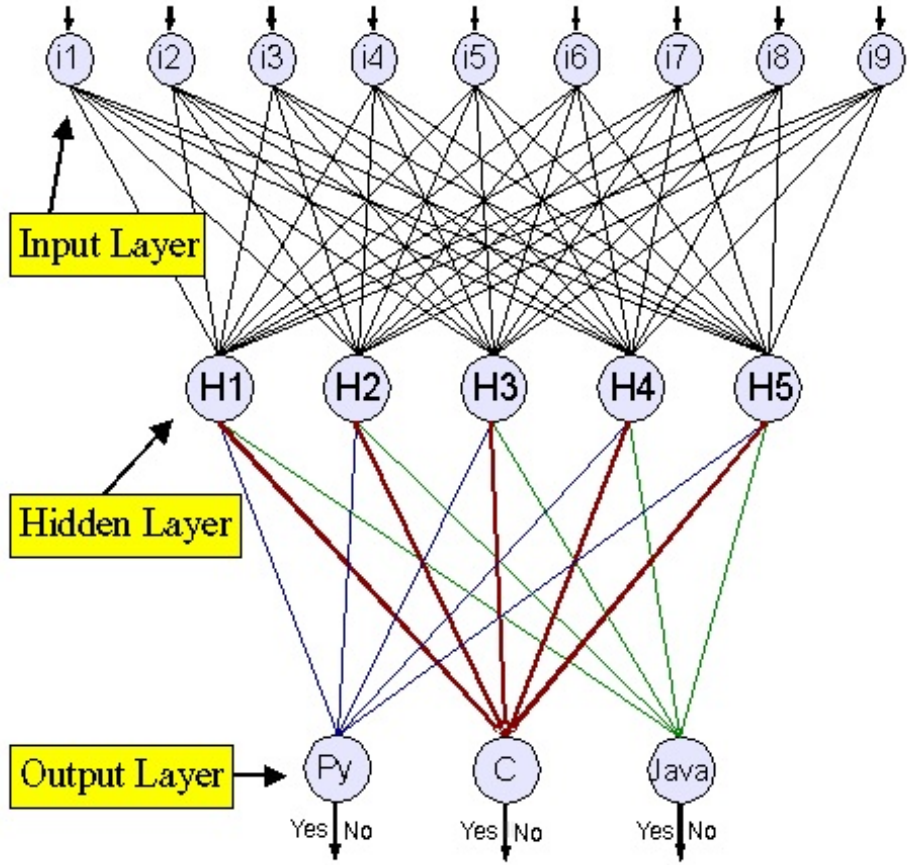
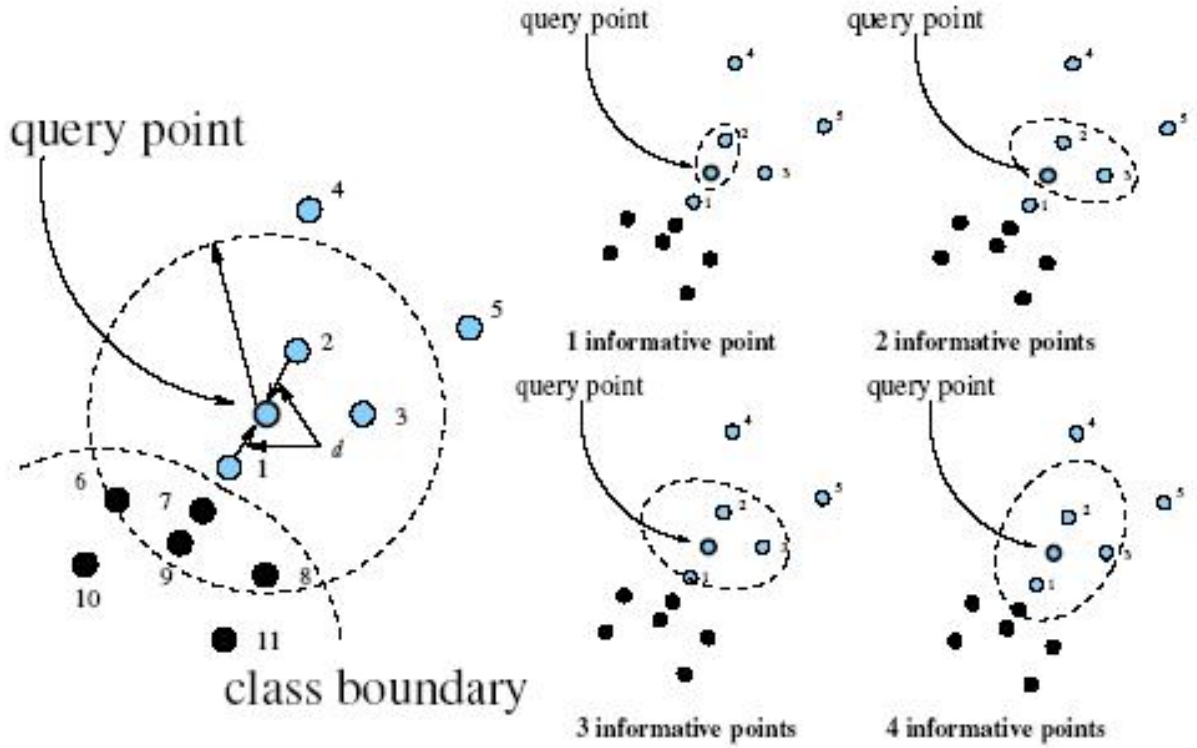
Pattern Recognition : More (Too ?) Flexible Methods
Nearest Neighbours, Trees and Neural Nets



