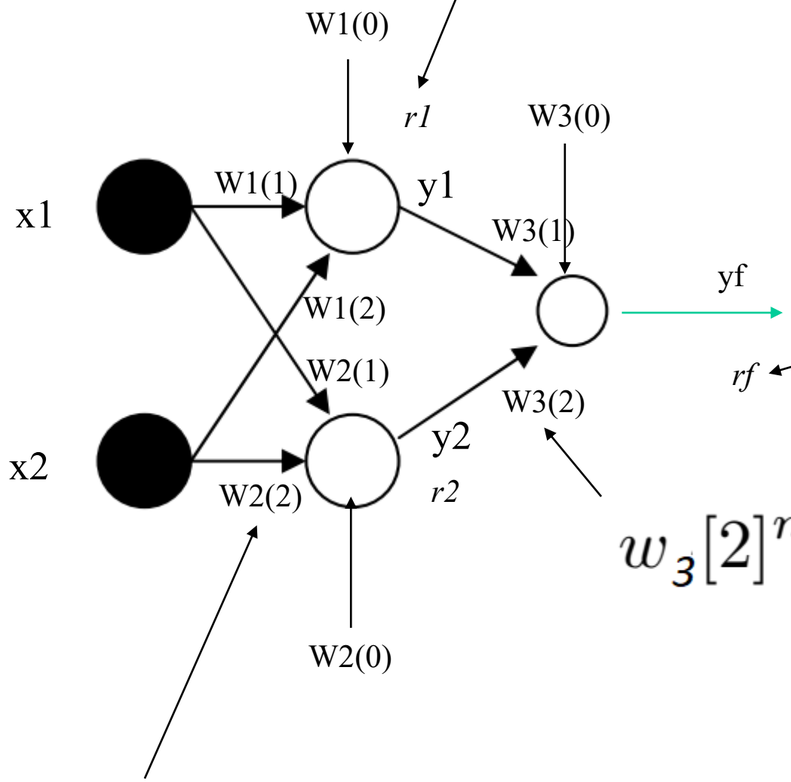
$$E = \frac{1}{2}(y^{des} - y^{(\Pi)})^t(y^{des} - y^{(\Pi)}) = \frac{1}{2}\sum_{j=1}^{d}(y^{des}(j) - y^{(\Pi)}(j))^2$$

Training Multilayer neural network: Backpropagation

$$r_1 = (w_3[1] * r_f)\,\varphi'(v_1)$$

$$w^{n+1} = w^n - \alpha r x$$

$$r = \frac{\partial E}{\partial v}$$



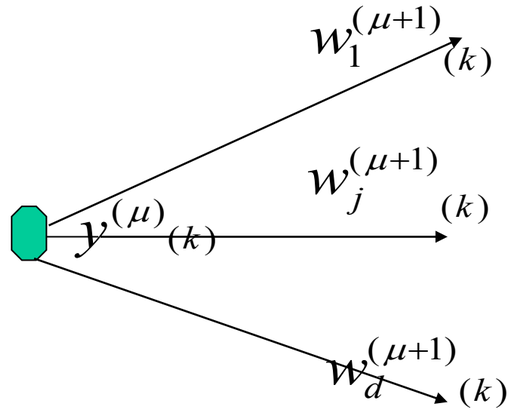
$$r_f = \frac{\partial E}{\partial v_f} = -(y_d - y_f)\varphi'(v_f)$$

$$w_3[2]^{n+1} = w_3[2]^n - \alpha\,[\quad -(y_d - y_f)\varphi'(v_f)\quad]\,y_2$$

$$w_2[2]^{n+1} = w_2[2]^n - \alpha\,[w_3[2] * r_f * \varphi'(v_2)]\,x_2$$

Training the Multilayer Neural Network: hidden layer

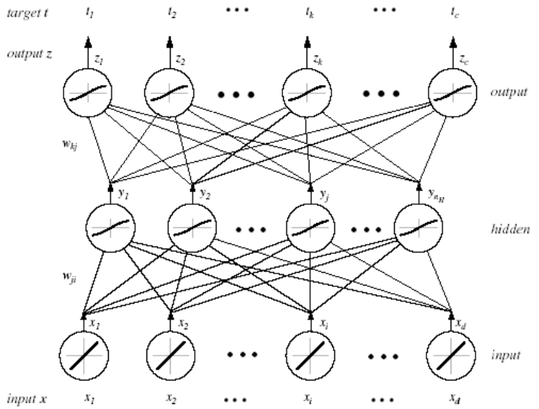
$$Err^{(\mu)}(k) = \varphi'(v^{(\mu)}(k)) \sum_{j \in \text{ couche } \mu+1} w_j^{(\mu+1)}(k) Err^{(\mu+1)}(j)$$



$$\Delta w_k^{(\mu)}(j) = -\alpha . Err^{(\mu)}(k) . y^{(\mu-1)}(k) = -\alpha . \left[\varphi'(v^{(\mu)}(k)) \sum_{j \in \text{ couche } \mu+1} w_j^{(\mu+1)}(k) Err^{(\mu+1)}(j) \right] . y^{(\mu-1)}(j)$$

Output

Propagation
Computation of the output



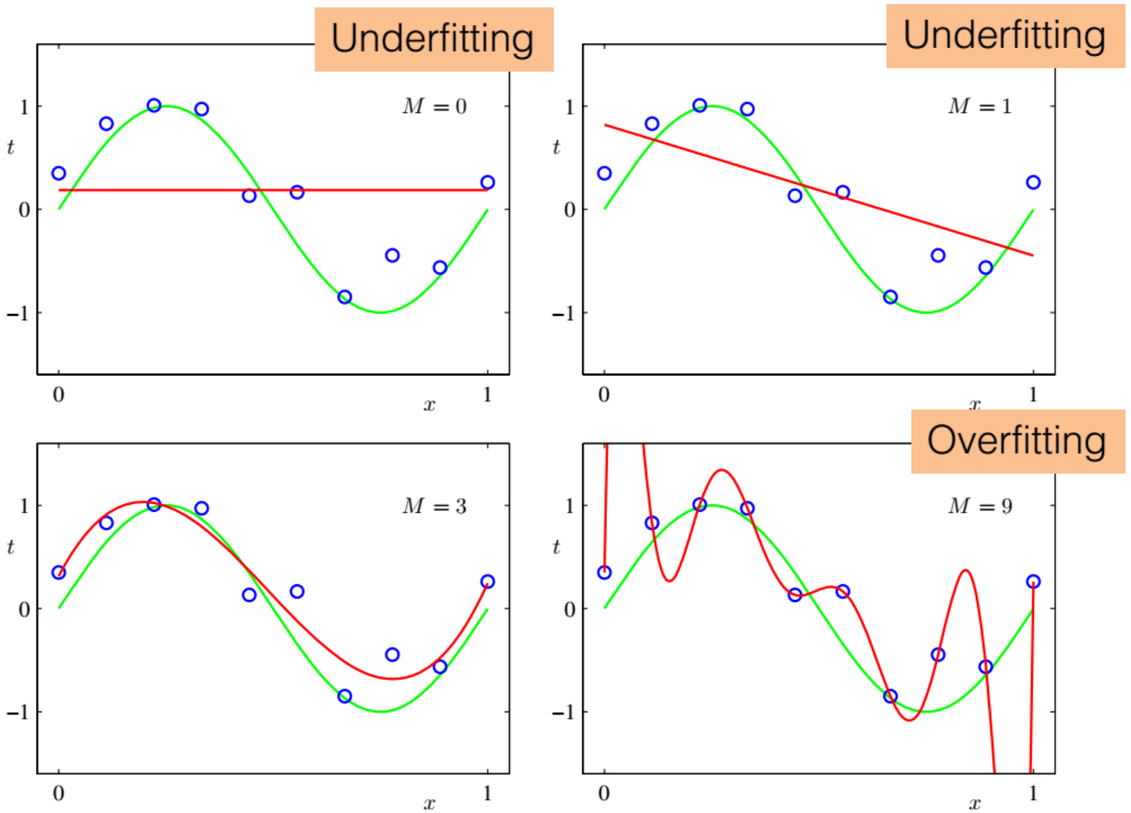
Backpropagation
Learning process
with the propagation of the error

Input

Number of parameters ?

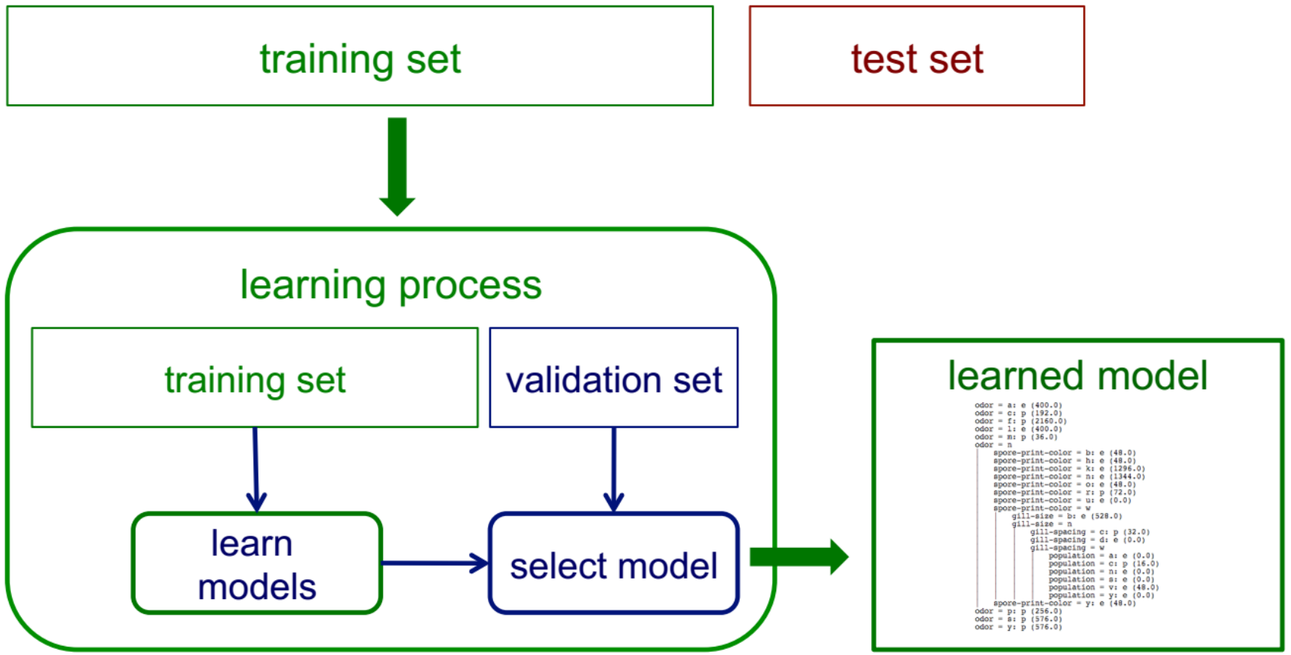
Illustration: model fitting and generalization

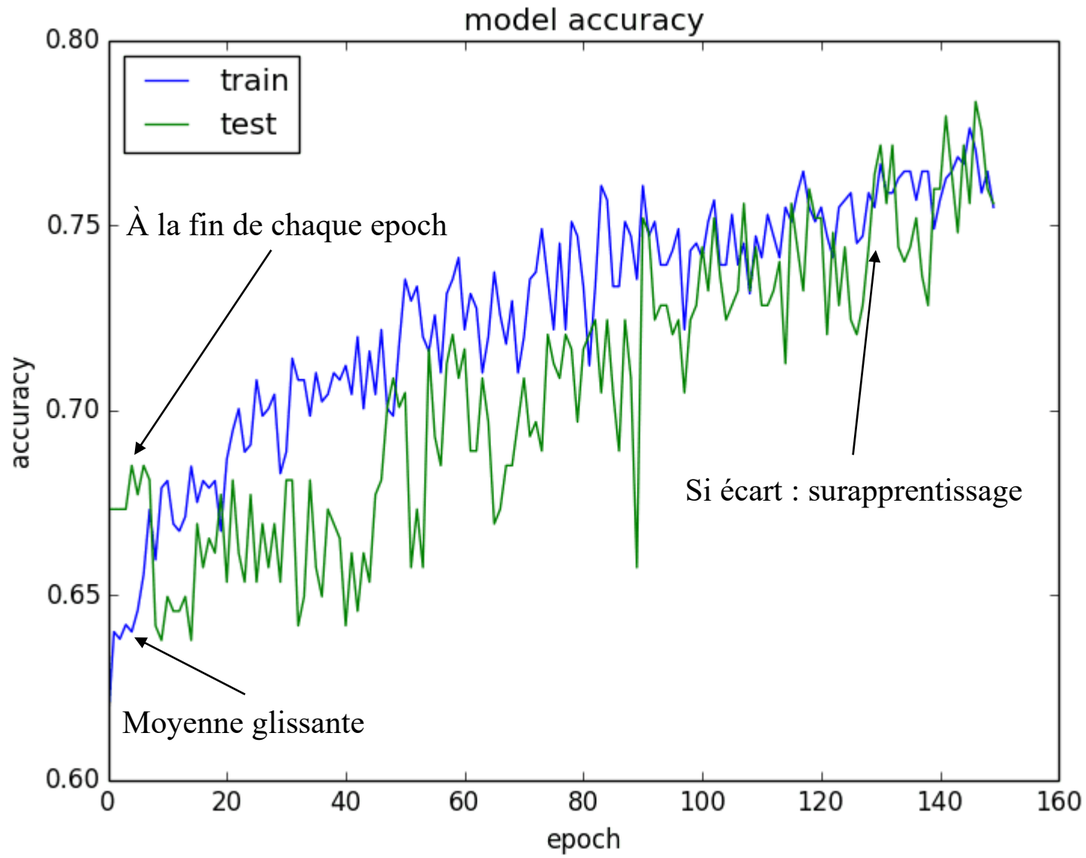
How do we chose model complexity?



