Des neurones pour la modélisation et la décision

(en 30 minutes …)

philippe.carre@univ-poitiers.fr

institut de recherche

ai4 industry

Université de Poitiers

FACULTÉ DES SCIENCES FONDAMENTALES ET APPLIQUÉES
UNIVERSITÉ DE POITIERS

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

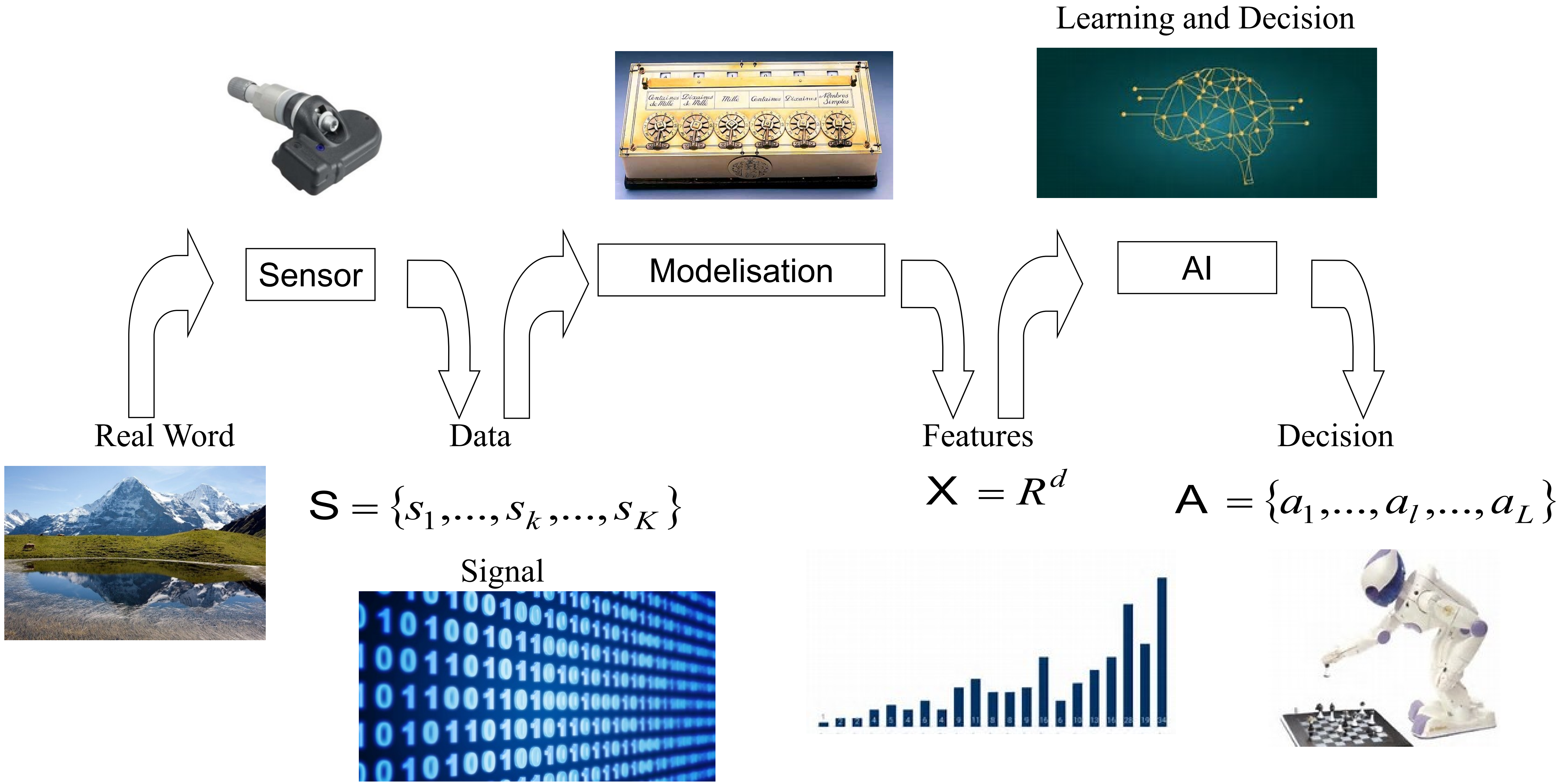
Machine learning
$$C: R^d \rightarrow \{1,\ldots,l,\ldots,L\}$$
$$x \mapsto C(x)$$

The aim:

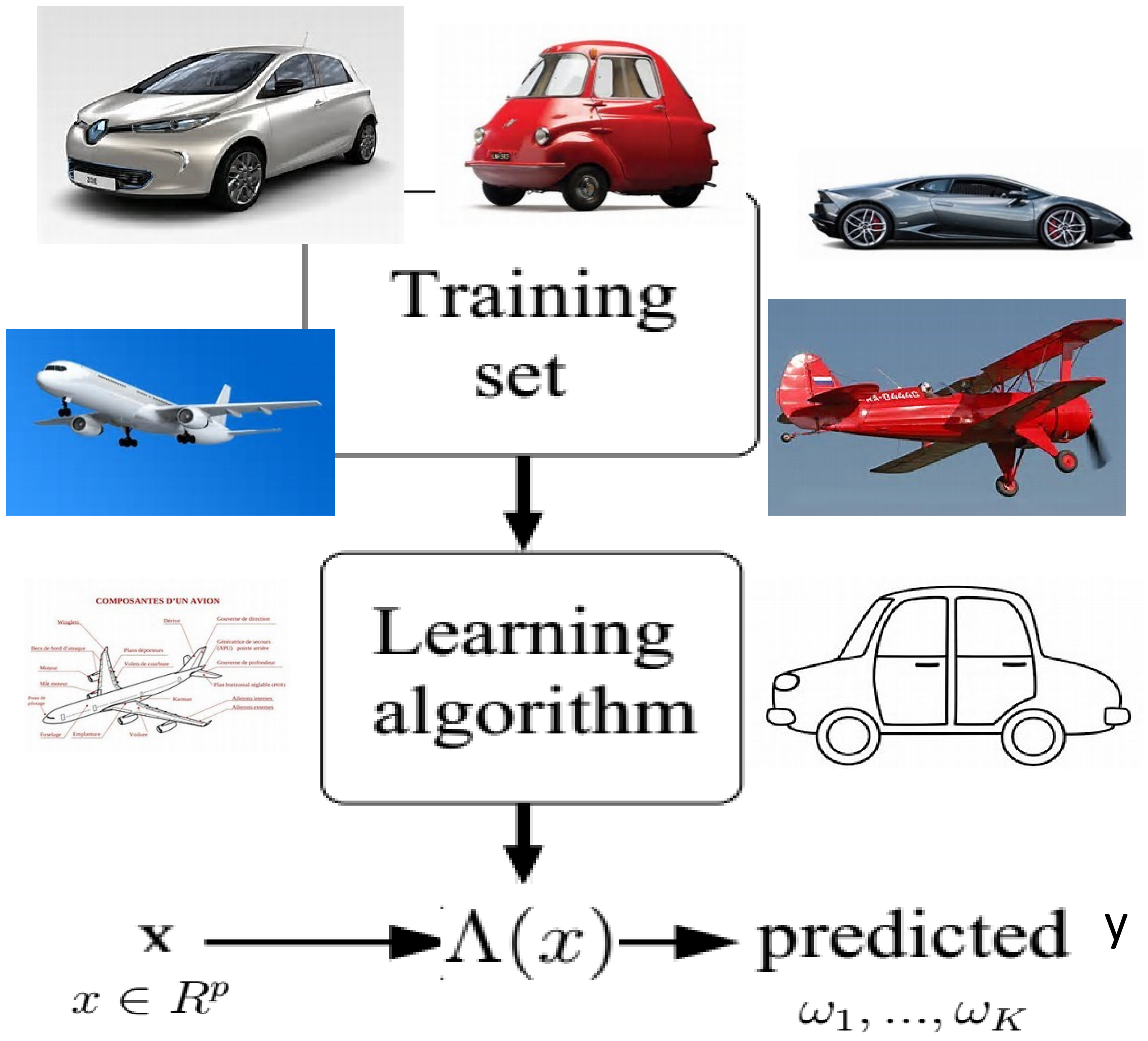
$$\forall x \in R^d, C(x) = \text{"the true decision"}$$