A simple, flexible and nonparametric method : K-nearest neighbours

- Assume that n labeled points are available for training. Let $K \leq n$. Consider a **metric** d on $\mathbb{R}^D$, (ex : Euclidean distance)
- At any point x, let $\sigma = \sigma_x$ be the permutation of $\{1, \ldots, n\}$ such that
$$d(x, x_{\sigma(1)}) \leq \ldots \leq d(x, x_{\sigma(n)})$$
- Extract the K-**nearest neighbours** of x
$$\{x_{\sigma(1)}, \ldots, x_{\sigma(K)}\}$$
- Apply the **majority vote** :
$N_y = \text{Card}\{k \in \{1, ..., K\};\ y_{\sigma(k)} = y\}$, $y \in \{-1, 1\}$
$$C(x) = \arg \max_{y \in \{-1,+1\}} N_y,$$

A simple nonparametric method : K-nearest neighbours



K-nearest neighbours

Universal consistency (Stone '77)

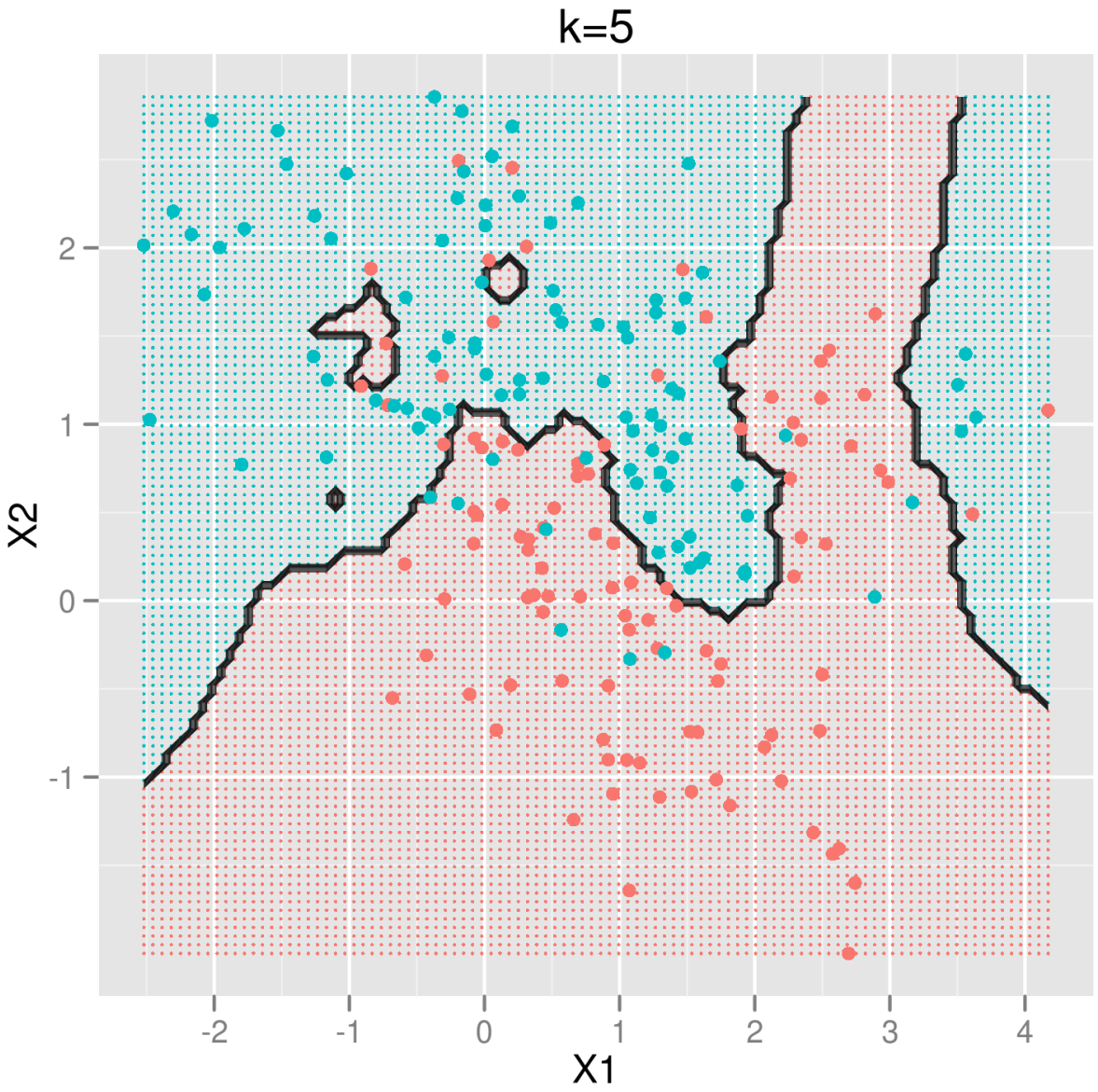
Si $k = k_n \to \infty$ et $k_n = o(n)$, the k- classifier is consistent

$$L(C_{K-NN}) - L^* \to 0, \quad as \quad n \to \infty$$

whatever the data distribution, but...

- the learning rate can be **arbitrarily slow**
- Curse of dimensionality dimension : sorting the data is **computationally costly**
- **Instabilité** : of K ? of the metric d ?
- **Metric learning** (*e.g.* Mahalanobis distance) - More in several weeks