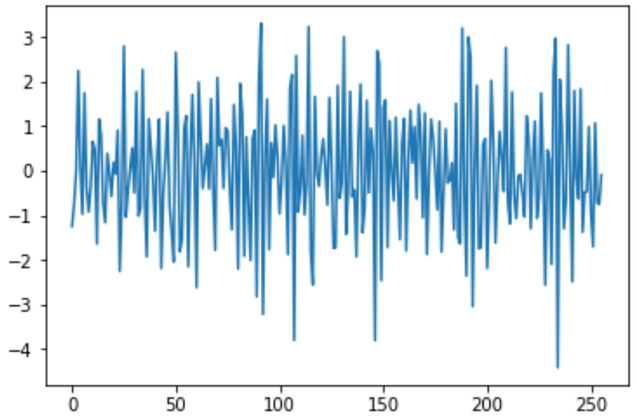
Number of parameters ?

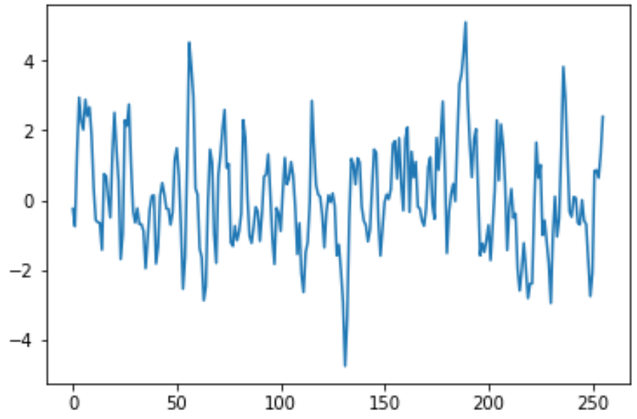
```
# Create model
model = Sequential()
# Add layers
model.add(...)

# Train model (use 10% of training set as validation set)
history = model.fit(X_train, Y_train, validation_split=0.1)

# Train model (use validation data as validation set)
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test))
```



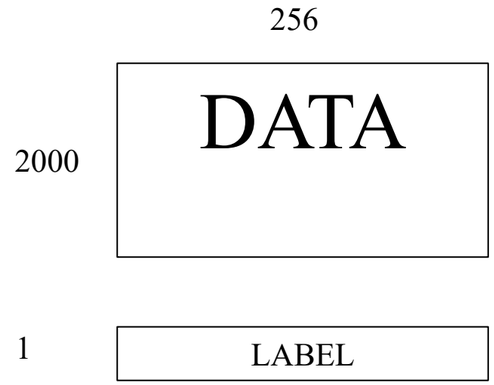


Example



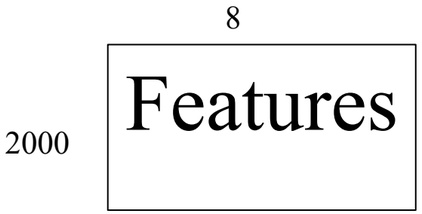


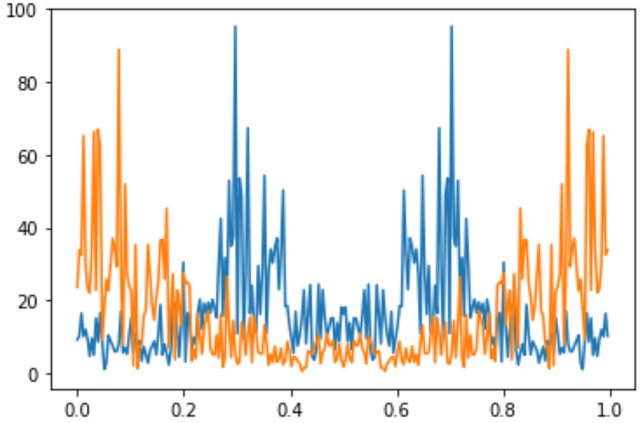
```
lg=256
f1=0.3
f2=0.1

x=randn(lg)
b=[1]
a=poly((0.8*(cos(2*pi*f1)+sin(2*pi*f1)*1j),0.8*(cos(2*pi*f1)-sin(2*pi*f1)*1j)))
y1=signal.lfilter(b,a,x)

x=randn(lg)
b=[1]
a=poly((0.6*(cos(2*pi*f2)+sin(2*pi*f2)*1j),0.6*(cos(2*pi*f2)-sin(2*pi*f2)*1j)))
y2=signal.lfilter(b,a,x)
```



Features : Energy for different intervals in the frequency domain

```
for k in range(0,Nbre_indivu*2):
    spec = abs(fft(Data[k,:]))**2
    for kk in range(0,8):
        sslg=int(lg/(8*2))
        Features[k,kk]=np.sum(spec[kk*sslg:(kk+1)*sslg])
```

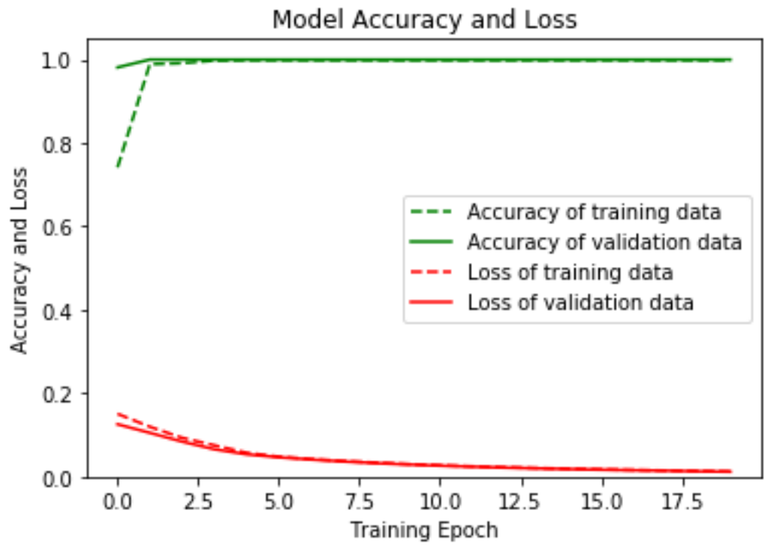




Learning and Test

```
model = Sequential()

#**********************************
# Discriminateur couche 1+2
#**********************************

model.add(Dense(8, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train, Y_train, epochs=30, batch_size=32, validation_split=0.2, verbose=1)
score = model.evaluate(X_test, Y_test, verbose=1)
```

score : [0.0005113882361911237, 1.0]



New Discrimination Process

learning

Learning and Decision

Sensor

Modelisation

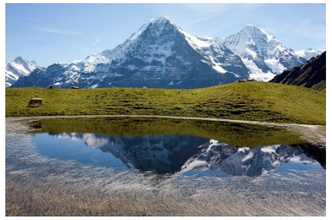
AI

Real Word

Data

Features

Decision

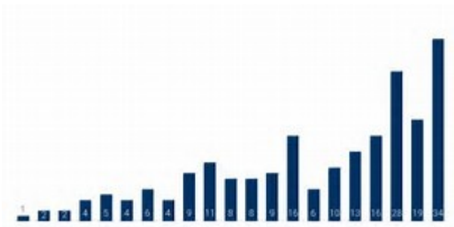


$S = \{s_1, \ldots, s_k, \ldots, s_K\}$

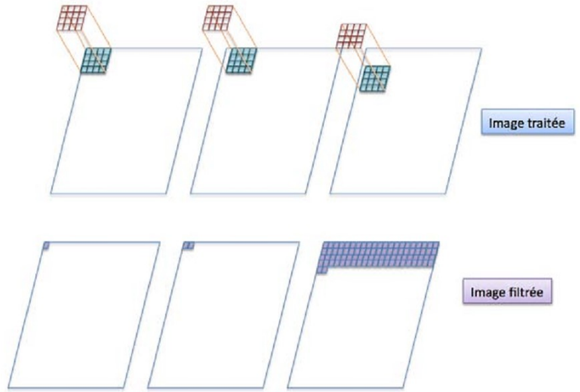
$X = R^d$

$A = \{a_1, \ldots, a_l, \ldots, a_L\}$

Signal



Features computation : Convolution product



$$g(p,q) = \sum_i \sum_j f(p-i, q-j).h(i,j)$$

After filtering

Original image

Convolution Mask

p

q

i

j

f(p-1,q-1)	f(p,q-1)	f(p+1,q-1)
f(p-1,q)	f(p,q)	f(p+1,q)
f(p-1,q+1)	f(p,q+1)	f(p+1,q+1)

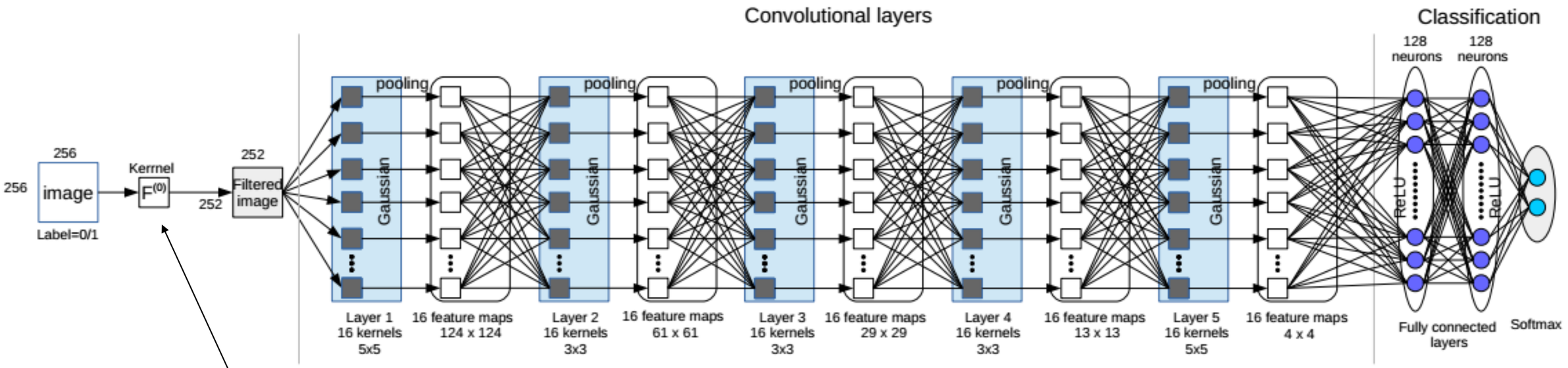
Original

h(-1,-1)	h(0,-1)	h(1,-1)
h(-1,0)	h0,0)	h(1,0)
h(-1,1)	h(0,1)	h(1,1)

Mask

g(p,q)

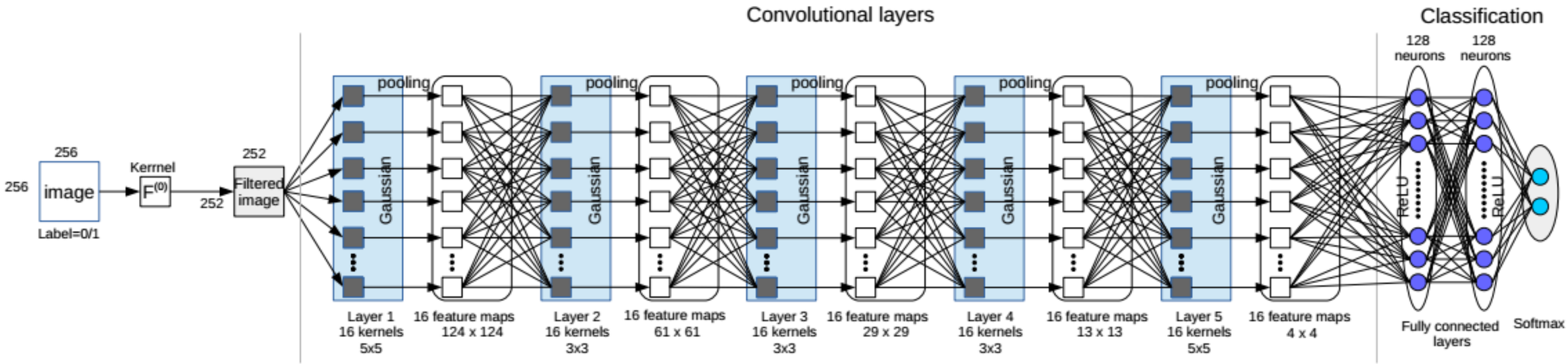
Deep-Learning



$$F^{(0)} = \frac{1}{12}\begin{pmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{pmatrix}$$