Difficulties: within-class variations

Discrimination Process



Learning

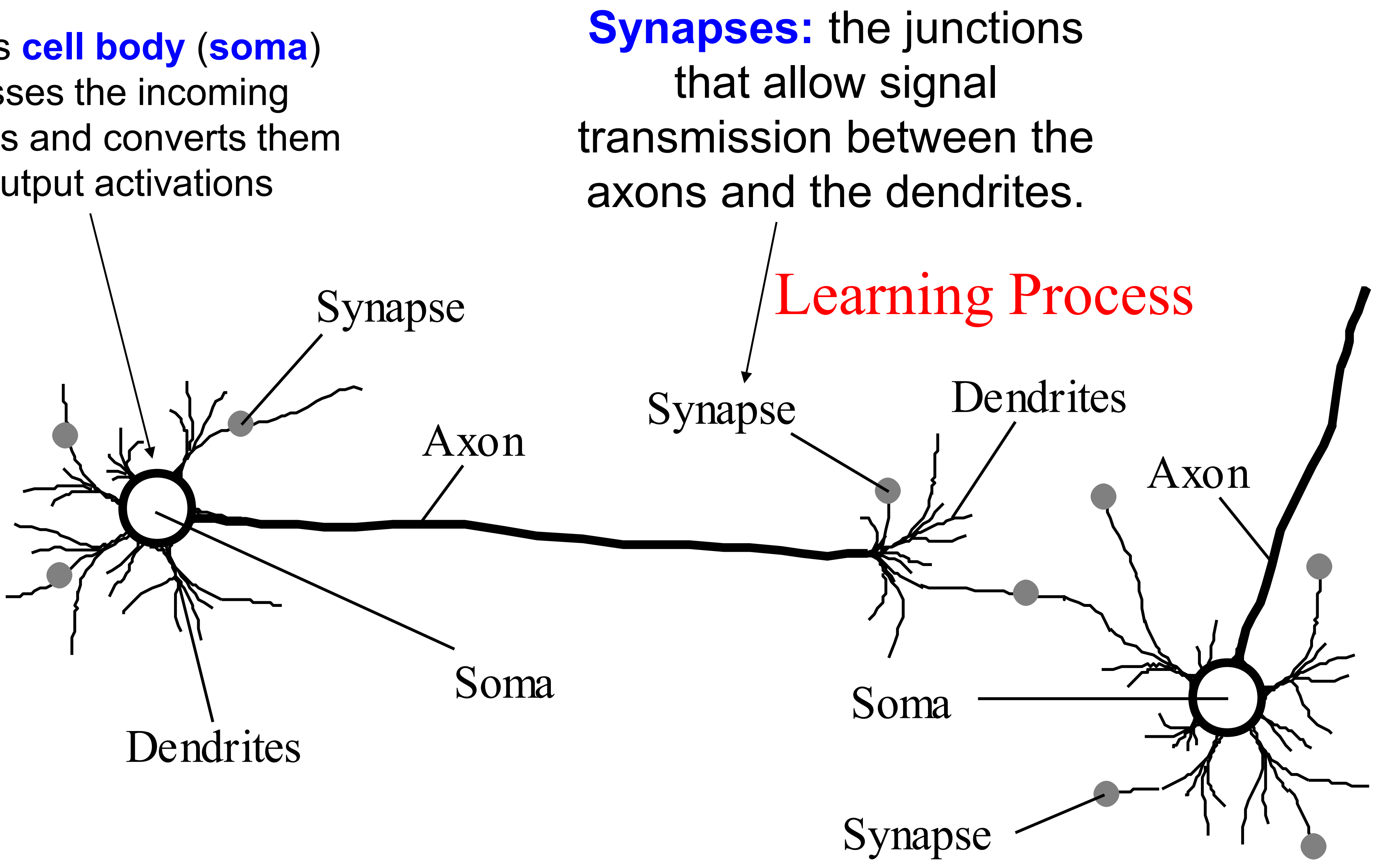


Our goal is, given a training set, to learn a function so that Λ(x) is a "good" predictor for the corresponding value of y

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

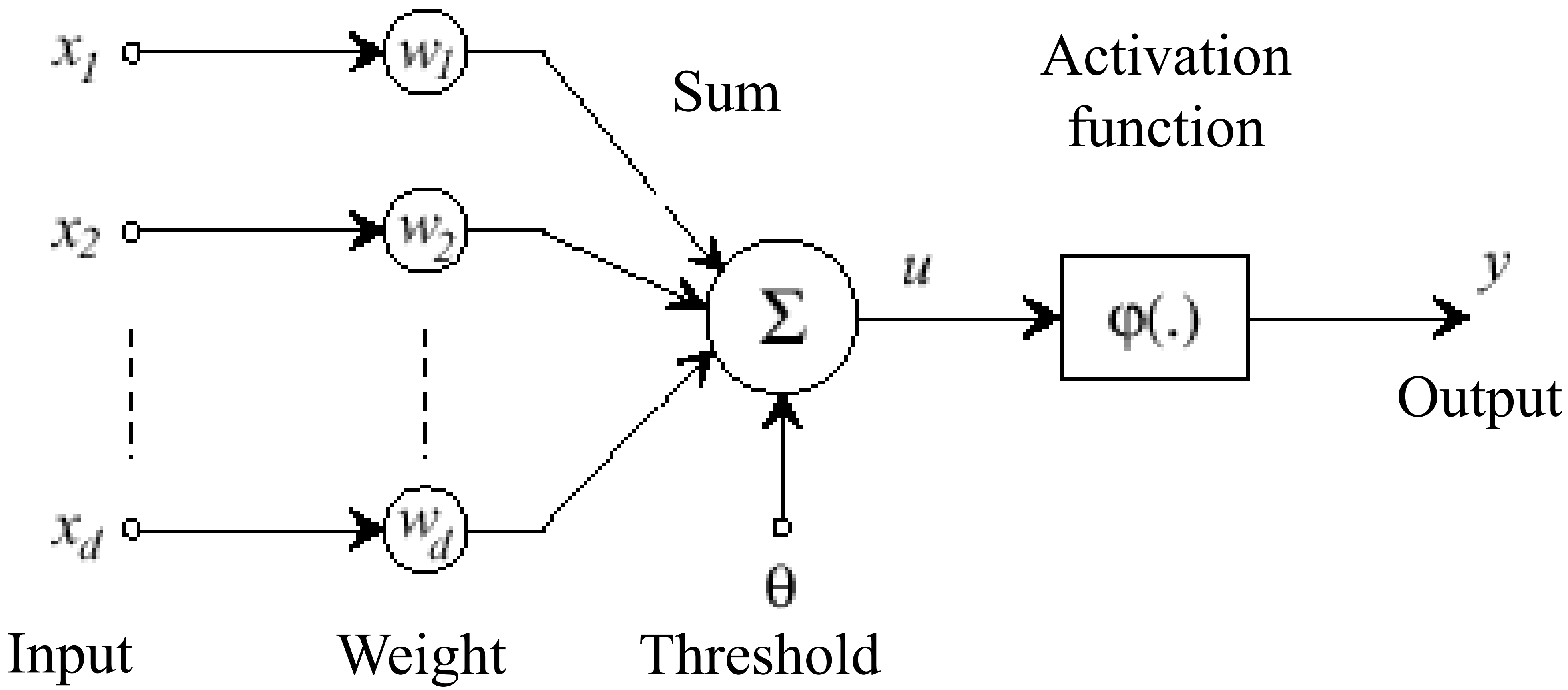
Neuronal Network Approach

A highly complex, non-linear and parallel information-processing system



Formal neuron

Mc Culloch et Pitts 1943



Threshold

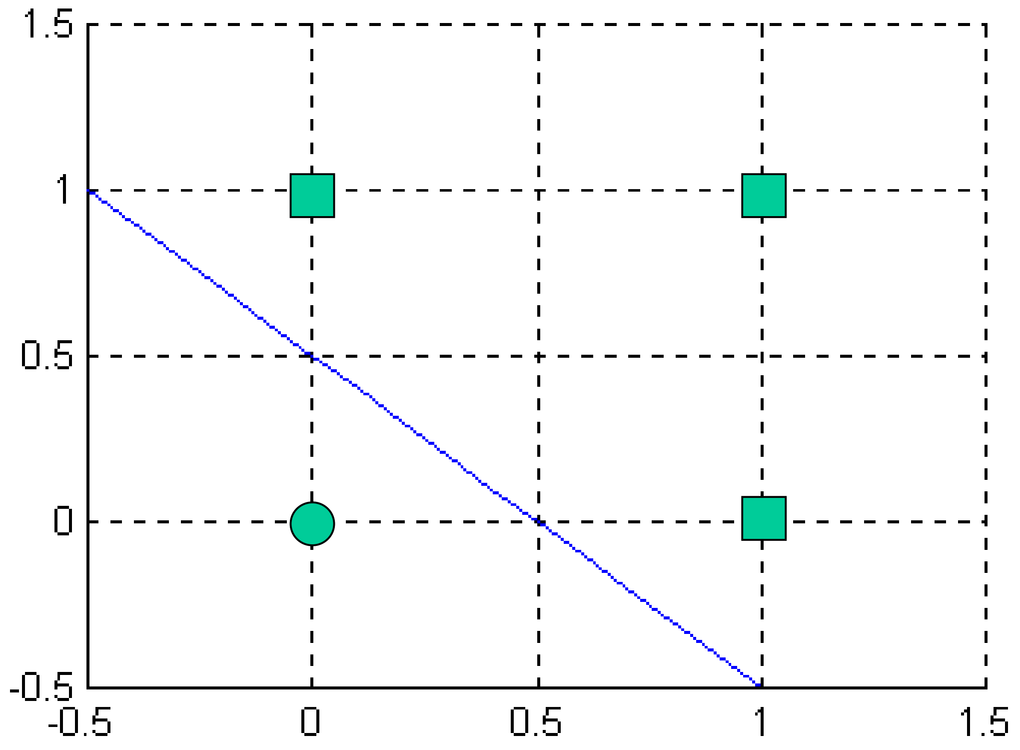
$$
\begin{aligned}
u &= \sum_{j=1}^{d} w_j x_j + \theta = \sum_{j=0}^{d} w_j x_j \text{ avec } w_0 = \theta \text{ et } x_0 = 1 \\
&= w^T x
\end{aligned}
$$

A non-linear activation function

$$y = \varphi(u) = \varphi(w^T x)$$

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

Perceptron: decision rule



$$\mathbf{w}^T\mathbf{u} \geq 0 \text{ for every input vector } \mathbf{u} \text{ belonging to class } C_1$$
$$\mathbf{w}^T\mathbf{u} < 0 \text{ for every input vector } \mathbf{u} \text{ belonging to class } C_2$$