- Many variants with **weights**

K-nearest neighbours : a too flexible method ?



Histograms - Local Averages

- Limitation of K-nearest neighbours : some of the nearest neighbours may be very far from X !
- Consider a **partition** of the input space :

$$C_1 \bigcup \cdots \bigcup C_K = \mathcal{X}$$

- Apply the **majority rule** : if X falls into C_k,
  1. Count the number of training examples with positive label in C_k
  2. If $\sum_{i:\, X_i \in C_k} \mathbb{I}\{Y_i = +1\} > \sum_{i:\, X_i \in C_k} \mathbb{I}\{Y_i = -1\}$, predict $Y = +1$. Predict $Y = -1$ otherwise.
- This rule corresponds to **"plug-in" classifier** $2\mathbb{I}\{\widehat{\eta}(x) \geq 1/2\} - 1$, where

$$\widehat{\eta}(x) = \sum_{k=1}^{K} \mathbb{I}\{x \in C_k\} \frac{\sum_{i=1}^{n} \mathbb{I}\{Y_i = +1,\ X_i \in C_k\}}{\sum_{i=1}^{n} \mathbb{I}\{X_i \in C_k\}}$$

Classification Trees : the CART algorithm

- If the partition is specified in advance (before observing the data)...

Classification Trees : the CART algorithm

- If the partition is specified in advance (before observing the data)...
  many cells may be possibly empty !