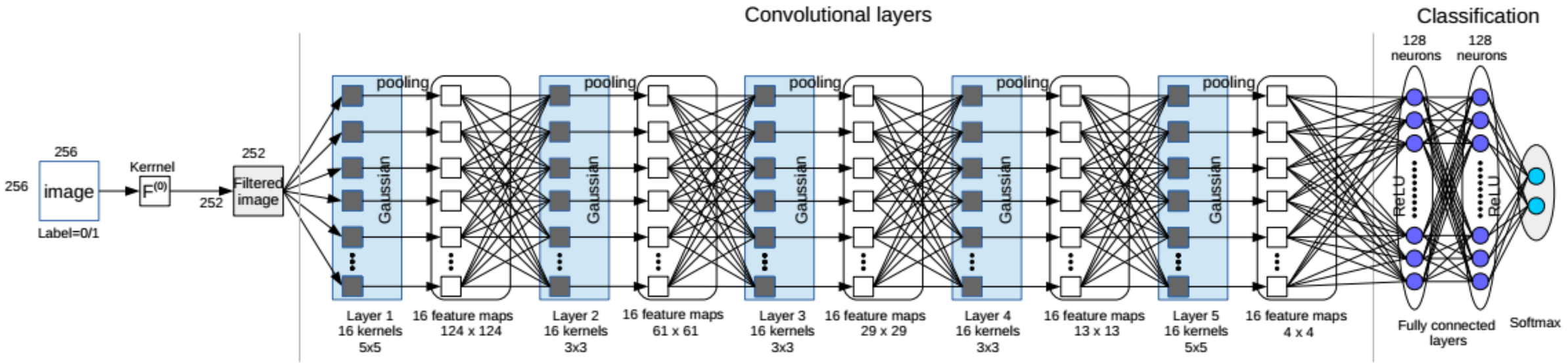
High-pass filter to improve results

Calculation of the features with learning



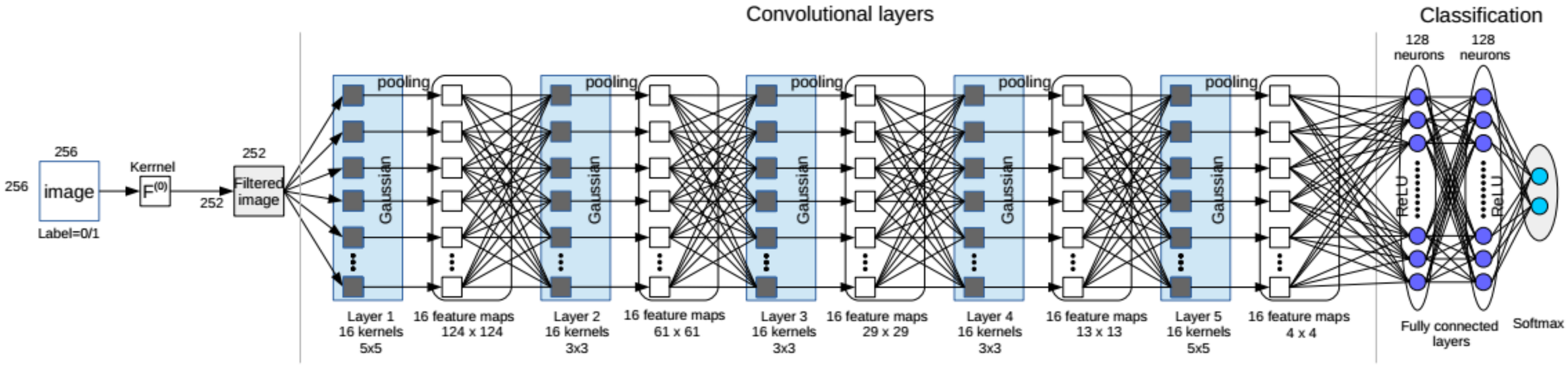
For each block (one layer); we have the following steps:

- Convolution product,
- Activation function,
- Pooling operation.



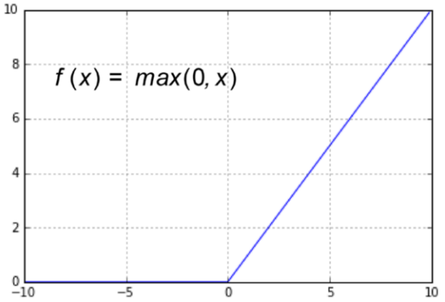
First layer $$\tilde{I}_k^{(1)} = I^{(0)} \star F_k^{(1)}.$$

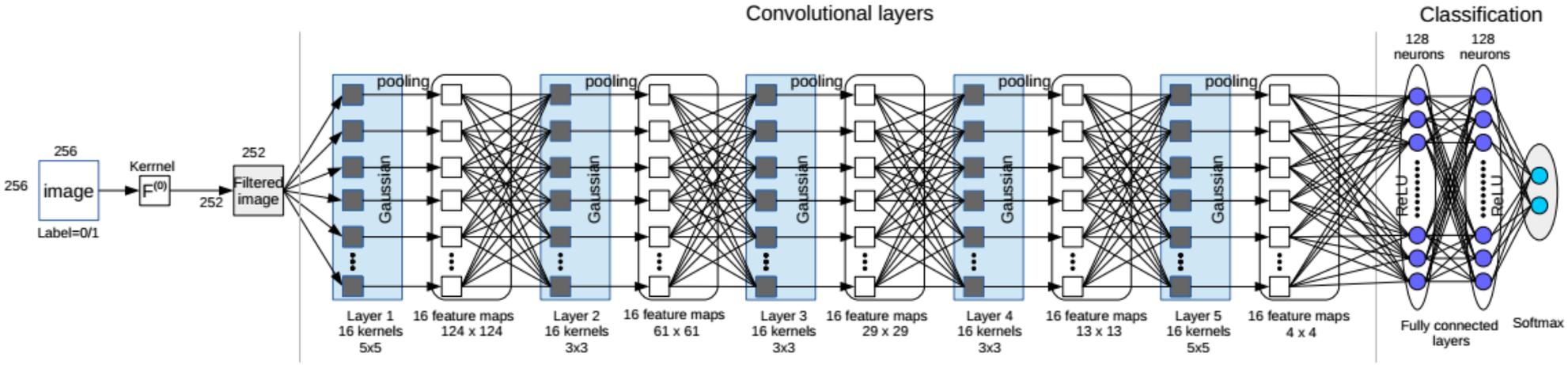
Other layers $$\tilde{I}_k^{(l)} = \sum_{i=1}^{i=K^{(l-1)}} I_i^{(l-1)} \star F_{k,i}^{(l)},$$



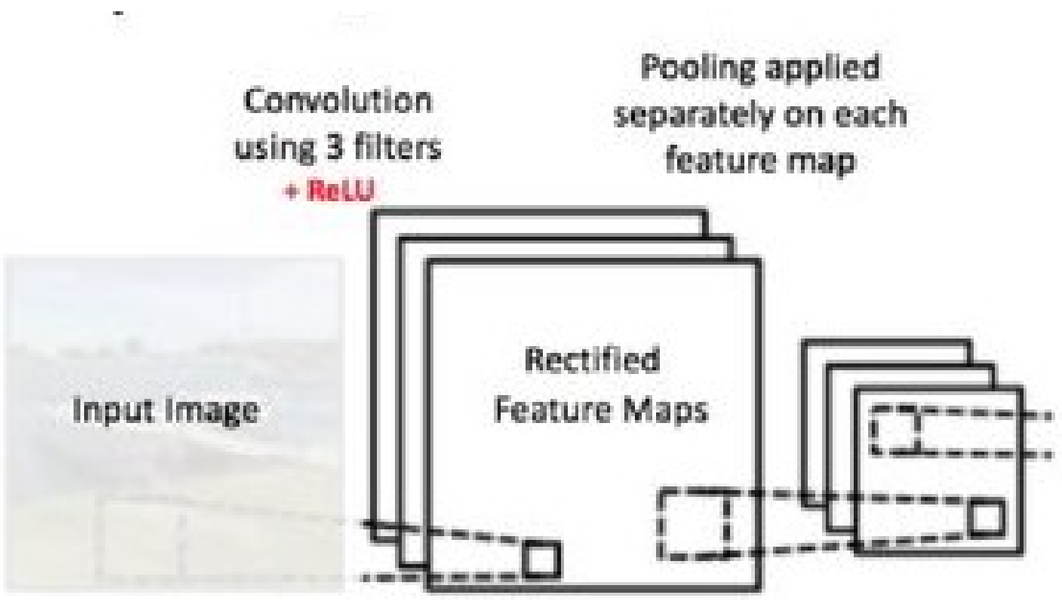
Activation function

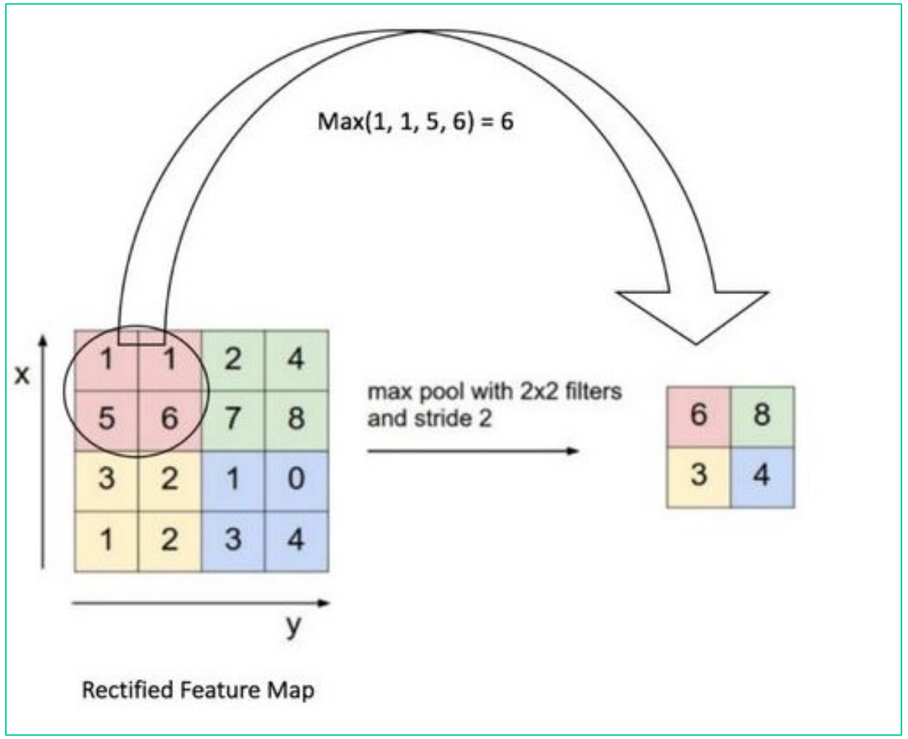
- Fonction absolue : $f(x) = |x|$,
- Fonction sinus : $f(x) = sinus(x)$,
- Fonction Gaussienne (réseau de Qian *et al.*) : $f(x) = \frac{e^{-x^2}}{\sigma^2}$,
- ReLU (pour Rectified Linear Units) : $f(x) = max(0, x)$,
- Tangent hyperbolique : $f(x) = tanh(x)$ ...



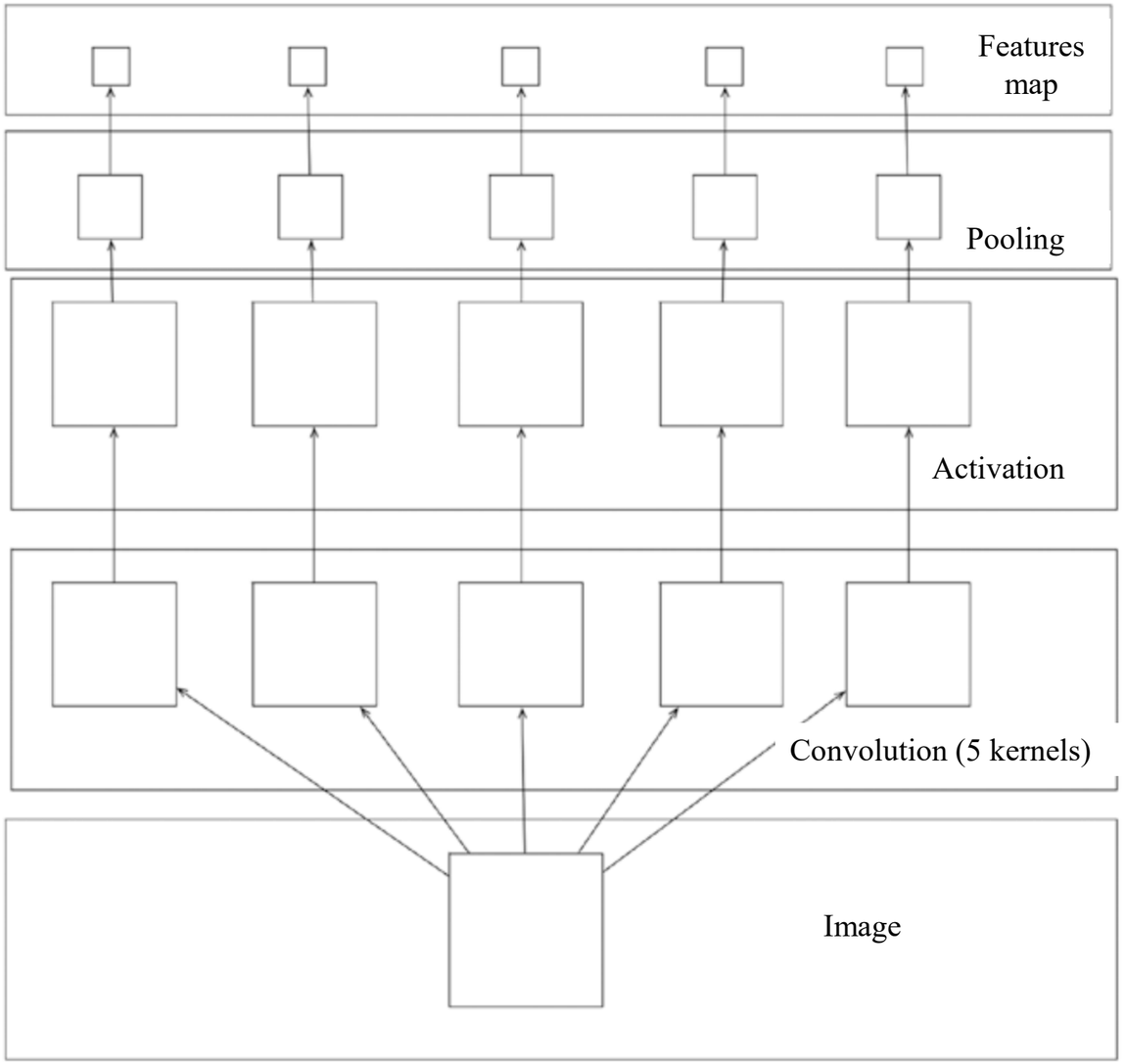


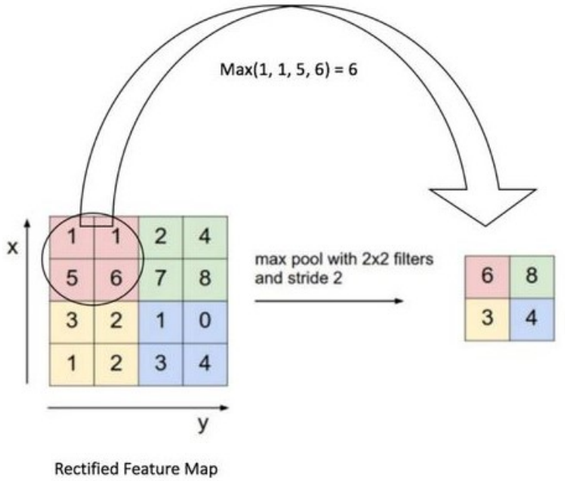
Pooling : local average or local max and downsampling