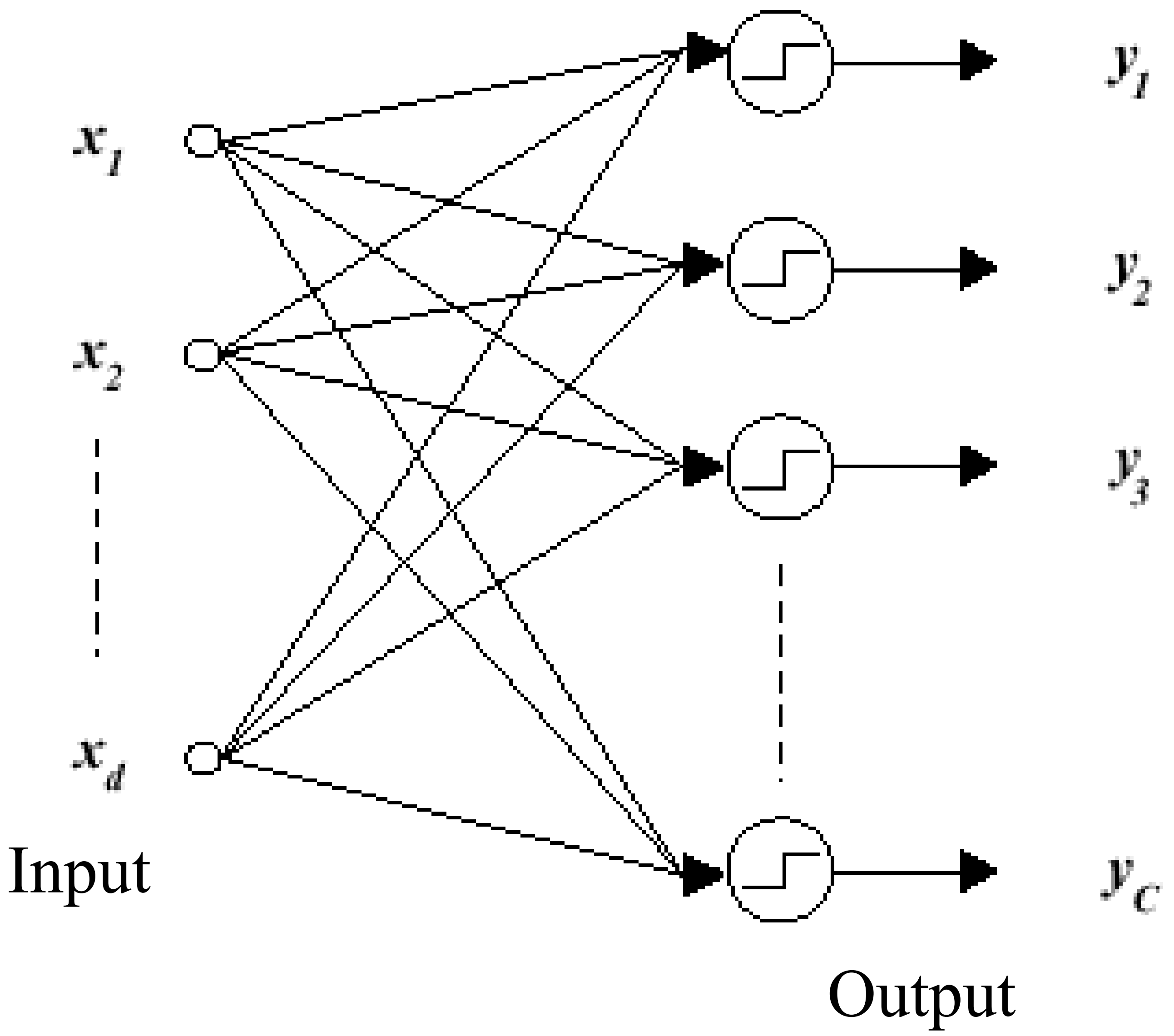
$$x \longrightarrow \omega_0 \text{ ssi } \underbrace{w^T x}_{\text{Linear decision rule}} \geq 0\,;\, x \longrightarrow \omega_1 \text{ otherwise}$$

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 cluster >2

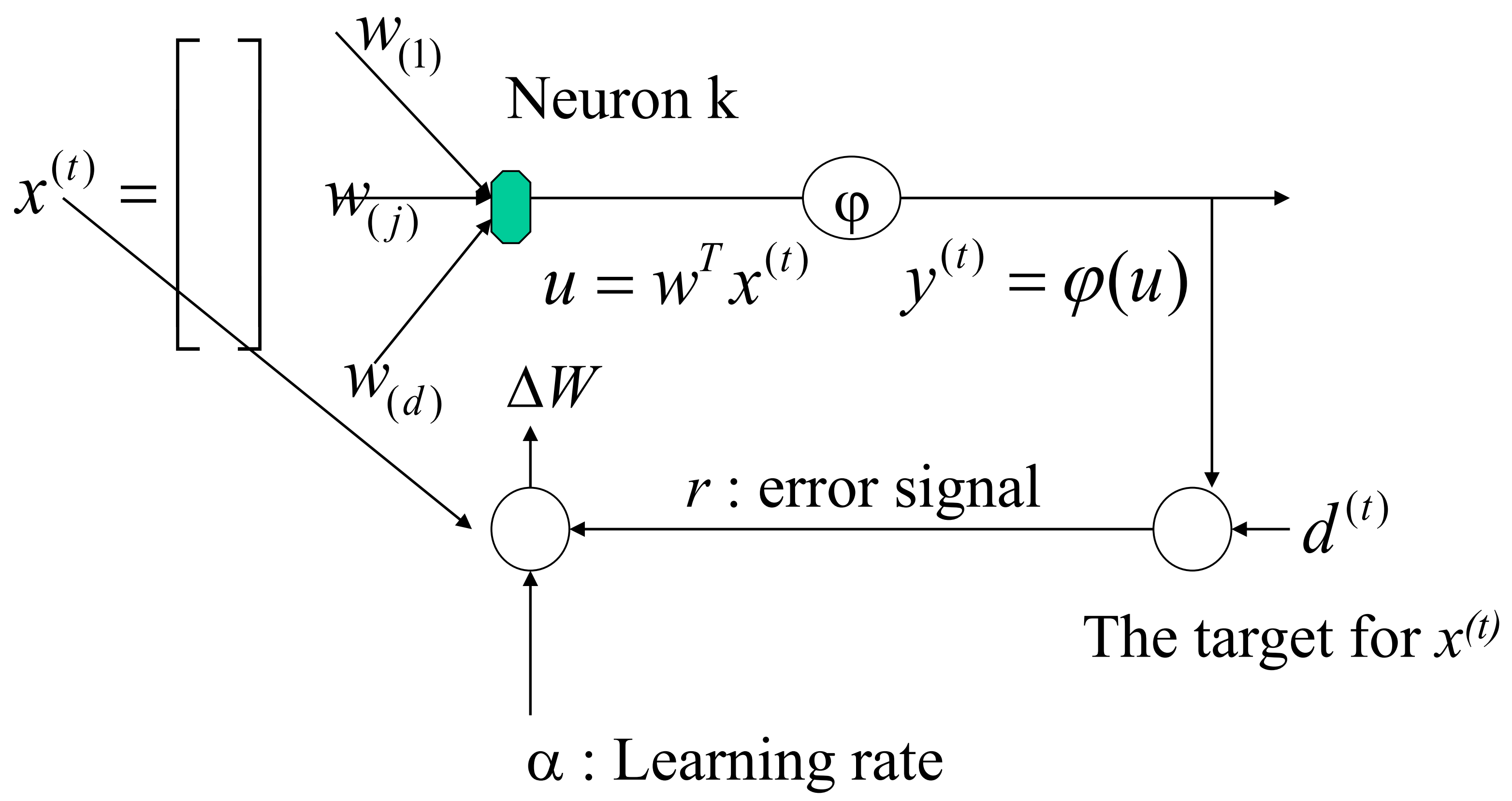


Cluster 1: yes or not

Cluster 2: yes or not

Cluster C: yes or not

Training the neural Network



Generate a training pair or pattern:
- an input ***x*** = *[x1 x2 … xn]*
- 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:

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

Repeat until "convergence"