Classification Trees : the CART algorithm

- If the partition is specified in advance (before observing the data)...
  many cells may be possibly empty !
- Choose the partition **depending on the training data !**

Classification Trees : the CART algorithm

- If the partition is specified in advance (before observing the data)...
  many cells may be possibly empty !
- Choose the partition **depending on the training data !**
- The CART Book - Breiman, Friedman, Olshen & Stone (1986)
- A **greedy recursive partitioning** algorithm :
  $X = (X^{(1)}, \ldots, X^{(d)}) \in \mathbb{R}^d$

Classification Trees : the CART algorithm

- Training data $(X_1, Y_1), \ldots, ; (X_n, Y_n)$
- For any région $R \subset X$, consider the **majoriy label** : $\bar{Y}_R$, where

$$\bar{Y}_R = +1 \text{ if } \sum_{i=1}^{n} \mathbb{I}\{Y_i = +1,\ X_i \in R\} > \frac{1}{2} \sum_{i=1}^{n} \mathbb{I}\{X_i \in R\}$$

$$\text{and } \bar{Y}_R = -1 \text{ otherwise}$$

from the root node $R = X = C_{0,0}$ and the constant classifier $\bar{Y}_{C_{0,0}}$. The goal is to split the cell $C_{0,0}$

$$C_{0,0} = C_{1,0} \bigcup C_{1,1}$$

so as to refine the current rule and get

$$g_1(x) = \bar{Y}_{C_{1,0}} \mathbb{I}\{x \in C_{1,0}\} + \bar{Y}_{C_{1,1}} \mathbb{I}\{x \in C_{1,1}\}.$$

"Growing the tree"

- Splitting $C_{0,0} = X$ is performed in order to minimize $\widehat{L}_N(g_1)$, or, equivalently, the *impurity measure*

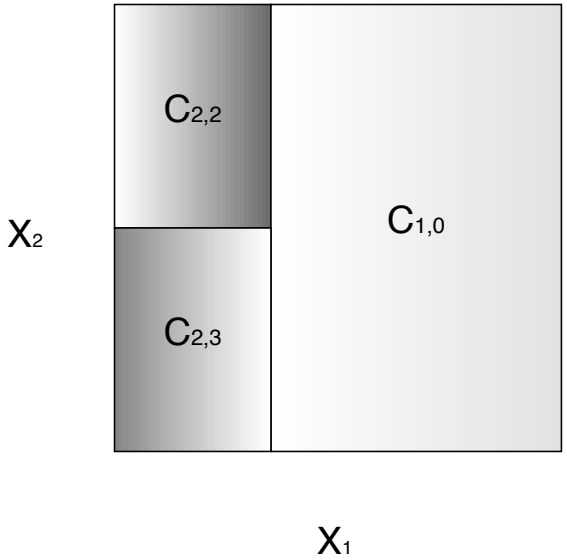
$$\sum_{i=1}^{N} \mathbb{I}\{X_i \in C_{1,0},\ Y_i \neq \bar{Y}_{C_{1,0}}\} + \mathbb{I}\{X_i \in C_{1,1},\ Y_i \neq \bar{Y}_{C_{1,1}}\}$$

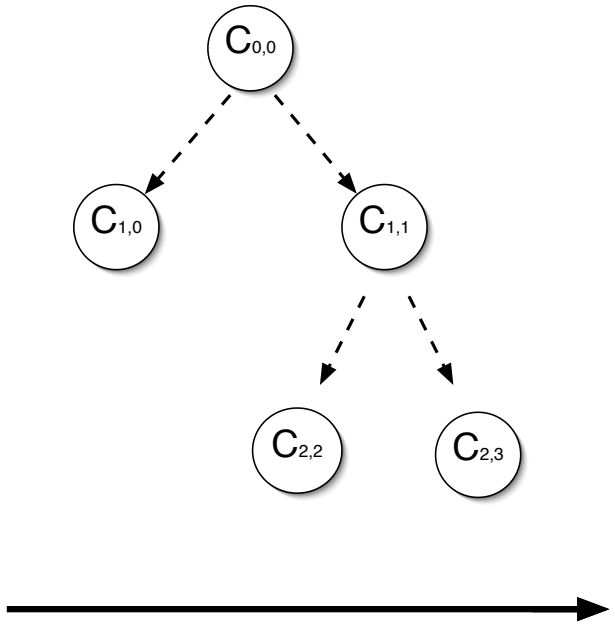
- Consider regions of the form

$$\begin{aligned} C_{1,0} &= C_{0,0} \cap \{X^{(j)} \leq s\}, \\ C_{1,1} &= C_{0,0} \cap \{X^{(j)} > s\}. \end{aligned}$$

- Ii is enough to choose the splitting thresholds among the observed values $X_i^{(j)}$'s !