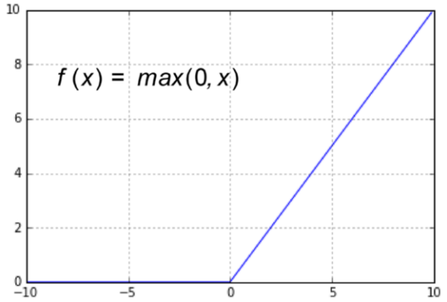


CNN





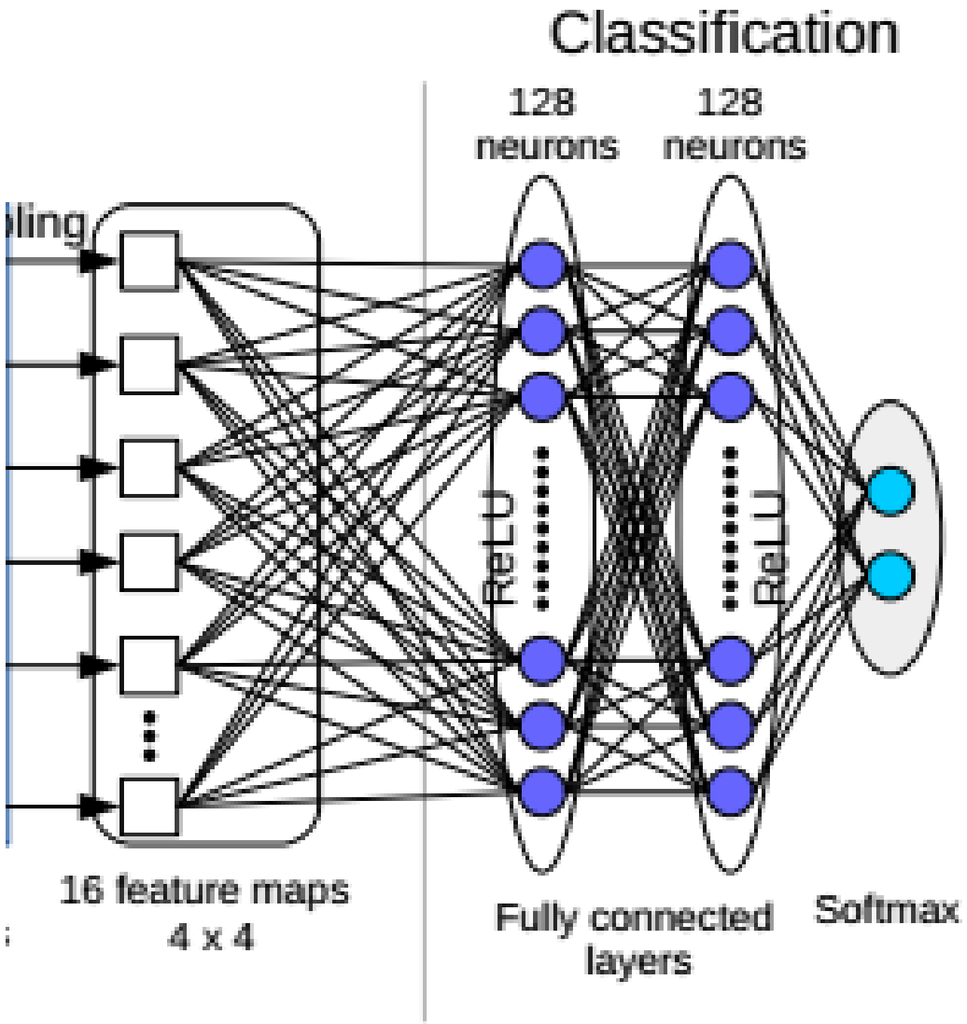
Relu



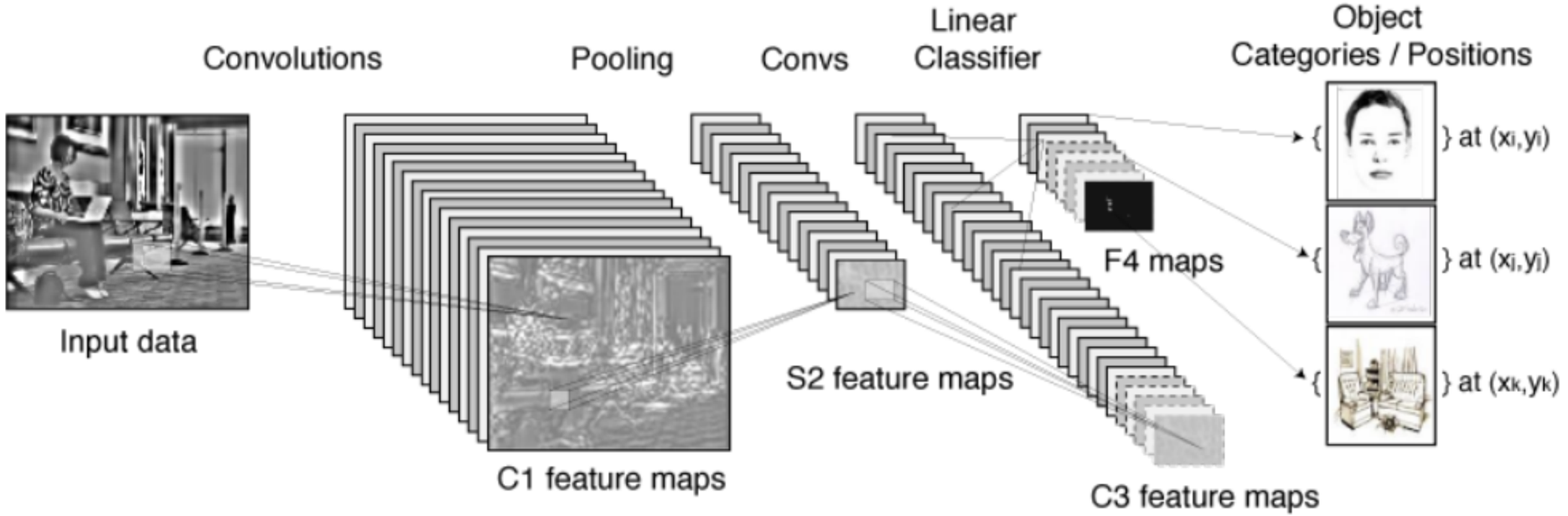
BackPropagation

- **Max-pooling** - the error is just assigned to where it comes from - the "winning unit" because other units in the previous layer's pooling blocks did not contribute to it hence all the other assigned values of zero
- **Average pooling** - the error is multiplied by $\frac{1}{N \times N}$ and assigned to the whole pooling block (all units get this same value).

Second part: classical classification network

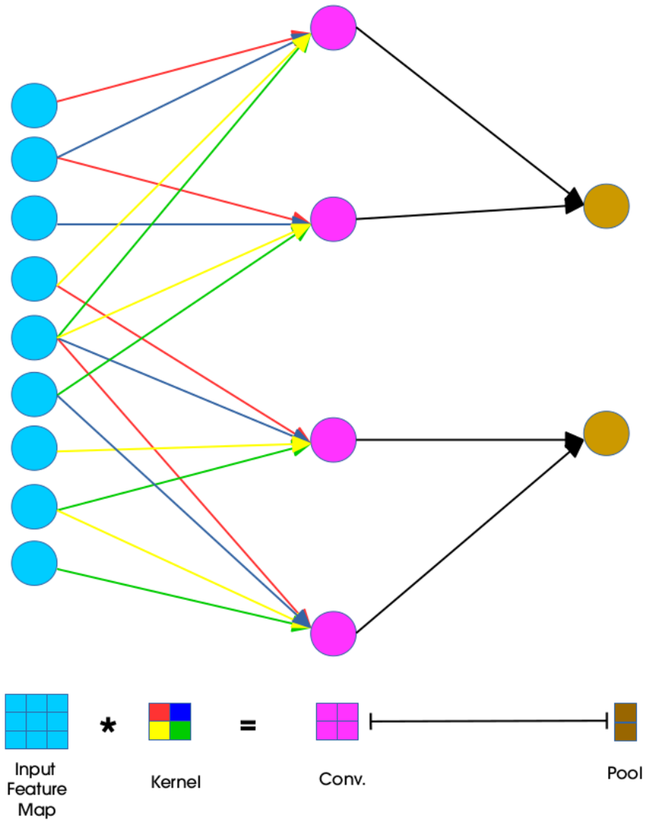


Abstract


Convolutions
Pooling
Convs
Linear Classifier
Object Categories / Positions
Input data
C1 feature maps
S2 feature maps
F4 maps
C3 feature maps
{ } at (xi,yi)
{ } at (xj,yj)
{ } at (xk,yk)

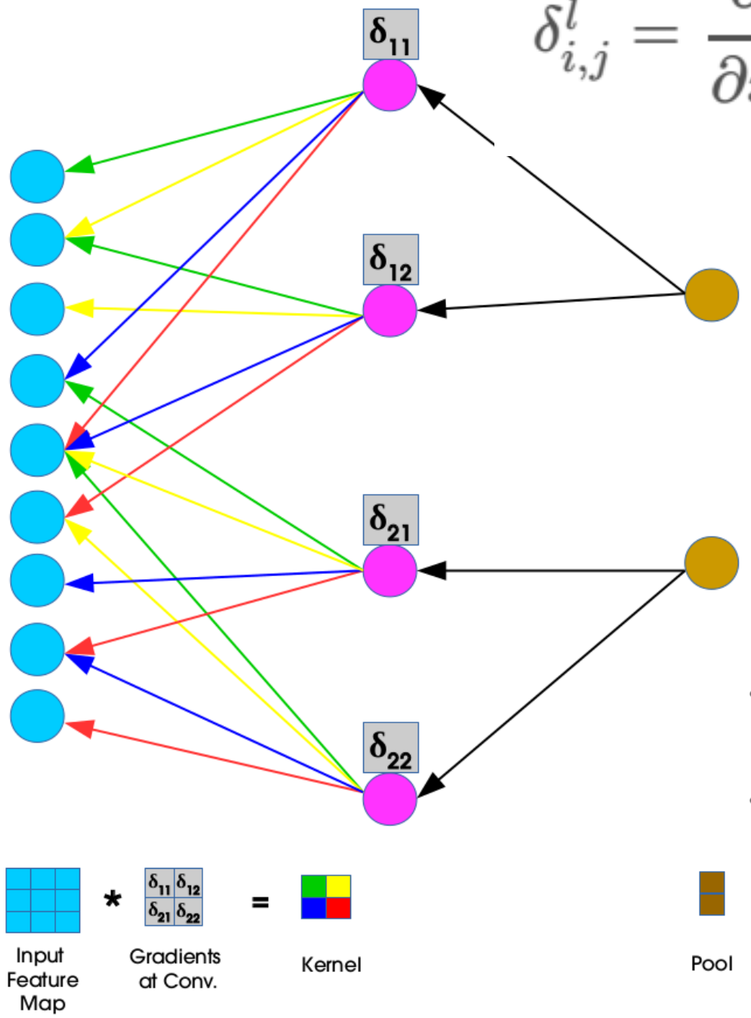
Deep-Learning: training

Propagation



BackPropagation

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}}$$

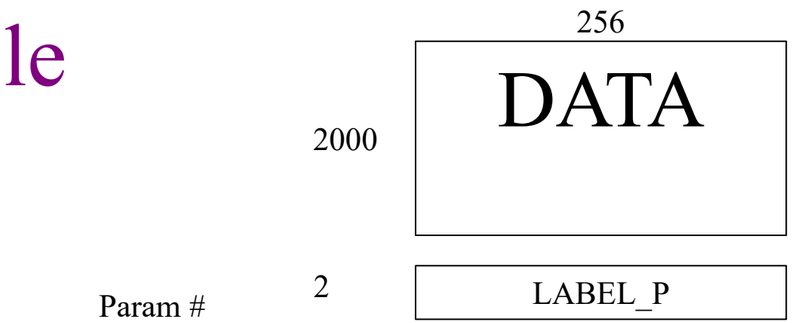


- **Max-pooling** - the error is just assigned to where it comes from - the "winning unit" because other units in the previous layer's pooling blocks did not contribute to it hence all the other assigned values of zero
- **Average pooling** - the error is multiplied by $\frac{1}{N \times N}$ and assigned to the whole pooling block (all units get this same value).