A **Cost Function** to quantify this difference

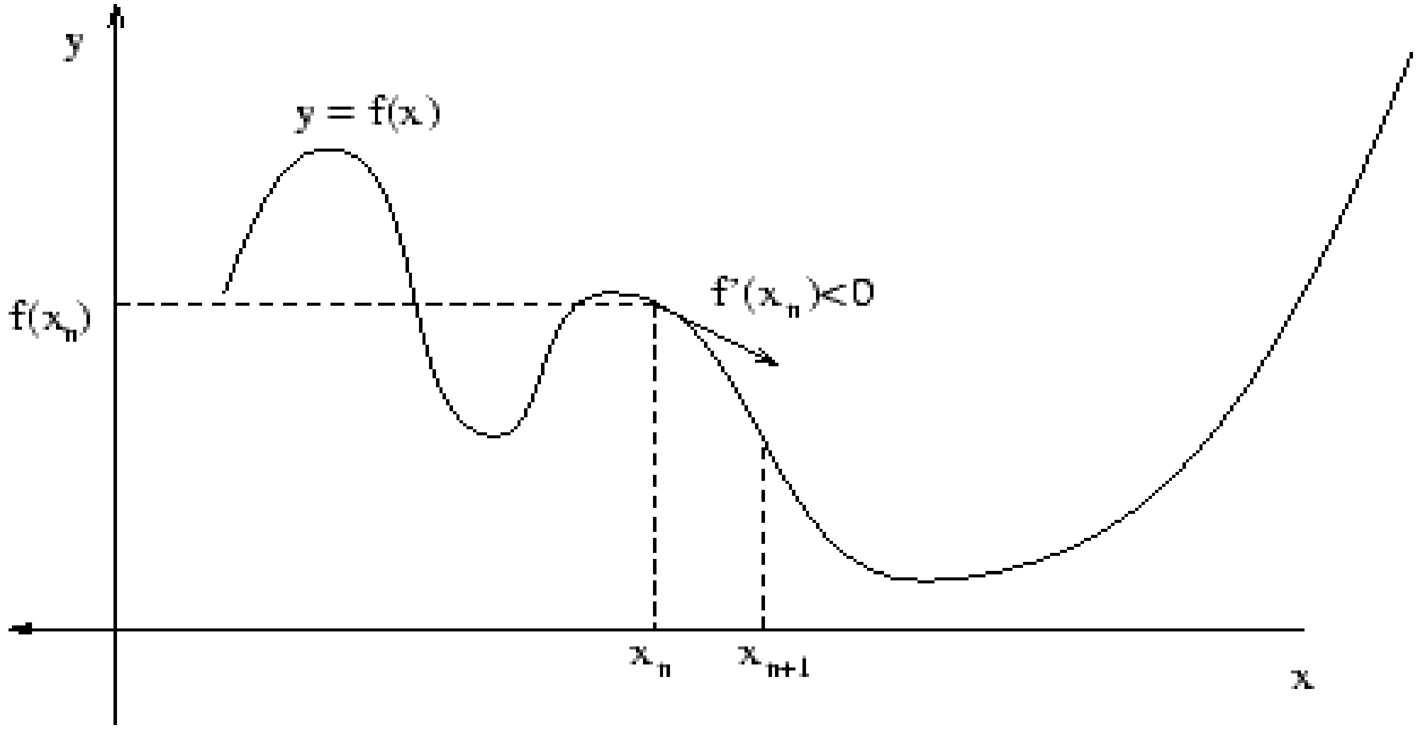
RMS cost function

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

*W** such that *E* minimun

Training the neural Network: Gradient descent

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



Initialize the initial position x0 at random

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

Repeat until convergence

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

Error defined for one training data samples

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

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

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

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

For each weight

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

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

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

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

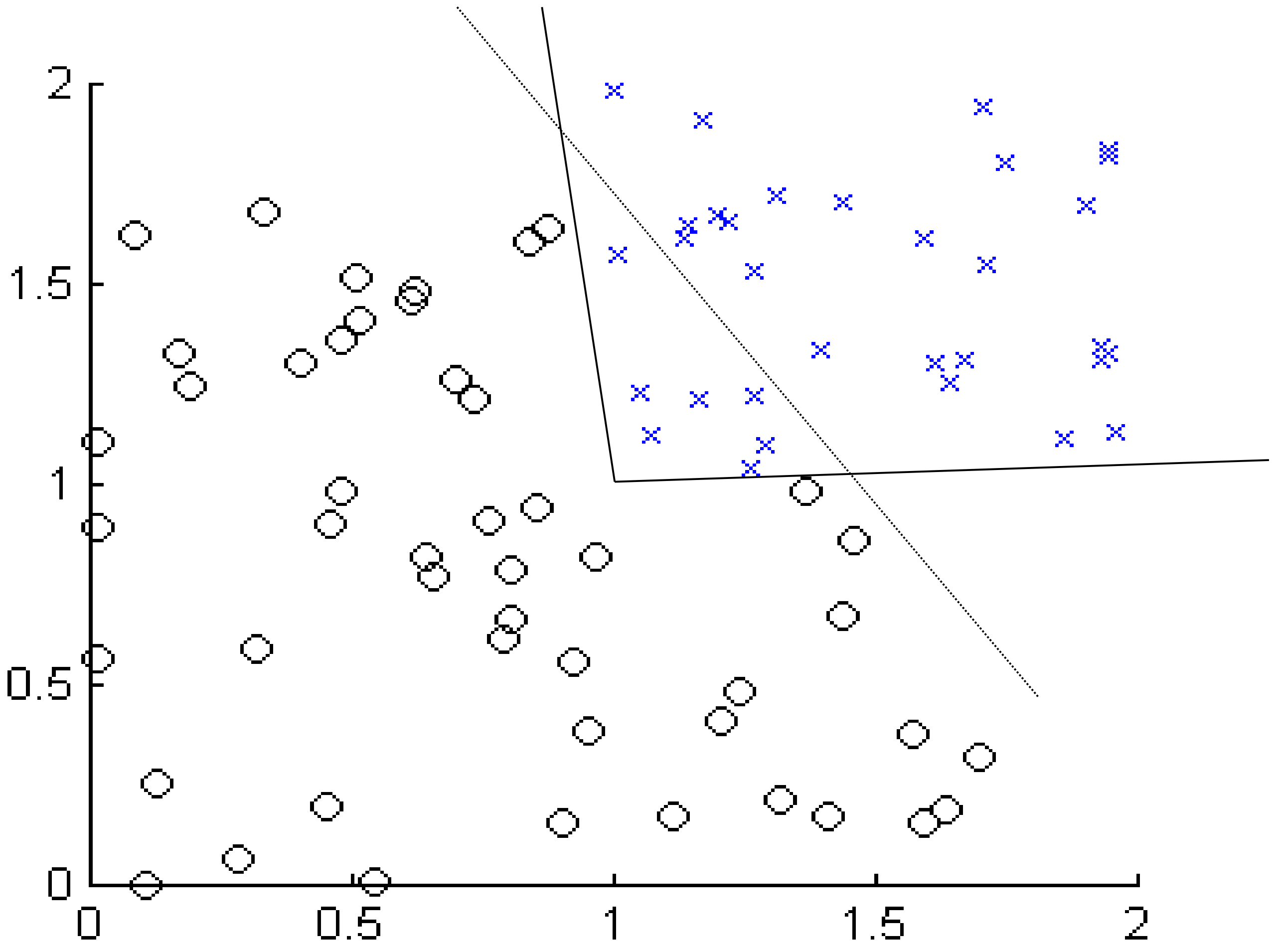
$$\Delta w_j = \sum_{t=1}^{n} \Delta w_j^{(t)}$$

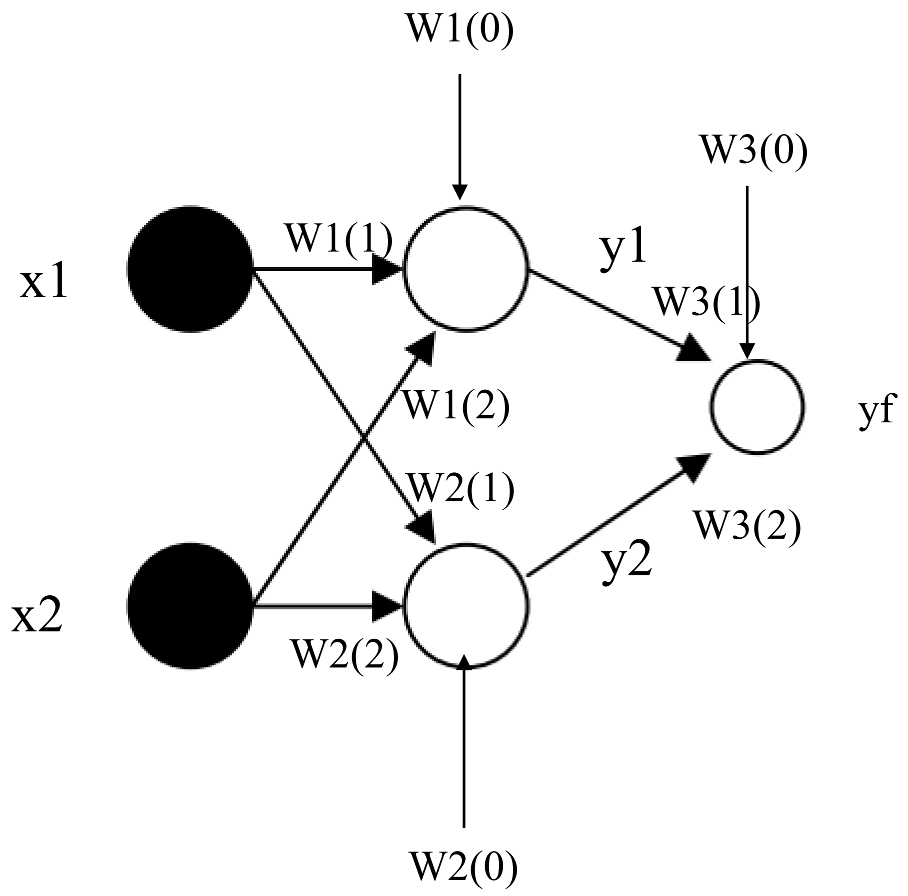
Epoch: the number of times the model is exposed to the training set

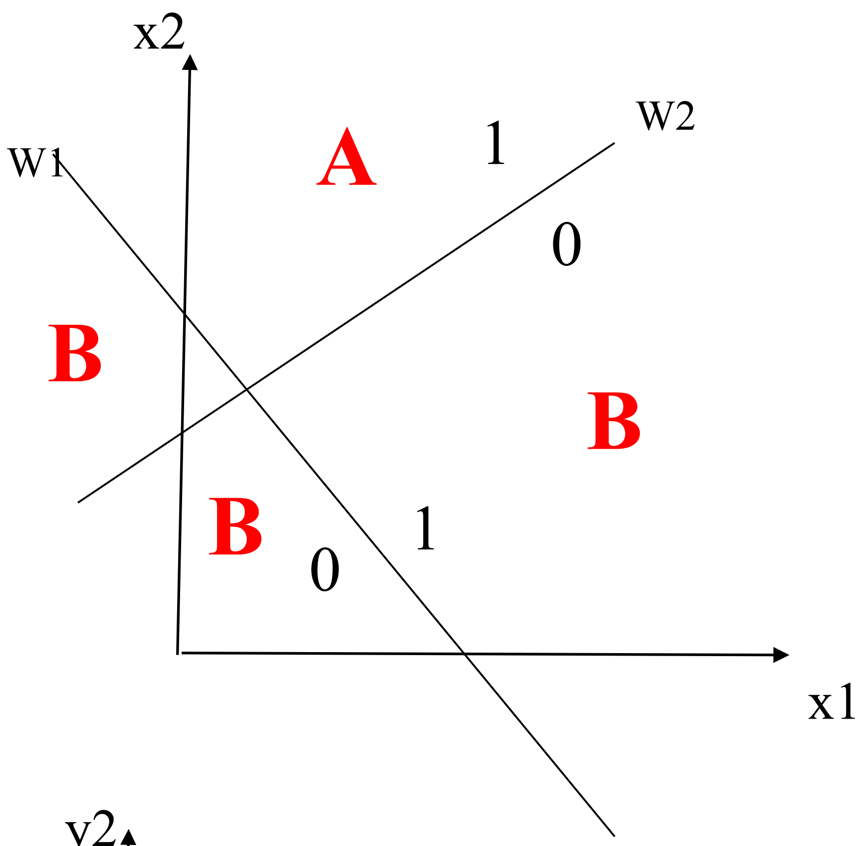
Batch_size: this is the number of training instances observed before the optimizer performs a weight update

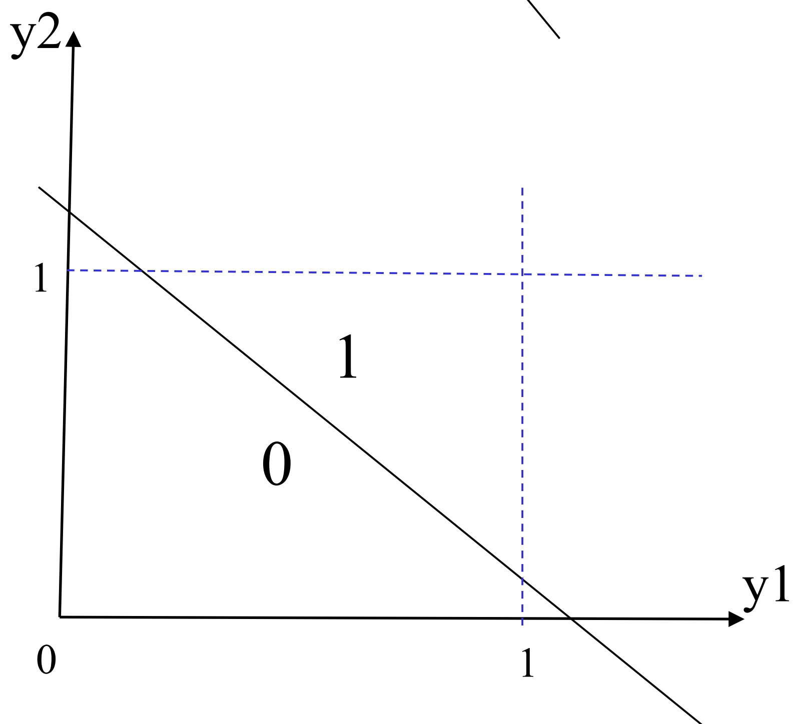
Multilayer neural network: why ?