"Growing the tree"





"Growing the tree"

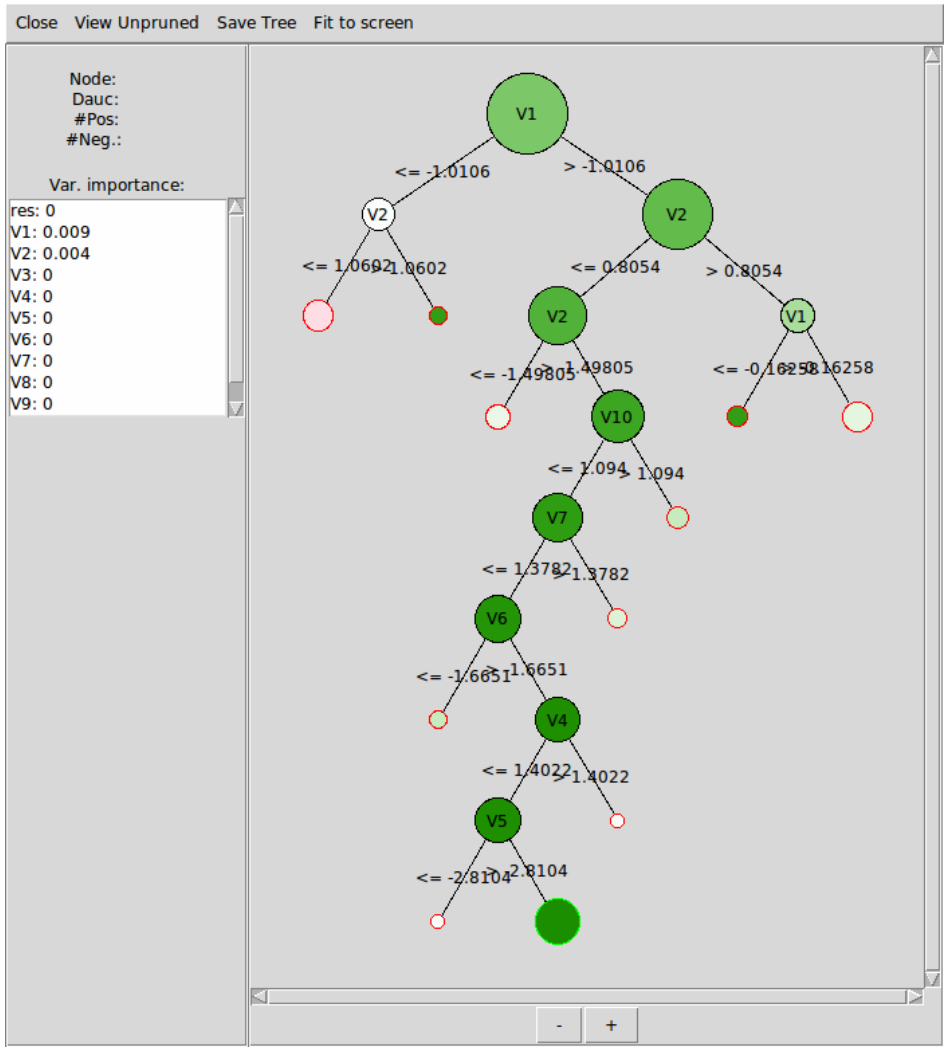
- In order to split node $C_{j,k}$, when not pure and containing at least $n_{\min}$ training data points, iterate the**double loop** :
  1. From $j = 1$ to d, find s (best splitting value for component $X^{(j)}$) so as to minimize the sum of the two impurity measures

$$C_{j,k} \cap \{X_j > s\} \quad \textit{and} \quad C_{j,k} \cap \{X_j \leq s\}$$

  2. Find the best splitting variable $X^{(j)}$

- **Impurity** measures :
  - classification error
  - Gini index
  - entropy

Classification Trees : the CART algorithm



Classification Trees : the CART algorithm

- Interpretability/explainability, visualization
- Qualitative variables, incomplete data
- Quantification of the relative importance relative of the predictive variables
- Randomisation
- Diagonal splits
- Balancing the two error types
- Extension to multiclass problems, to regression
- Model selection, complexity tuning : best sub-tree, fast **pruning** ('weakest link pruning')
- Alternative algorithm : C4.5 (Ross Quinlan)
- Popularity : the decision tree mimics an ad-hoc **expert system**
- but... its **predictive performance is moderate** in general and it exhibits a **great unstability**
- Decision trees are the essential bricks of **Ensemble Learning**