Example

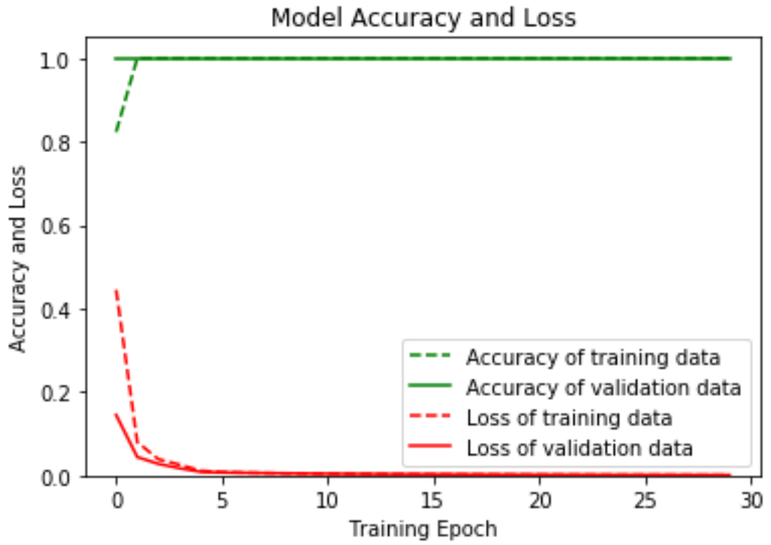
```
#*************************************************************************
# CNN couche 1
#*********************************

model = Sequential()
model.add(Conv1D(filters=4, kernel_size=5, input_shape=(256,1),activation="relu"))
model.add(MaxPooling1D(pool_size=2))

# CNN couche 2
#*********************************
model.add(Conv1D(filters=8, kernel_size=5, activation="relu"))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

#*********************************
# Discriminateur couche 1+2
#*********************************

model.add(Dense(8, activation='tanh'))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#*************************************************************************
#**** Apprentissage/Test
#*************************************************************************

history = model.fit(X_train, Y_train, epochs=30, batch_size=32, validation_split=0.1, verbose=1)
score = model.evaluate(X_test, Y_test, verbose=1)
```

[8.341362496139482e-05, 1.0]

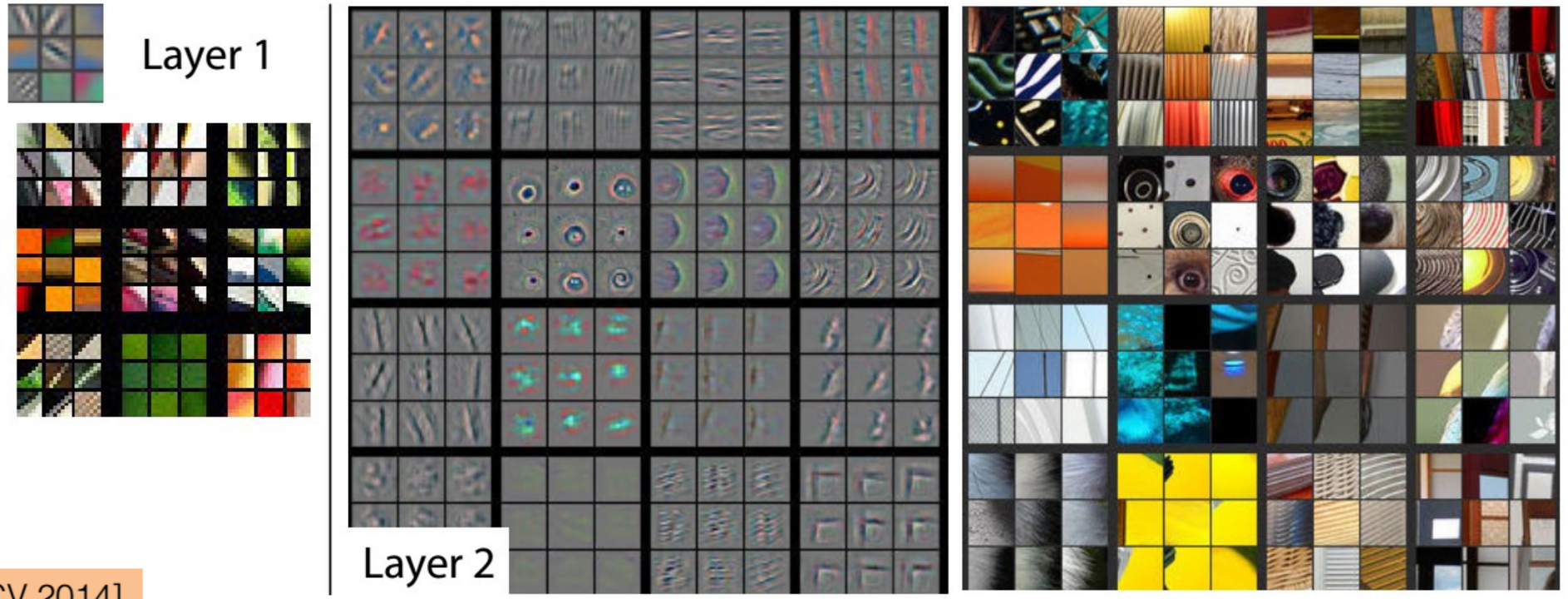


Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 252, 4)	24
max_pooling1d_3 (MaxPooling1	(None, 126, 4)	0
conv1d_4 (Conv1D)	(None, 122, 8)	168
max_pooling1d_4 (MaxPooling1	(None, 61, 8)	0
flatten_2 (Flatten)	(None, 488)	0
dense_7 (Dense)	(None, 8)	3912
dense_8 (Dense)	(None, 2)	18

Total params: 4,122
Trainable params: 4,122
Non-trainable params: 0

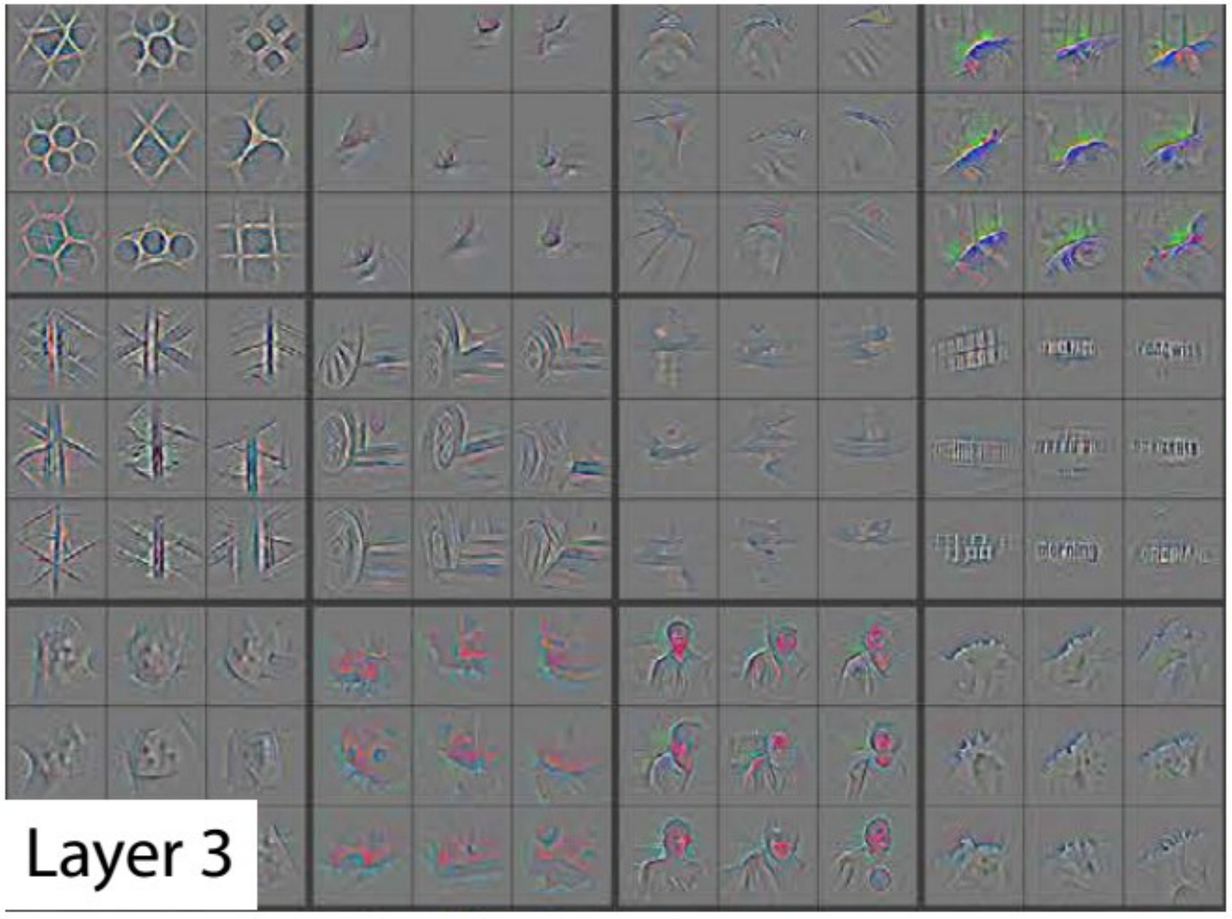


Features ???

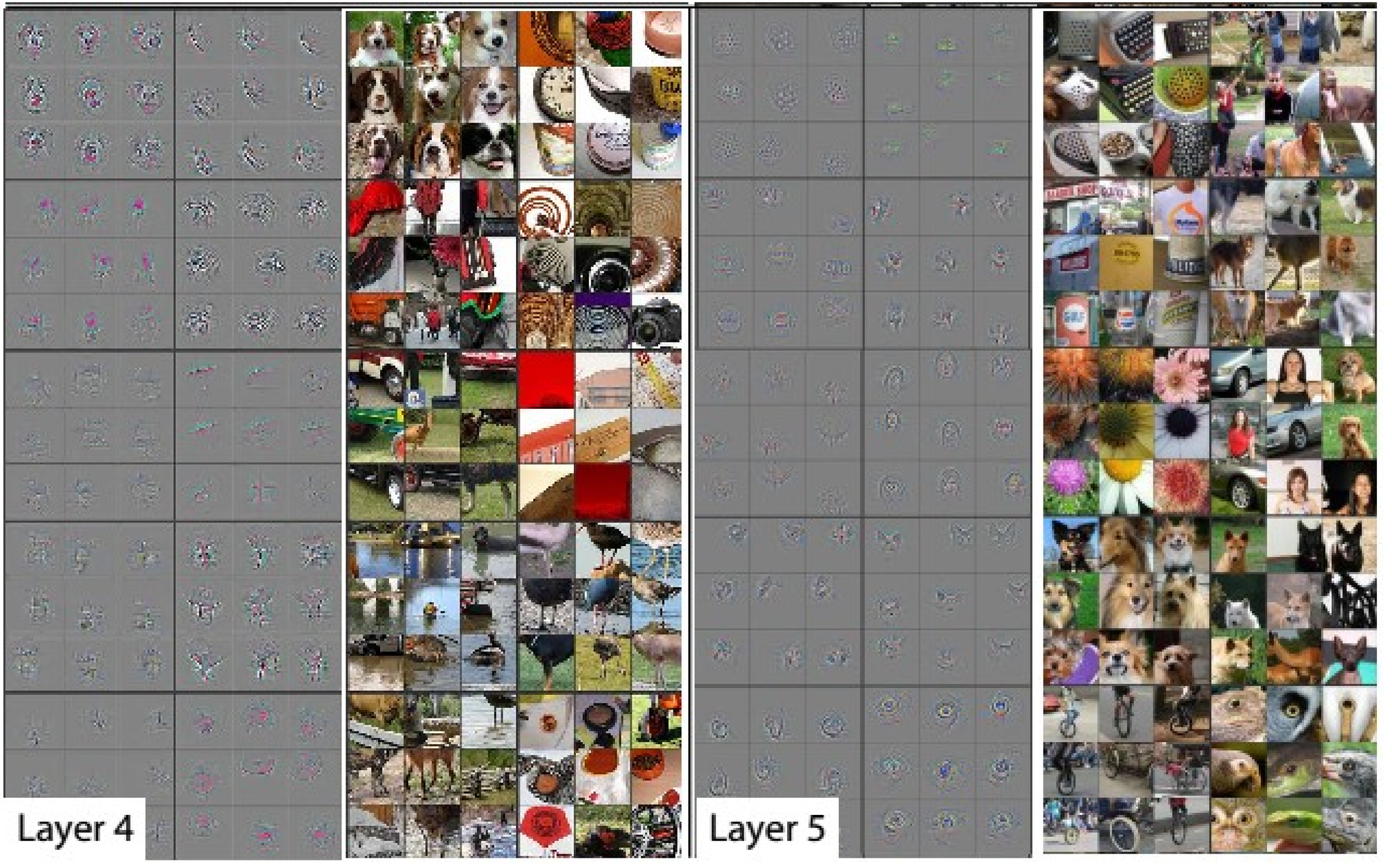


[Zeiler and Fergus, ECCV 2014]

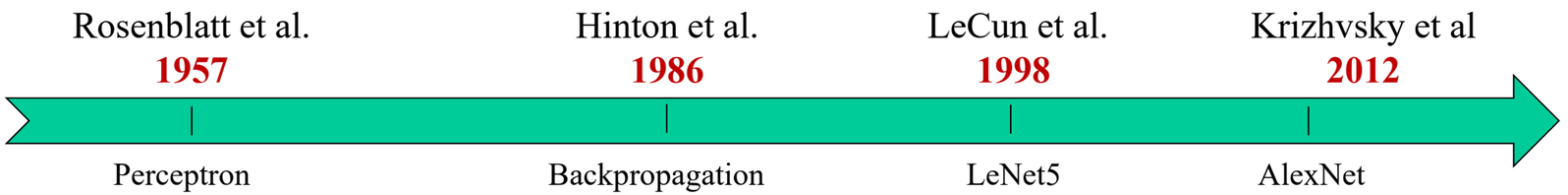
Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.