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







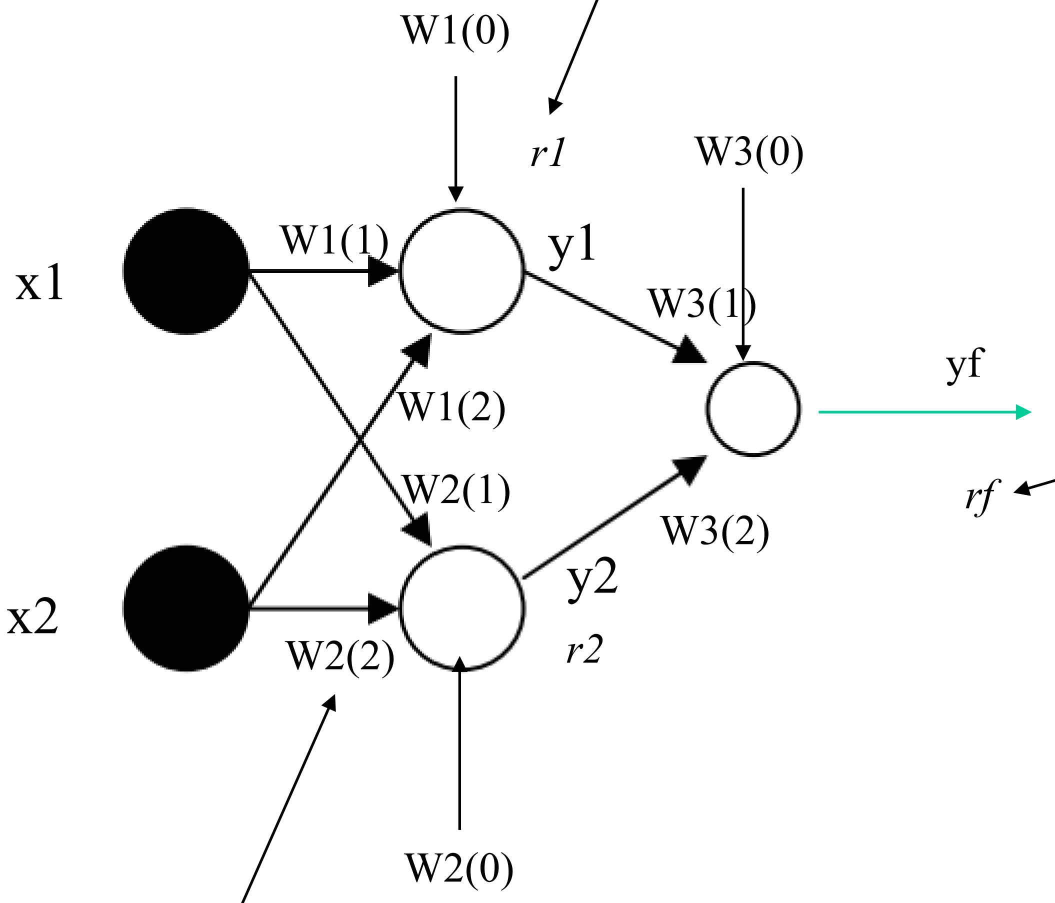


The propagation pass begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the output signal for each output neuron