Lecture

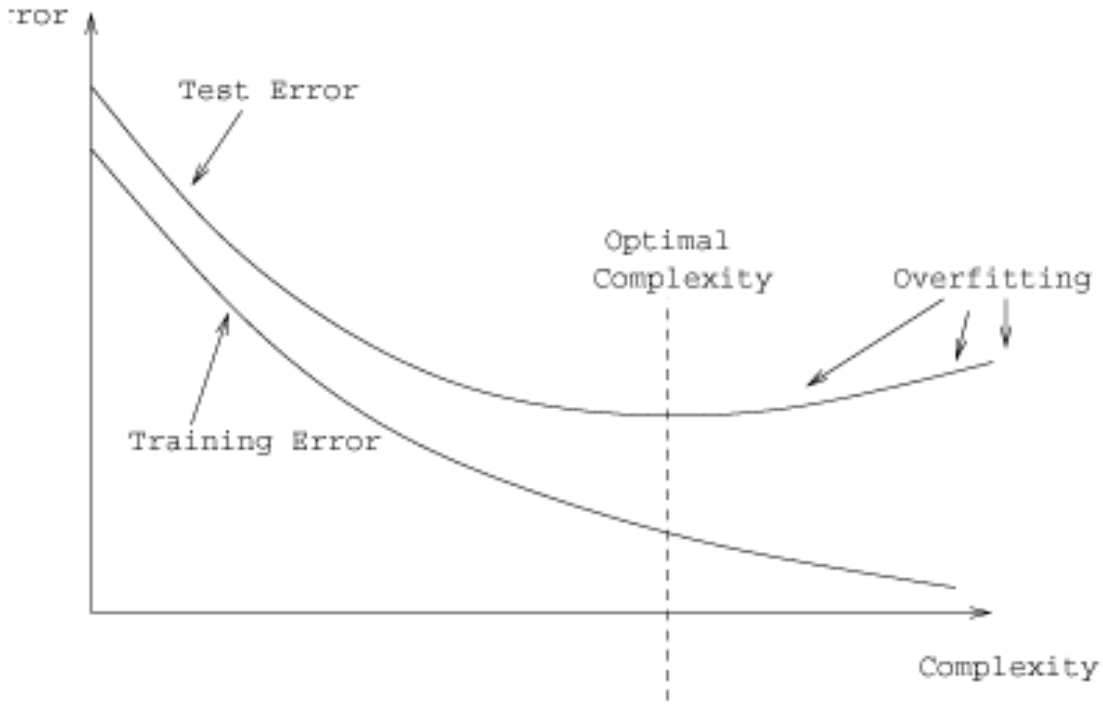
—

Model Assessment
Model Selection

Agenda

- Generalization ability
- Bias, variance and model complexity
- The "data-rich situation" : Train-Validation-Test
- Cross-validation : a popular method for prediction error estimation
- Bootstrap techniques

Looking for the right amount of complexity



Errors, training errors, test/generalization errors

- Learning is based on a training sample

$$\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$$

- The classifier $\widehat{C}_n \in \mathcal{G}$ selected through an "ERM like" method is **random**, depending on $\mathcal{D}_n$, as well as its **error** :

$$L(\widehat{C}_n) = \mathbb{E}\left[\mathbb{I}\{Y \neq \widehat{C}_n(X)\} \mid \mathcal{D}_n\right]$$

  Expectation is taken over a pair (X, Y) independent from training data $\mathcal{D}_n$

- One may take next expectation over $\mathcal{D}_n$

$$Err = \mathbb{E}\left[L(\widehat{C}_n)\right]$$

Methods for performance assessment, for model selection

- Training error is not a good estimate !

$$\widehat{L}_n(\widehat{C}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{Y_i \neq C(X_i)\}$$

  It vanishes as soon as the class $\mathcal{G}$ is complex enough
  $\Rightarrow$ Overfitting and poor generalization

- The objective is twofold
  - Model selection : choose the best model among a collection of models
  - Model assessment : for a given model, estimate its generalization error

When data are not expensive

- Divide the data into three parts :

  Training - Validation - Test

- Typical choice : 50% - 25% - 25%

- $K \geq 1$ model candidates : $\mathcal{G}_1, \ldots, \mathcal{G}_K$

  - For each $k \in \{1, \ldots, K\}$, apply ERM to training data $\Rightarrow \widehat{C}^{(k)}$
  - Use validation data to find the "best" $\widehat{k} \in \{1, \ldots, K\}$
  - Estimate the error using the test data (independent from $\widehat{k}$)

- How to proceed in a data-poor situation ?