Layer 3


Layer 4
Layer 5

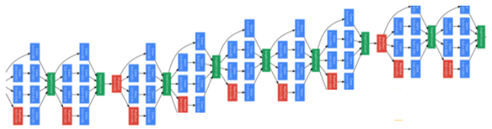
Pourquoi maintenant?



- Larges bases de données annotées
  - disponibilité
  - Capacité de stockage

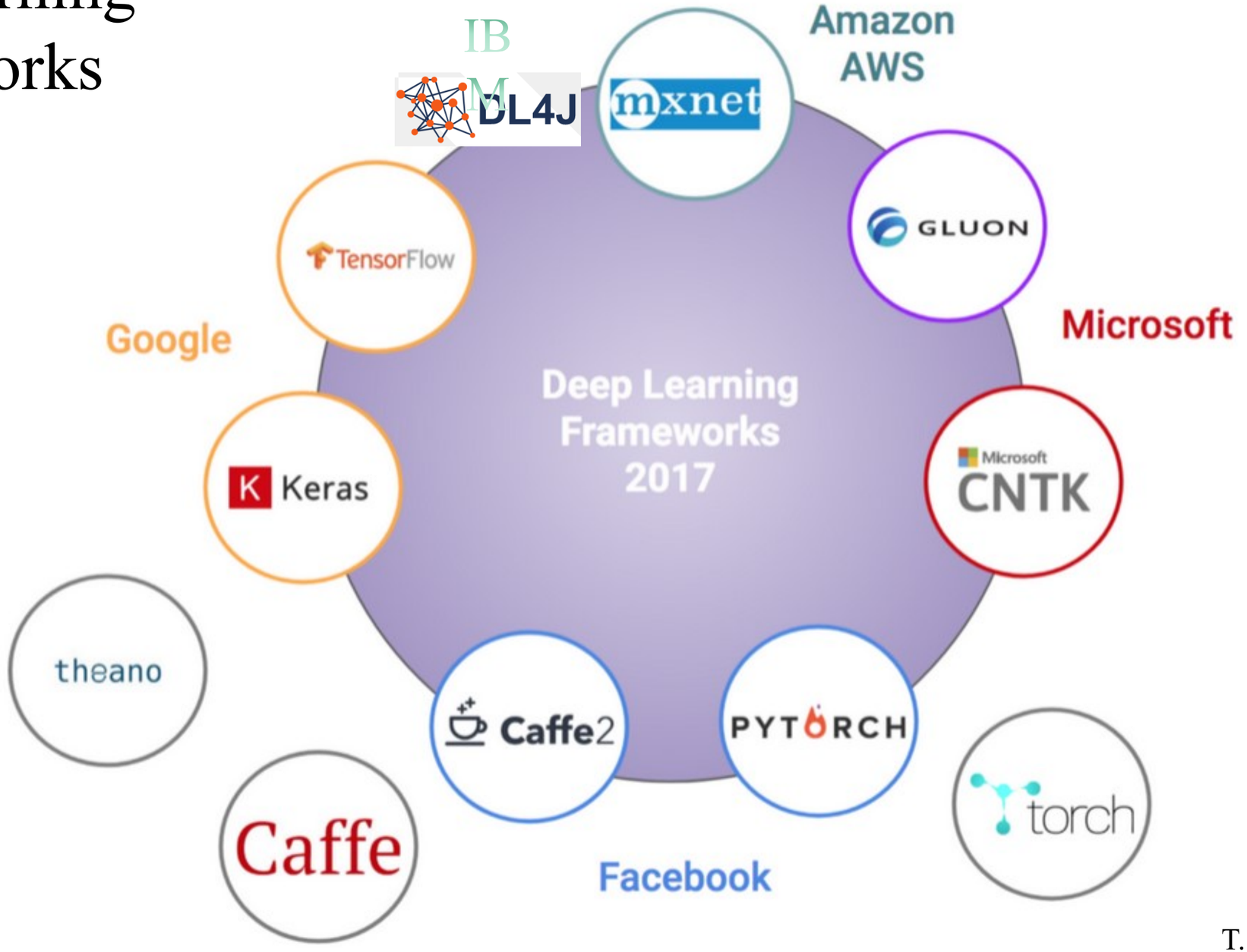
- Avancement GPU
  - Diminution importante du temps de calcul
  - Service Cloud GPU
    - achat temps gpu/cpu à Amazon, Google, Ibm, etc.

- Software
  - Techniques améliorées
  - Modélisation adaptée
  - Open source
  - Tutoriels disponibles
  - Cours en ligne



- Utilisation des réseaux de neurones devenue possible !

Deep Learning Frameworks



T. Urruty (XLIM)

Ref : https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

Evaluating Machine Learning Methods

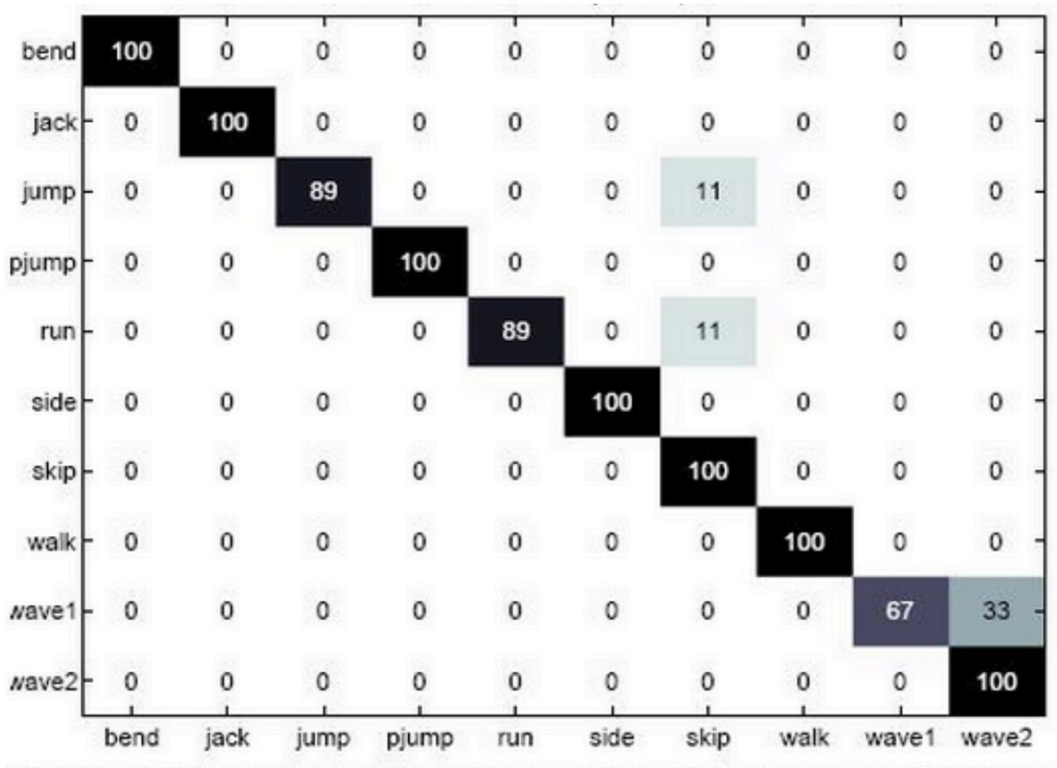
Confusion matrices

How can we understand what types of mistakes a learned model makes?

- The *confusion matrix*:

predicted class \ actual class	positive	negative
positive	true positives (TP)	false positives (FP)
negative	false negatives (FN)	true negatives (TN)

activity recognition from video



Confusion matrix for 2-class problems

predicted class \ actual class	positive	negative
positive	true positives (TP)	false positives (FP)
negative	false negatives (FN)	true negatives (TN)

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy may not be useful measure in all cases

there are differential misclassification costs – getting a positive wrong costs more than getting a negative wrong : Consider a medical domain in which a false positive results in an extraneous test but a false negative results in a failure to treat a disease

$$\text{true positive rate (recall)} = \frac{TP}{\text{actual pos}} = \frac{TP}{TP + FN}$$

$$\text{false positive rate} = \frac{FP}{\text{actual neg}} = \frac{FP}{TN + FP}$$

Estimation of the error

How can we get an estimate of the accuracy of a learned model?

Natural performance measure for classification problems: *error rate*

- *Success*: instance's class is predicted correctly
- *Error*: instance's class is predicted incorrectly
- Error rate: proportion of errors made over the whole set of instances

Decision rule

$$\Lambda(x) \rightarrow \omega_m$$

$$\epsilon = \frac{\sum_{x_i \in \Omega} \Lambda(x_i) == Y_d(x_i)}{card(\Omega)}$$

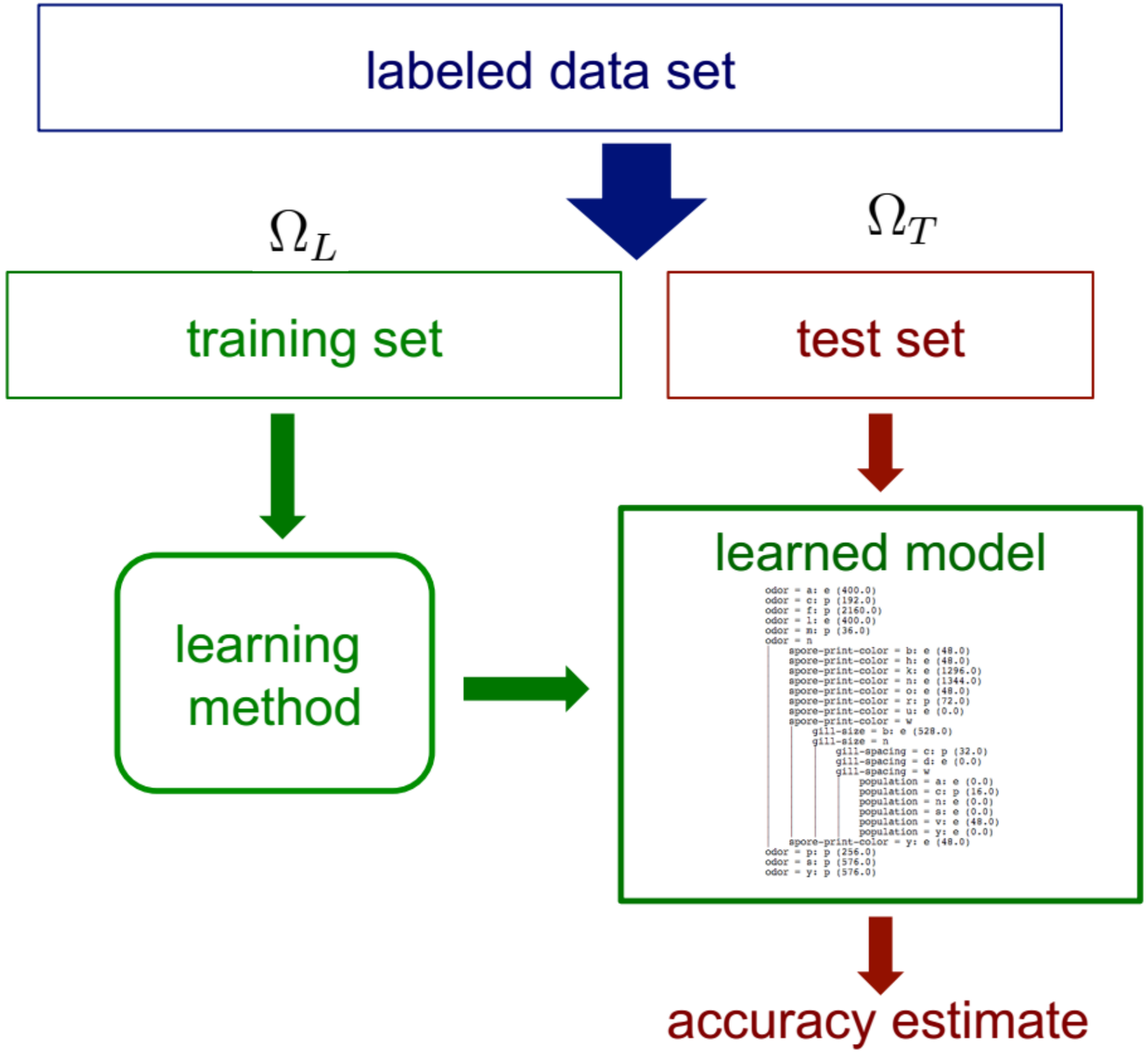
Estimation $\quad Bias(\tilde{\epsilon}) = \epsilon - E(\tilde{\epsilon}) \qquad Var(\tilde{\epsilon}) = E[\tilde{\epsilon} - E(\tilde{\epsilon})]^2$

Resubstitution error: error rate obtained from training data

Resubstitution error is (hopelessly) optimistic! (Bias is very important)

$$\frac{\sum_{i=1}^{N} \Lambda(x_i) == Y_d(x_i)}{N}$$

Unbiaised Estimation of the error



Test set: independent instances that have played no part in formation of classifier