Training Multilayer neural network: Backpropagation

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

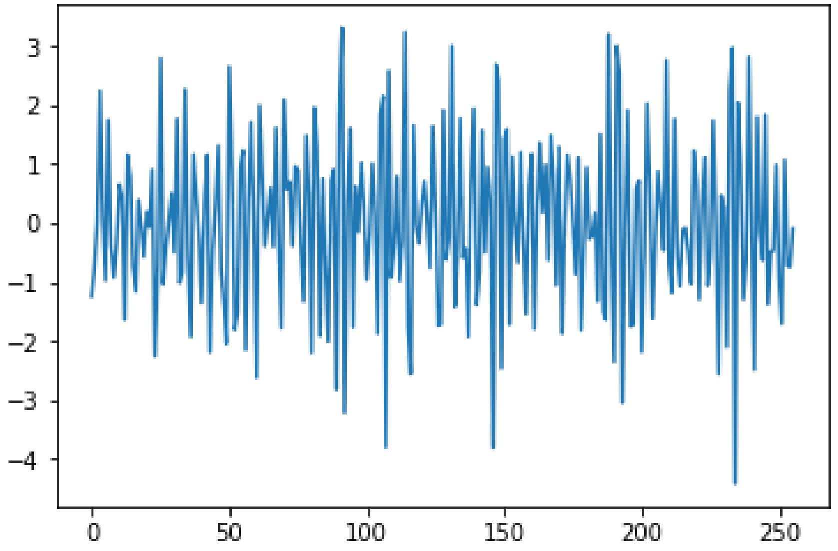
$$w^{n+1} = w^n + \alpha r x$$

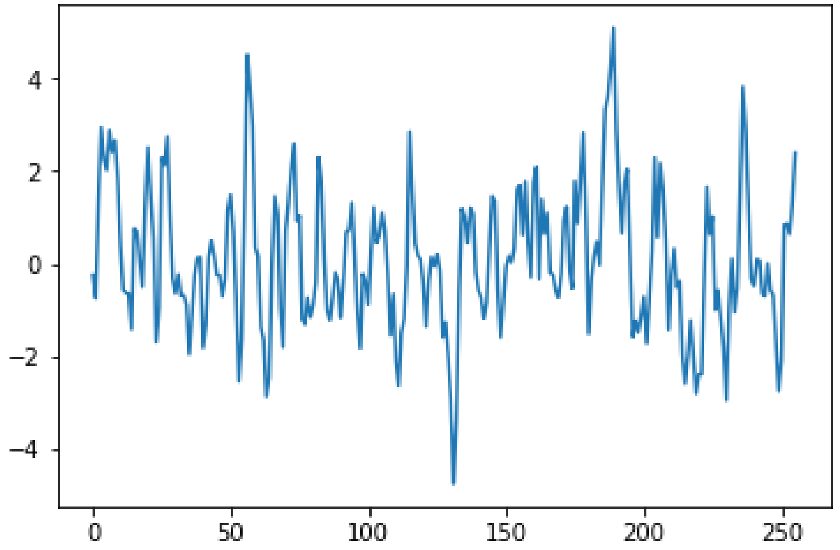


$$r_f = \frac{\partial E}{\partial v_f} = -(y_d - y_f)\varphi'(v_f)$$

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

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)
```

256

2000 DATA

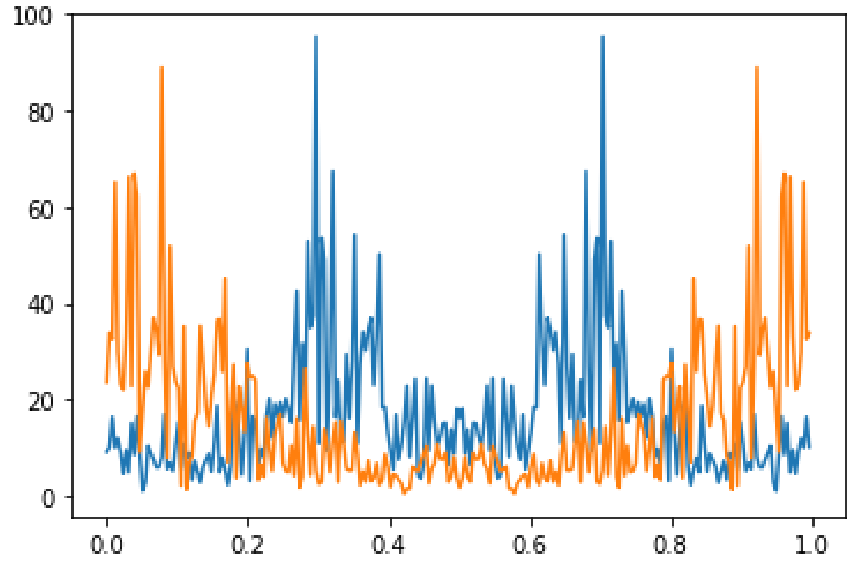
1 LABEL

Features : Energy for different intervals in the frequency domain

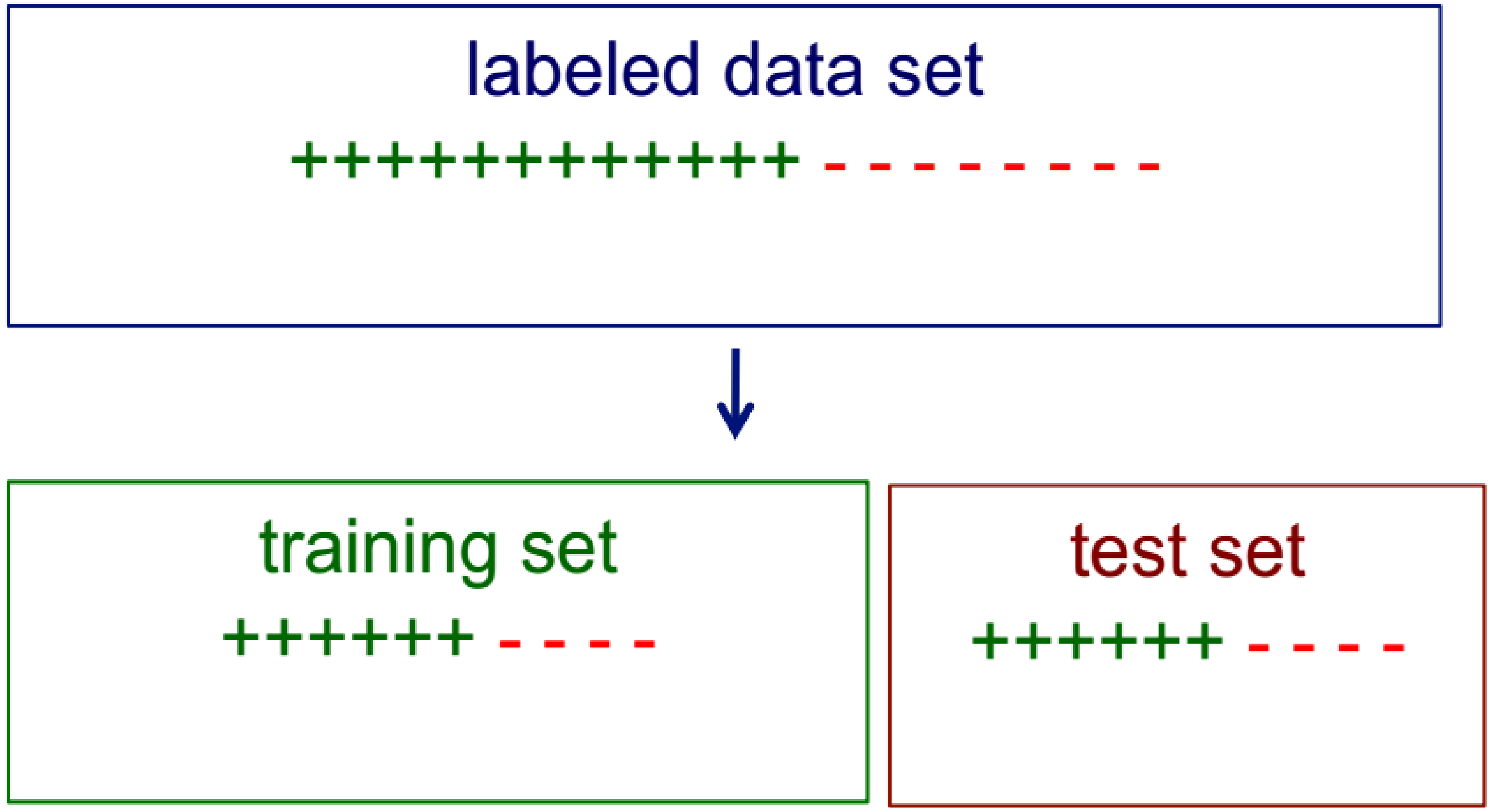
```
for k in range(0,Nbre_indivu*2):
    spec = abs(fft(Data[k,:]))**2
    for kk in range(0,8):
        sslg=int(lg/(8*2))
        Features[k,kk]=np.sum(spec[kk*sslg:(kk+1)*sslg])
```

8

2000 Features



Unbiaised Estimation of the error

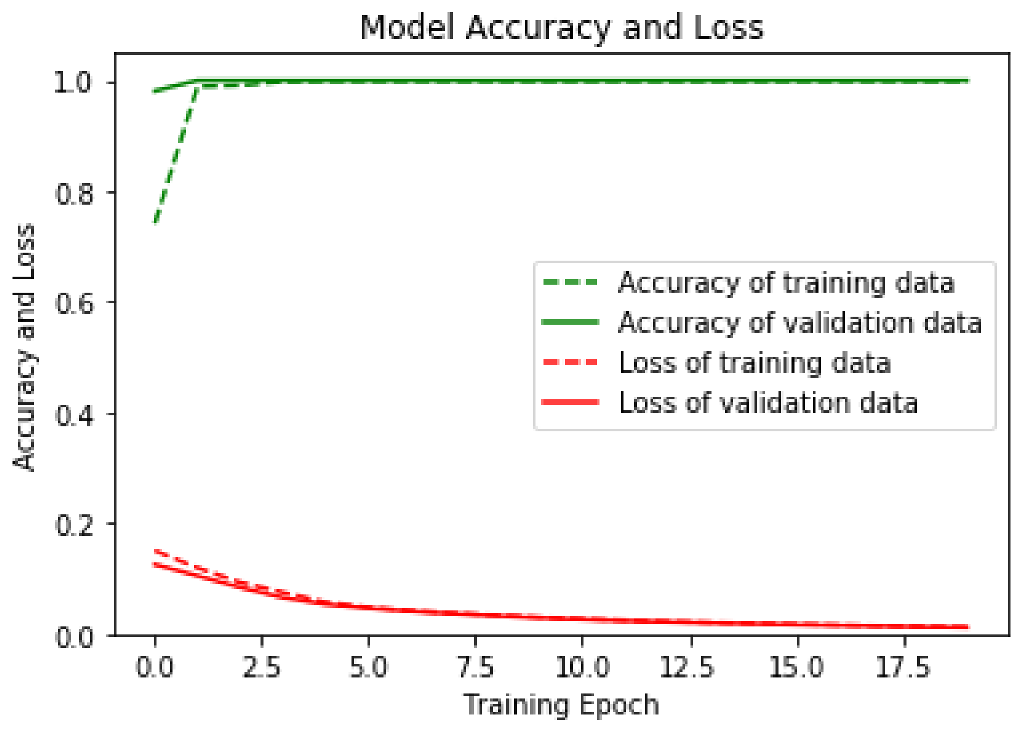



```
Label=np.concatenate((np.zeros(1000),np.ones(1000)))

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(Features,Label, test_size=0.2, random_state = 42,stratify = Label)
```

Learning and Test

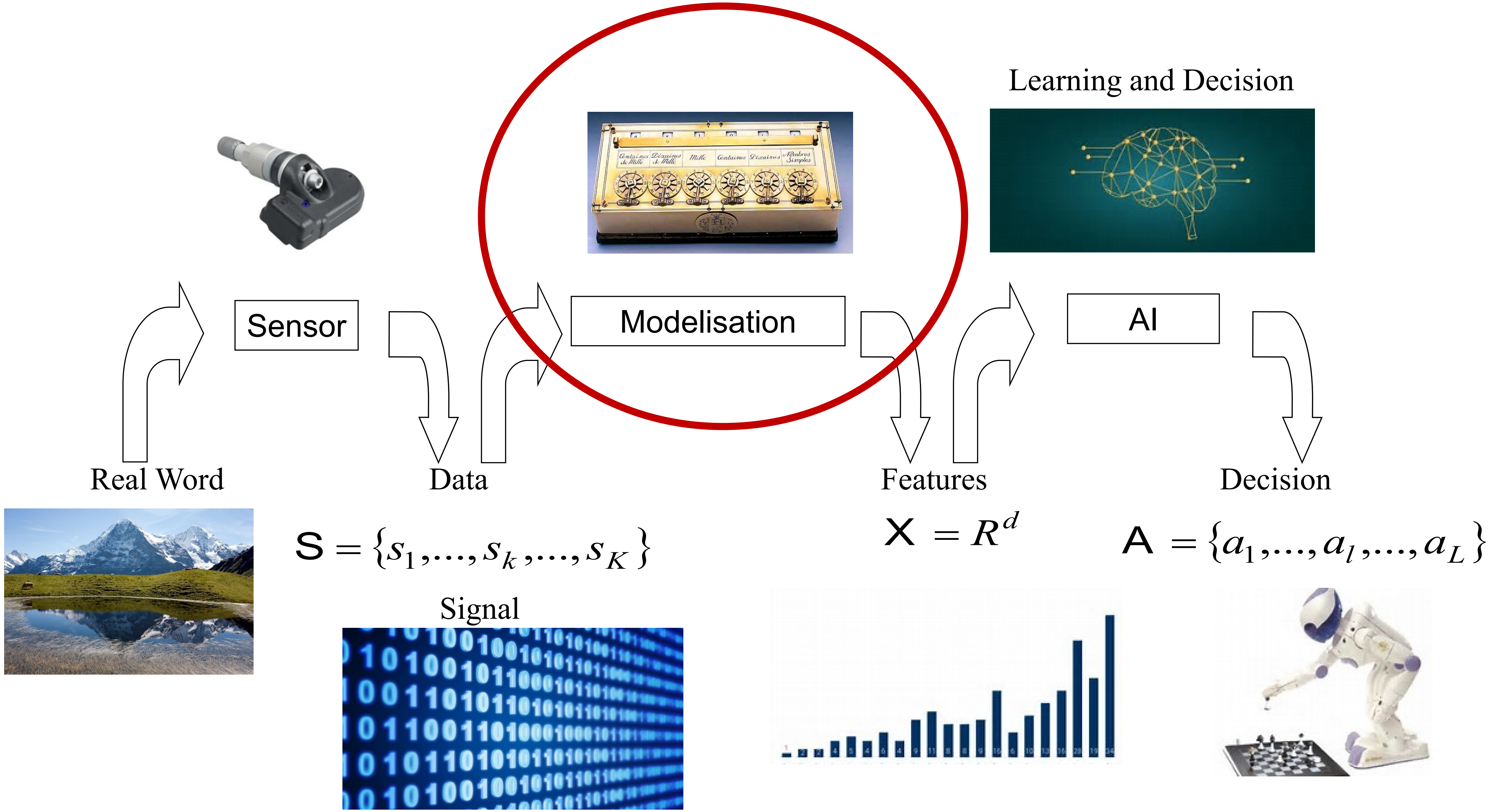
```
model = Sequential()

#**********************************
# 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]