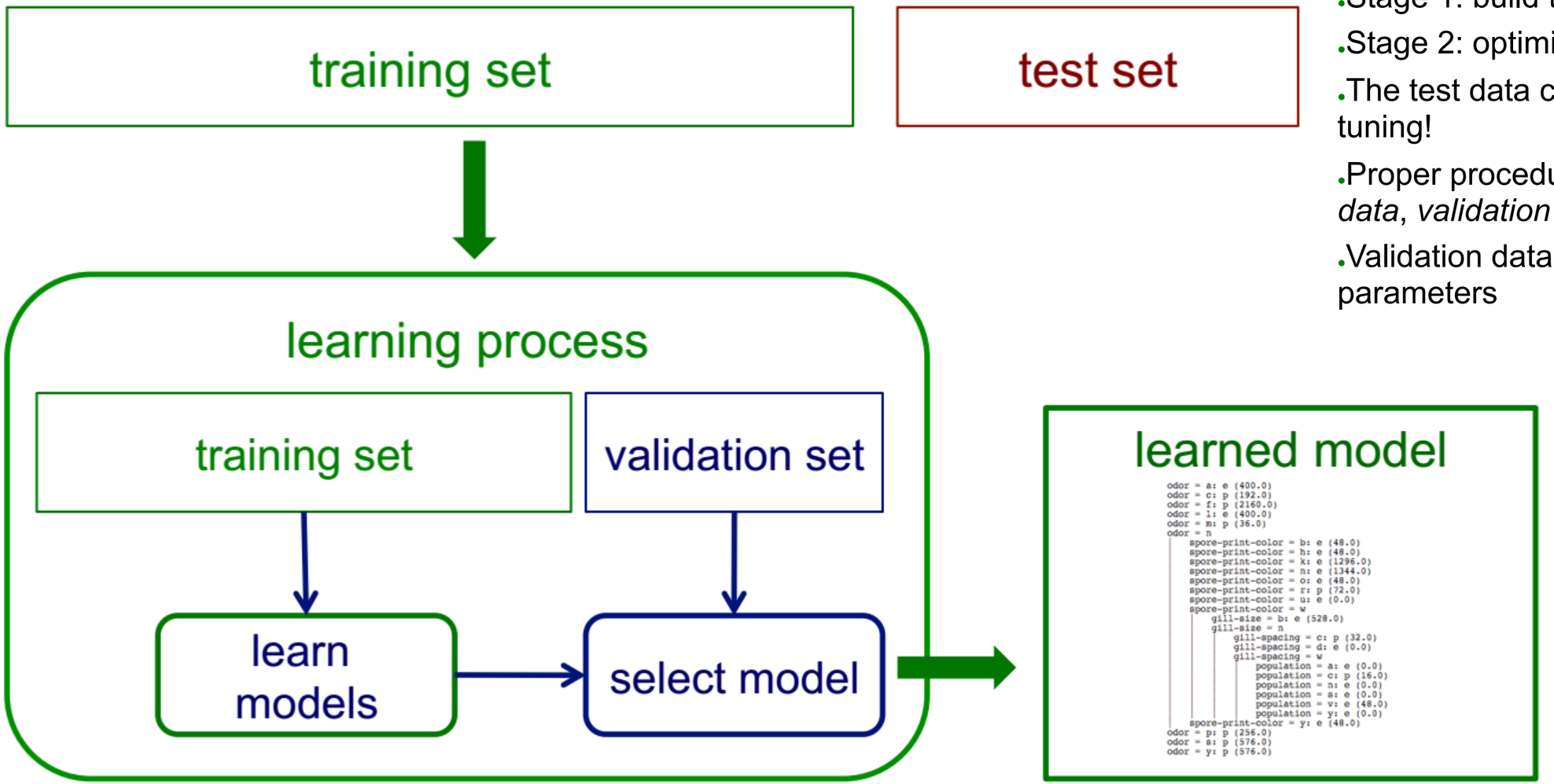
- Assumption: both training data and test data are representative samples of the underlying problem
- Test and training data may differ in nature

$$\frac{\sum_{x_i \in \Omega_T} \Lambda(x_i) == Y_d(x_i)}{card(\Omega_T)}$$

Bias is 0 but the variance is important (depends on *card* of the test data)

Validation (tuning) sets revisited

Suppose we want unbiased estimates of accuracy during the learning process (e.g. to choose the best K of KNN)?



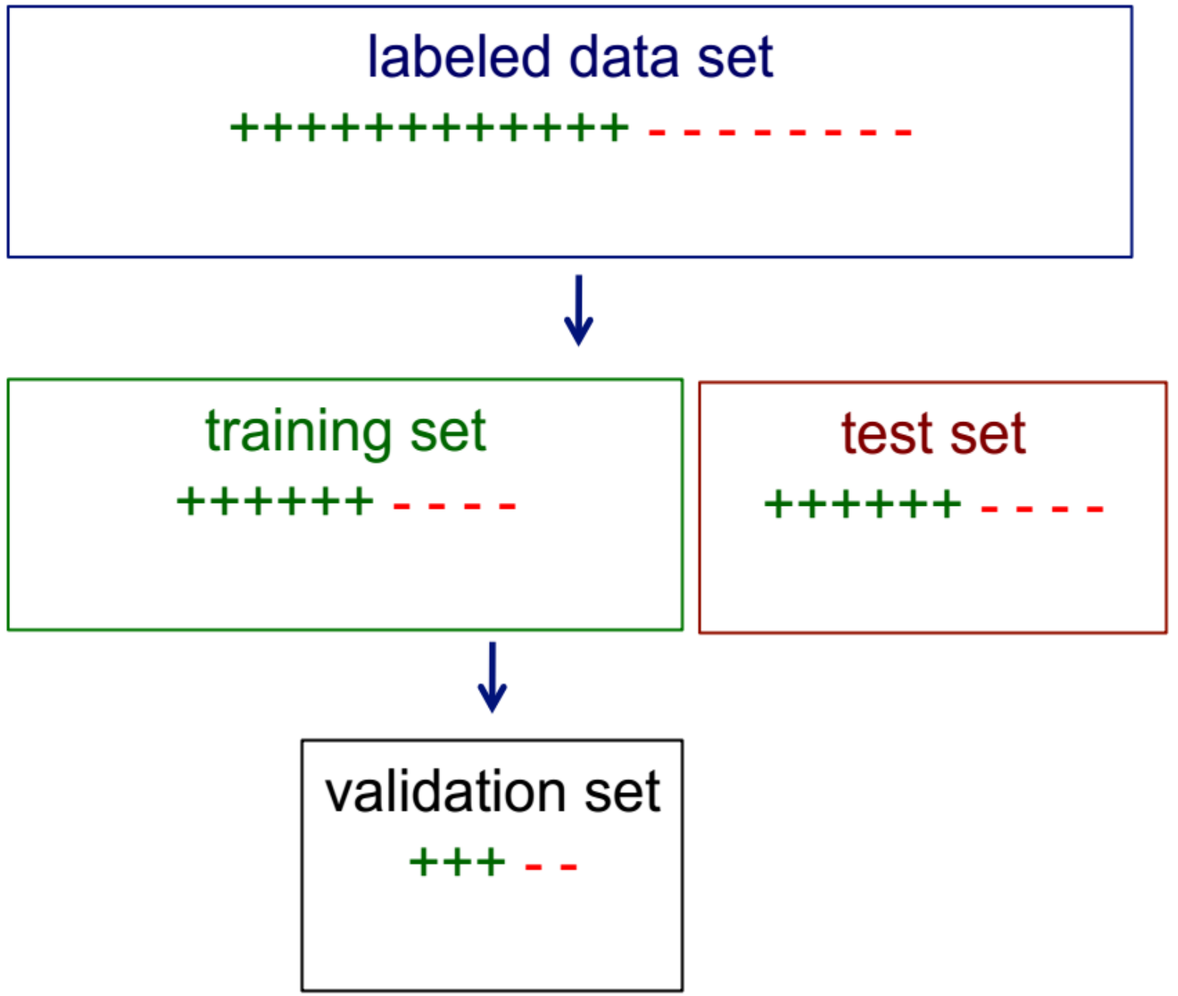
It is important that the test data is not used *in any way* to create the classifier

Some learning schemes operate in two stages:

- Stage 1: build the basic structure
- Stage 2: optimize parameter settings
- The test data can't be used for parameter tuning!
- Proper procedure uses *three* sets: *training data*, *validation data*, and *test data*
- Validation data is used to optimize parameters

Stratified sampling

When randomly selecting training or validation sets, we may want to ensure that class proportions are maintained in each selected set



Limitations of using a single training/test partition

We may not have enough data to make sufficiently large training and test sets

- a larger test set gives us more reliable estimate of accuracy (i.e. a lower variance estimate)

- but… a larger training set will be more representative of how much data we actually have for the learning process

A single training set doesn't tell us how sensitive accuracy is to a particular training sample

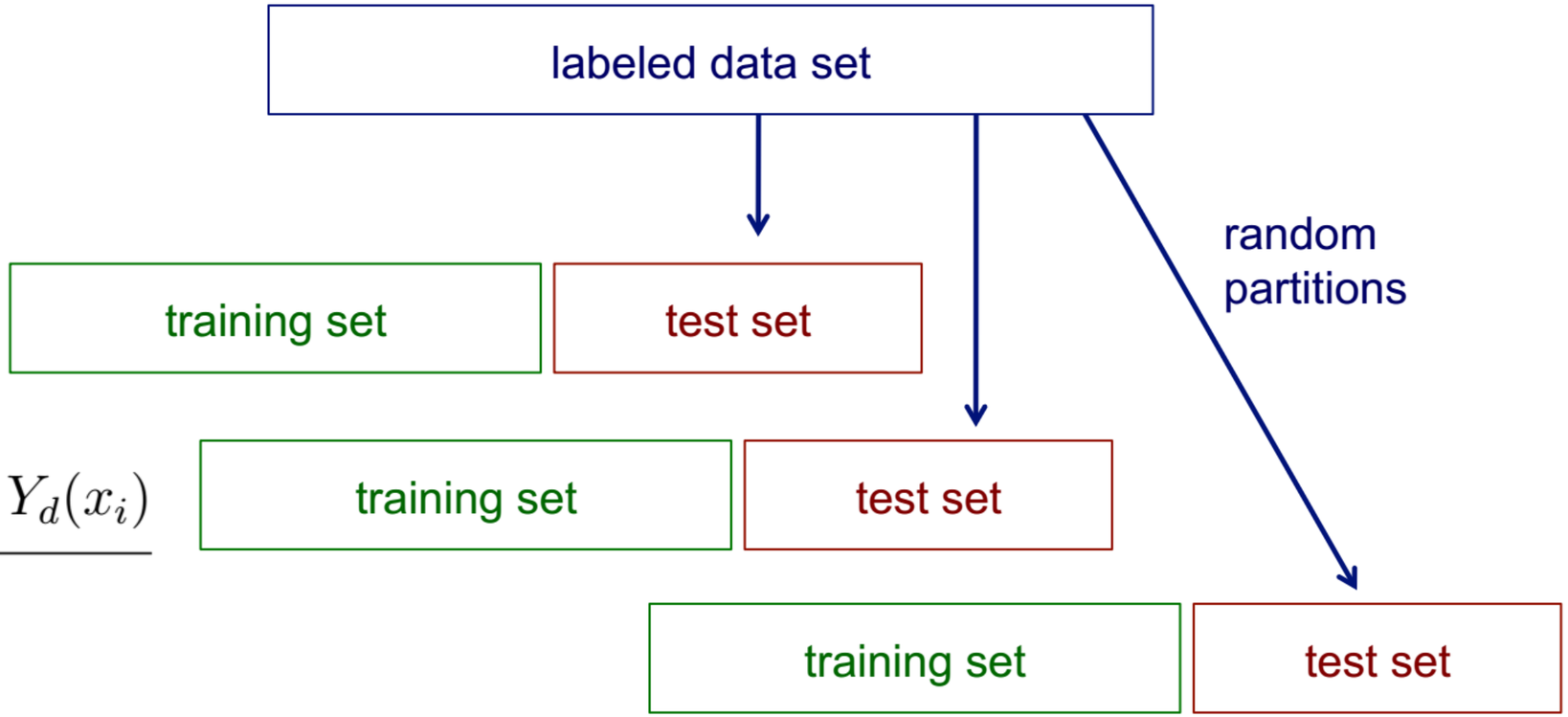
Repeated holdout method

Holdout estimate can be made more reliable by repeating the process with different subsamples

- In each iteration, a certain proportion is randomly selected for training (possibly with stratificiation)
- The error rates on the different iterations are averaged to yield an overall error rate

This is called the *repeated holdout* method

Still not optimum: the different test sets overlap



$$\tilde{\epsilon} = \frac{1}{B}\sum_{b=1}^{b=B}\frac{\sum_{x_i \in \Omega_{T^b}} \Lambda_{\Omega_{L^b}}(x_i) == Y_d(x_i)}{card(\Omega_{T^b})}$$

Cross-validation

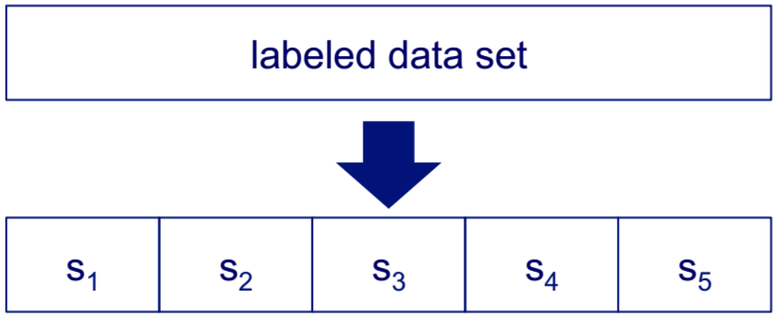
Cross-validation avoids overlapping test sets

- First step: split data into *B* subsets of equal size
- Second step: use each subset in turn for Testing, the remainder for Learning (training)

Called *k-fold cross-validation*

- Often the subsets are stratified before the cross-validation is performed
- The error estimates are averaged to yield an overall error estimate

partition data into *n* subsamples



iteratively leave one subsample out for the test set, train on the rest

iteration	train on	test on
1	s_2 s_3 s_4 s_5	s_1
2	s_1 s_3 s_4 s_5	s_2
3	s_1 s_2 s_4 s_5	s_3
4	s_1 s_2 s_3 s_5	s_4
5	s_1 s_2 s_3 s_4	s_5

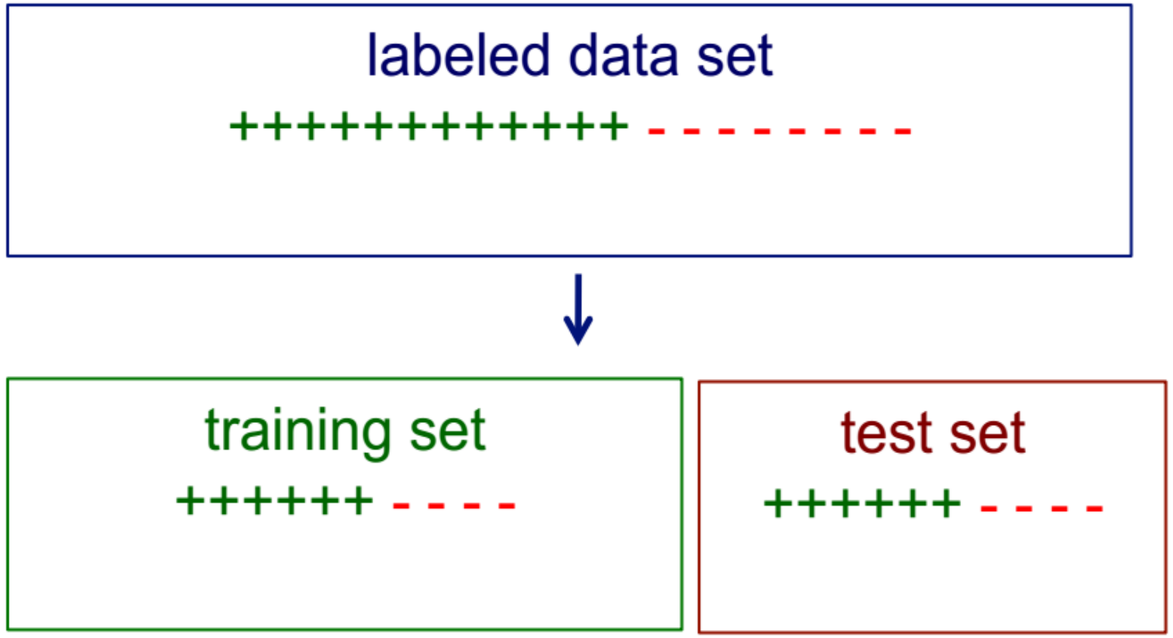
sklearn

Plusieurs fonctionnalités de pré-traitement des données sont disponibles dans le package sklearn.preprocessing :

- outils pour centrer le nuage des observations,
- pour centrer et réduire les variables,
- pour (re)coder des variables (par exemple, codage d'une variable nominale à m modalités par m variables binaires
- …

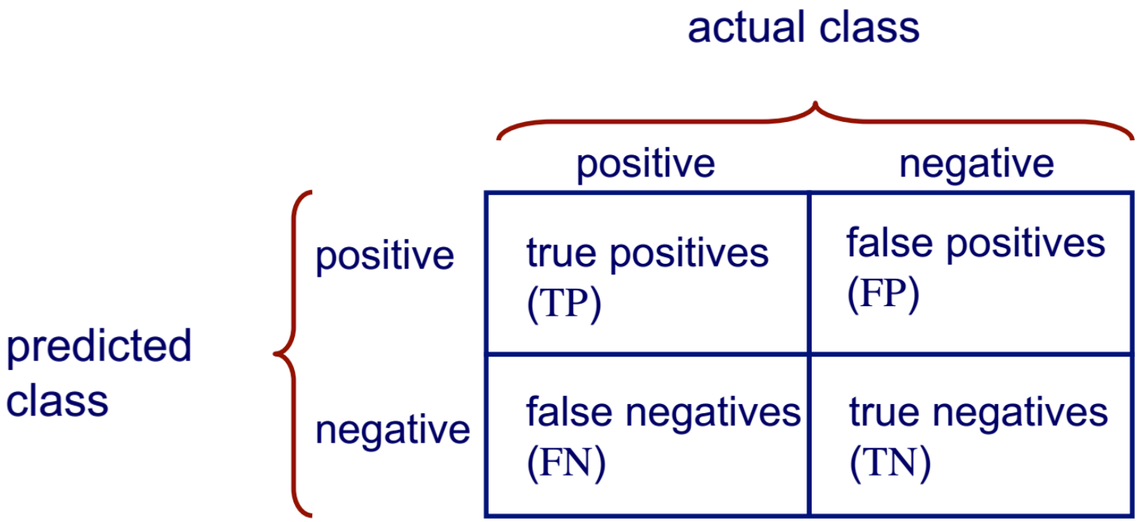
```
from sklearn import preprocessing
dataCR = preprocessing.scale(data)
```

Dans Scikit-learn, un estimator est un objet qui implémente la méthode fit(X, y) pour estimer un modèle et predict(T) afin d'utiliser ce modèle pour prendre des décisions (faire des prédictions).




```
from sklearn.model_selection import train_test_split

#split dataset into train and test data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y)
```

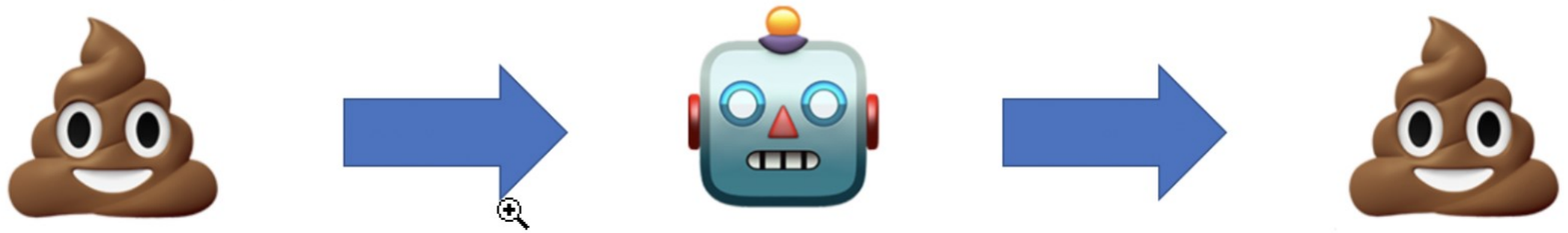
The *confusion matrix*:




```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

Deep Neural Network

- Training the network, quality of data is important

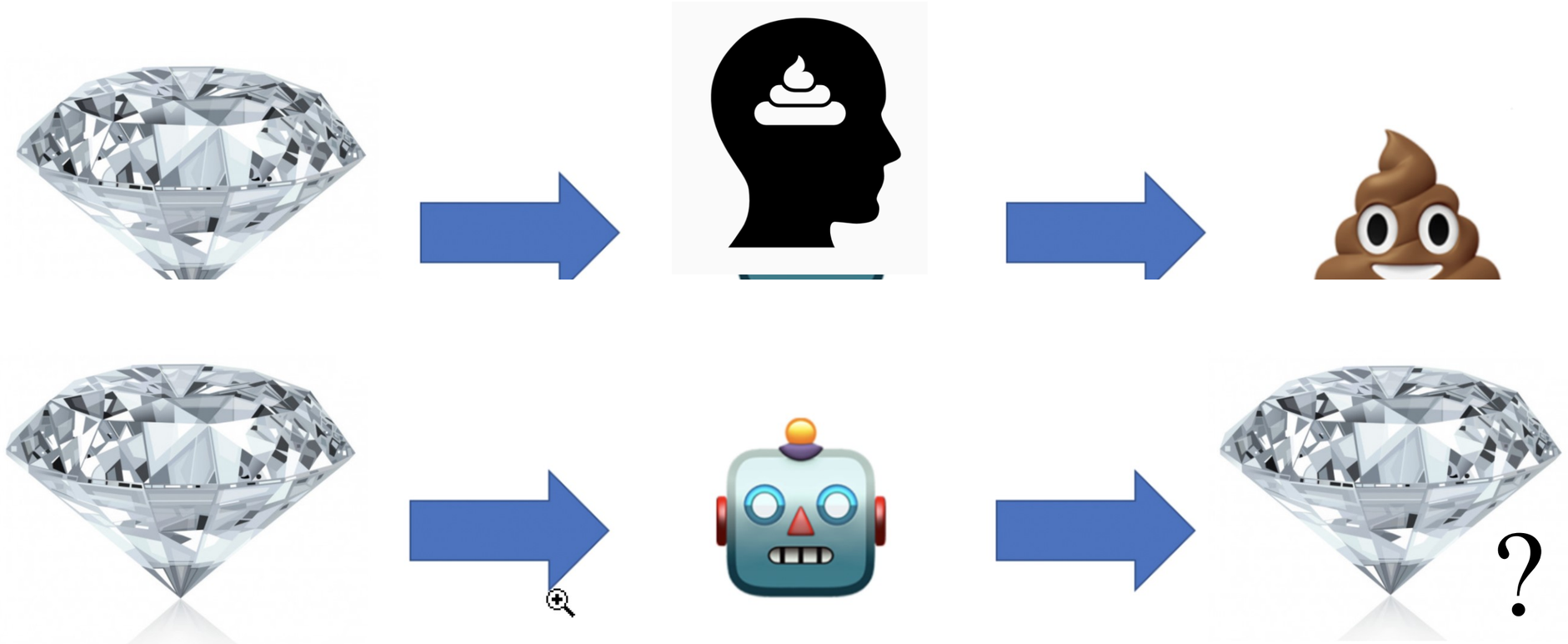


A phrase to express the idea that in computing and other fields, incorrect or poor-quality (data) input will produce faulty output.

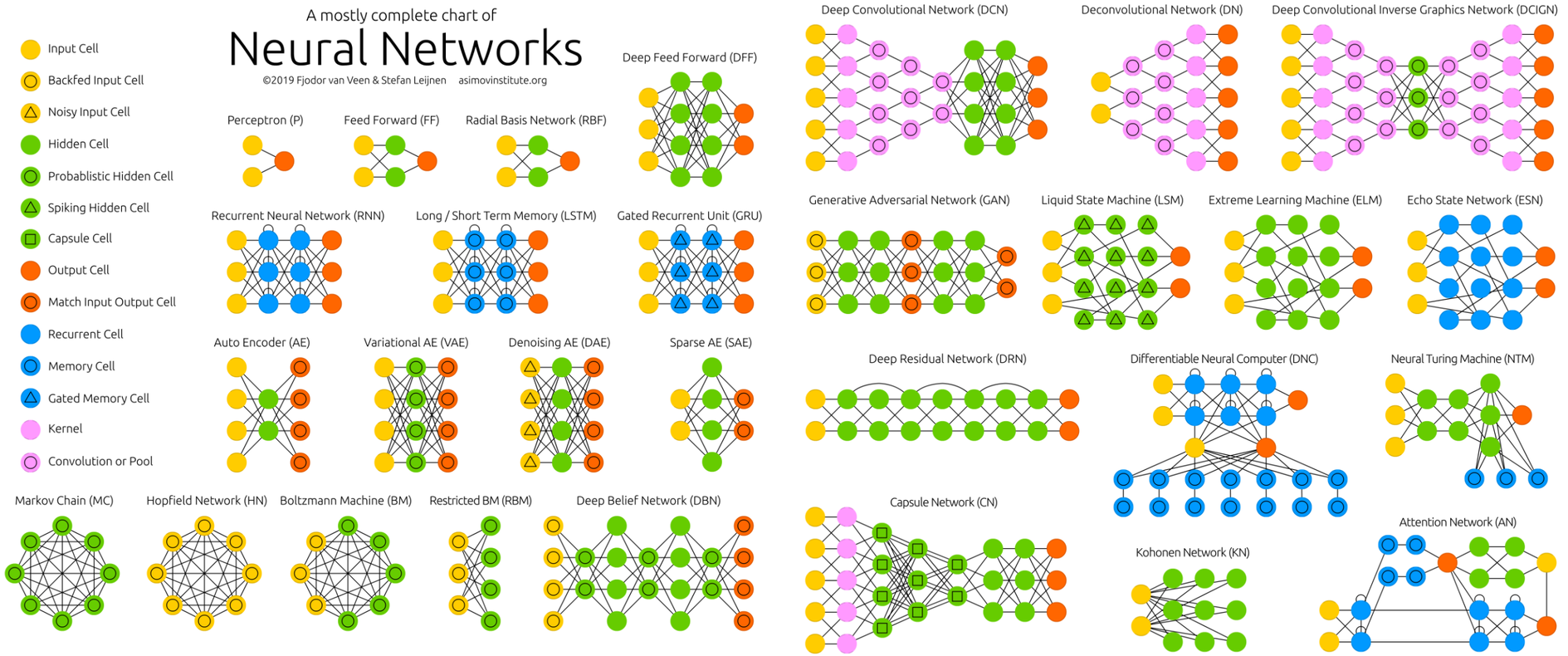
Definitions from Oxford Languages

Deep Neural Network

- Well chosen model is also important



Neural Network Zoo



https://www.asimovinstitute.org/neural-network-zoo/