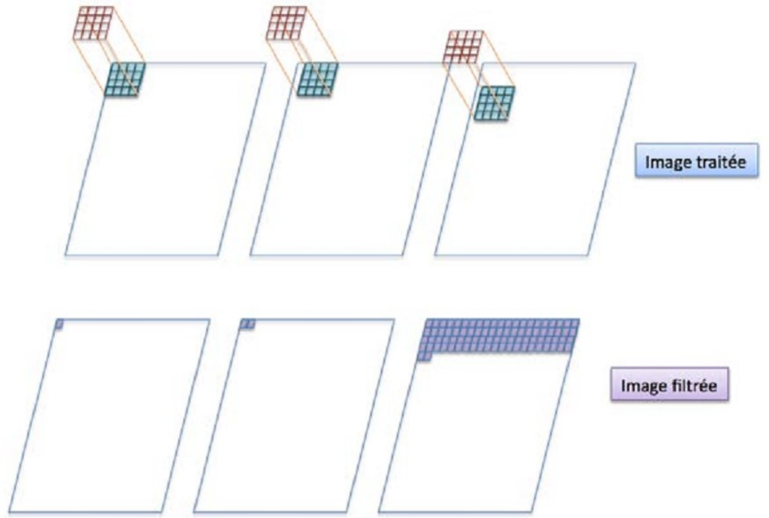


New Discrimination Process

learning



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

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

i

j

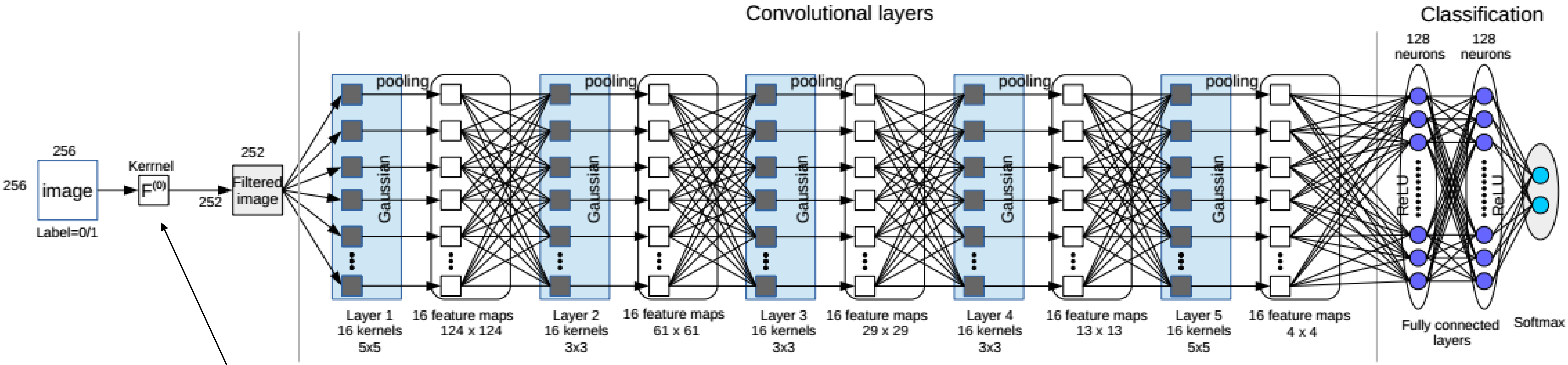
h(-1,-1)	h(0,-1)	h(1,-1)
h(-1,0)	h0,0)	h(1,0)
h(-1,1)	h(0,1)	h(1,1)

Mask

$\Rightarrow$ g(p,q)

Deep-Learning

Calculation of the features with learning



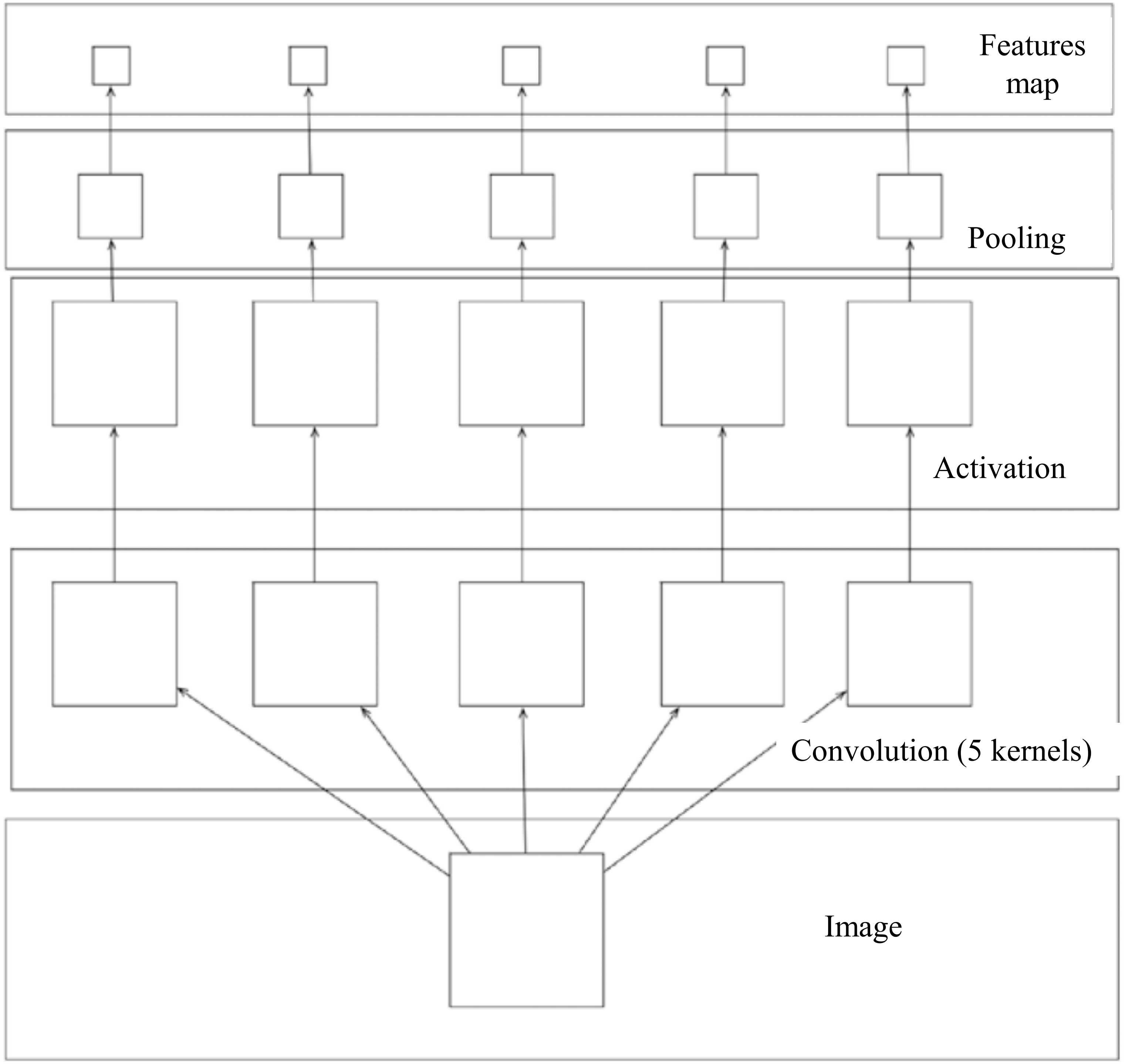
$$F^{(0)} = \frac{1}{12}\begin{pmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{pmatrix}$$

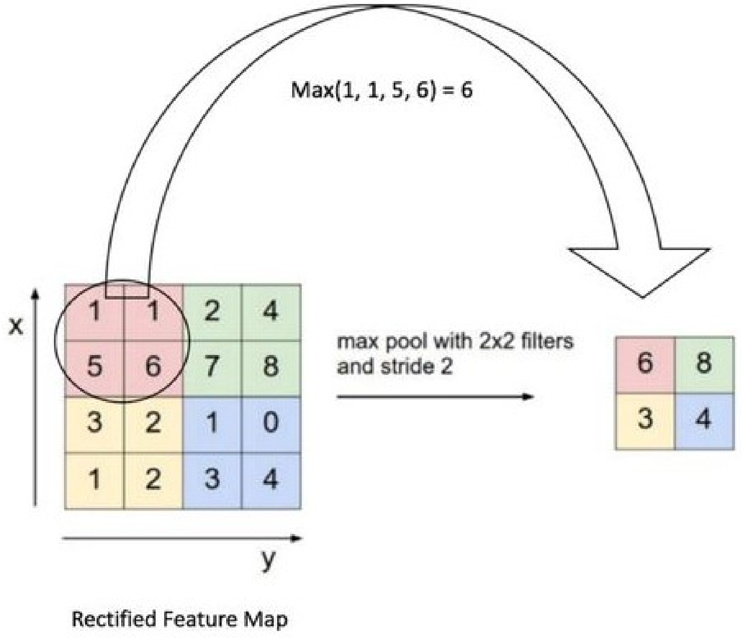
High-pass filter to improve results

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

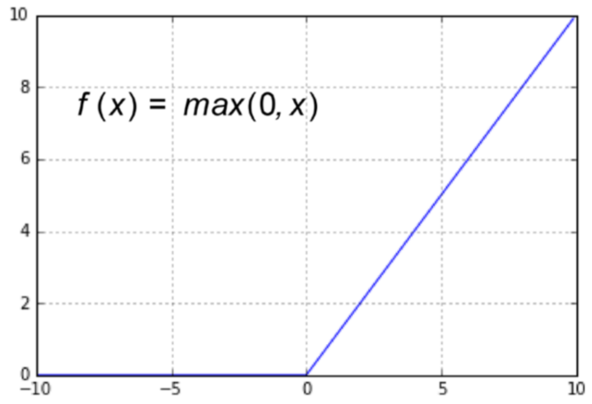
- Convolution product,
- Activation function,
- Pooling operation,
- Normalisation.

CNN





Relu

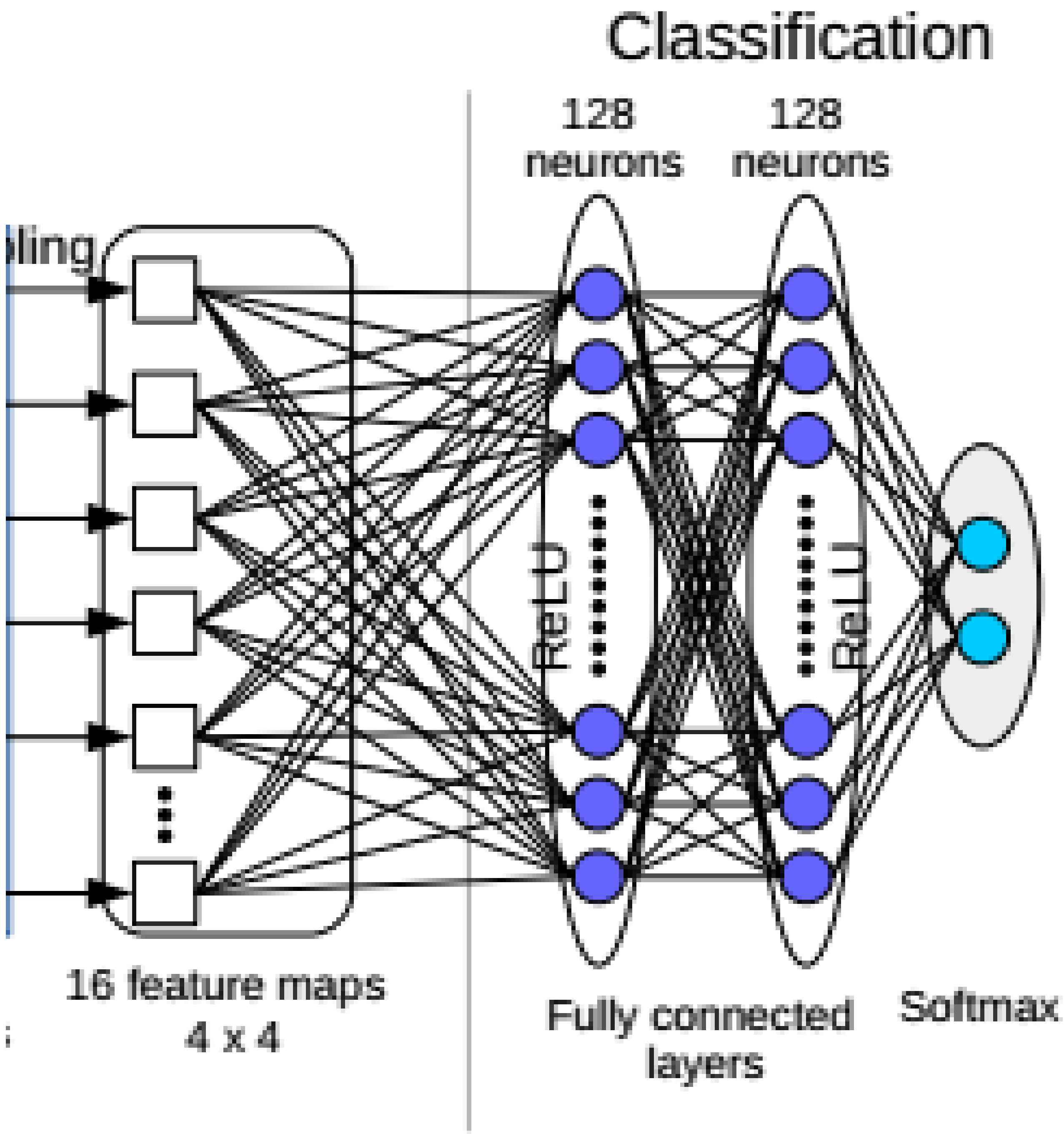


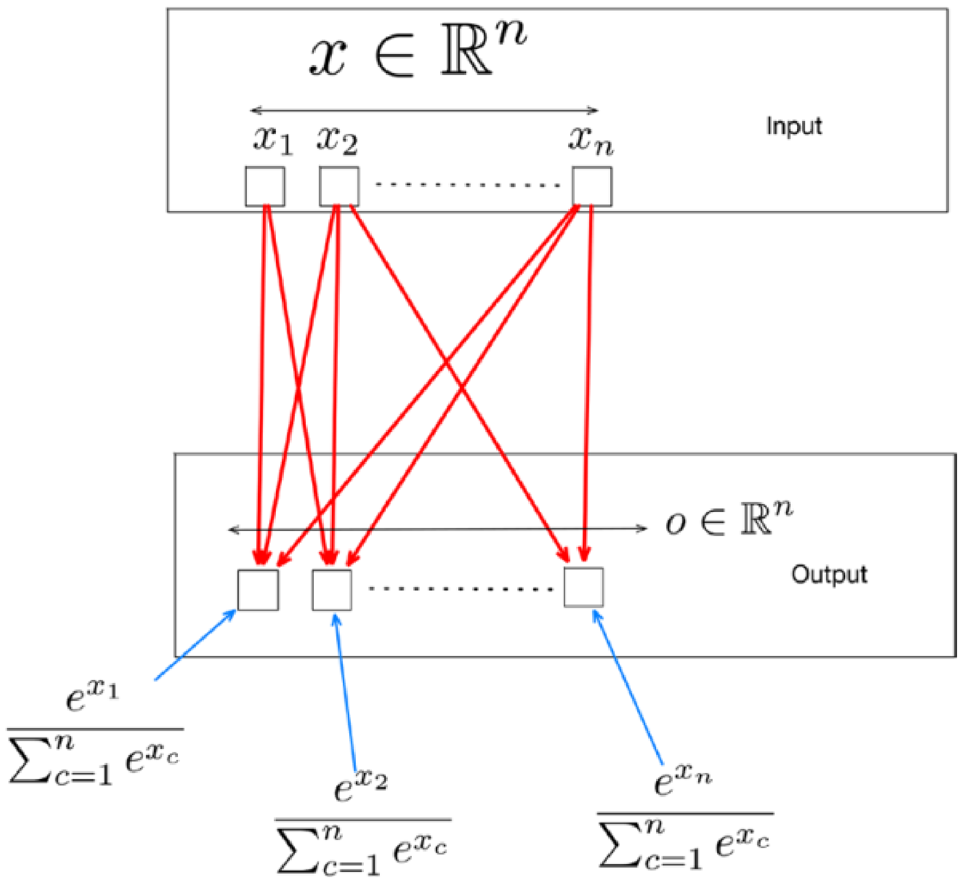
BackPropagation

- **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).

Decision

Second part: classification network

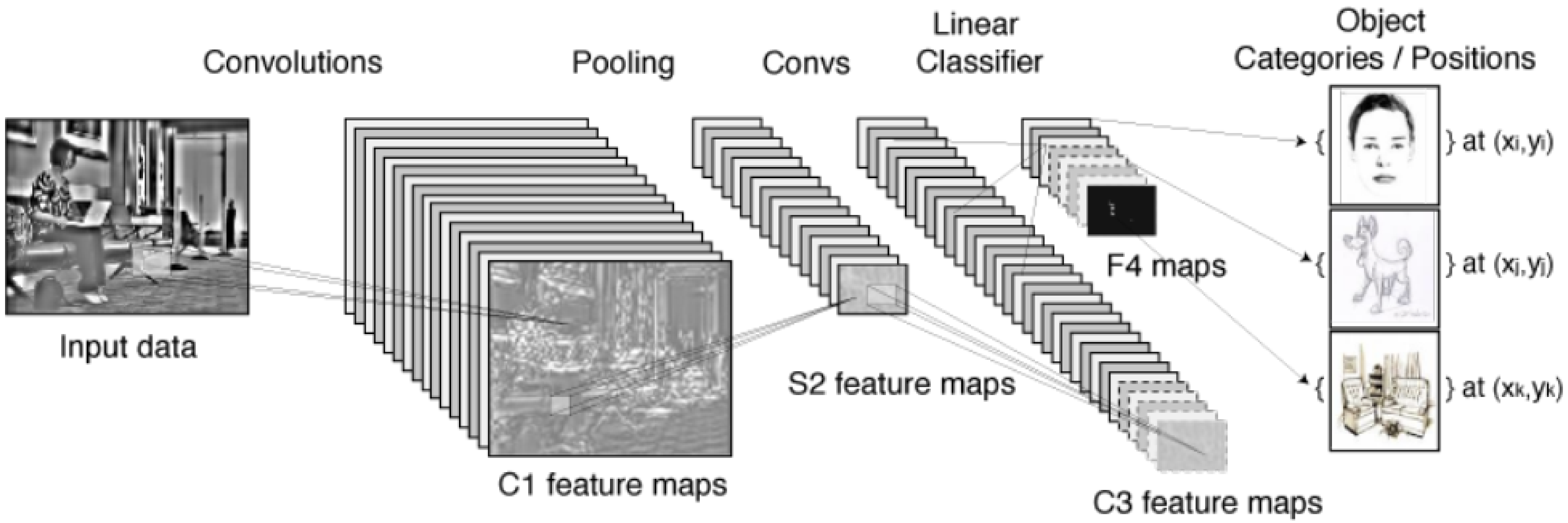




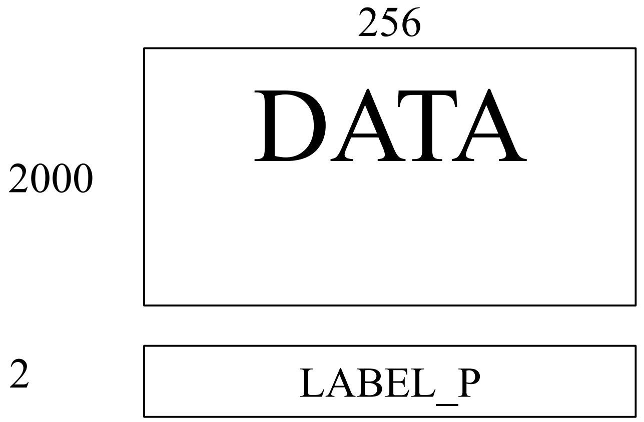
Cross-entropy

$$-\sum_{i=1}^{n} y_i \log f(x_i, \theta)$$

Abstract


Convolutions
Pooling
Convs
Linear
Classifier
Object
Categories / Positions
Input data
C1 feature maps
S2 feature maps
F4 maps
C3 feature maps
{ } at (x_i,y_i)
{ } at (x_j,y_j)
{ } at (x_k,y_k)

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)
```

```
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
```

[8.341362496139482e-05, 1.0]