  Complexity regularization (structural risk minimiation),
  resampling methods, *etc.*

Cross-Validation

- Goal : estimate the generalization error
- Let $K \geq 1$ (typical choices are 5 or 10), "K-fold cross-validation"
  (K=n "leave-one-out" estimation)
- Split the data into K parts (of same size)
- For all $k \in \{1, \ldots, K\}$,
  - learn $\widehat{C}^{(-k)}$ based on all data except the k-th part
  - calculate the error of $\widehat{C}^{(-k)}$ over the k-th part
- Average the K quantities

"Pulling yourself up by your own bootstrap" (Baron de Münchausen)

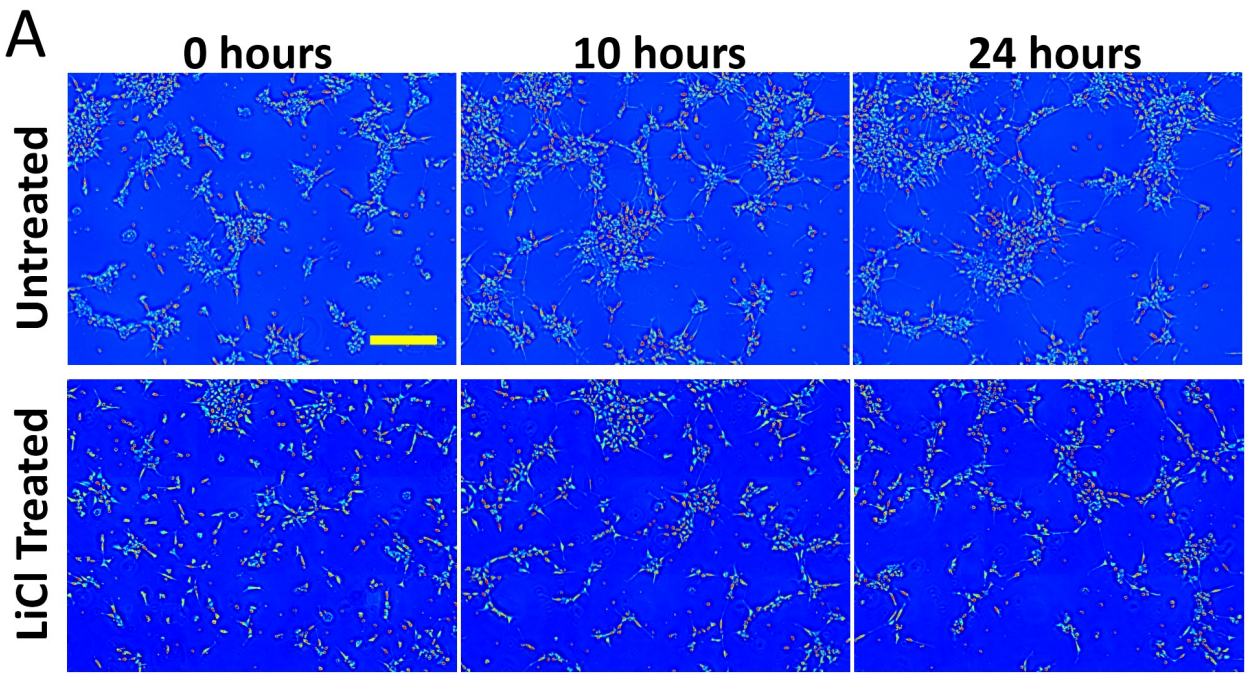
- Bootstrap (the plug-in principle) : estimate the distribution of

$$\mathbb{E}^*[\mathbb{I}\{\widehat{C}(X) \neq Y\}]$$

  where $\mathbb{E}^*[.]$ is the expectation w.r.t. the empirical df of the $(X_i, Y_i)'s$

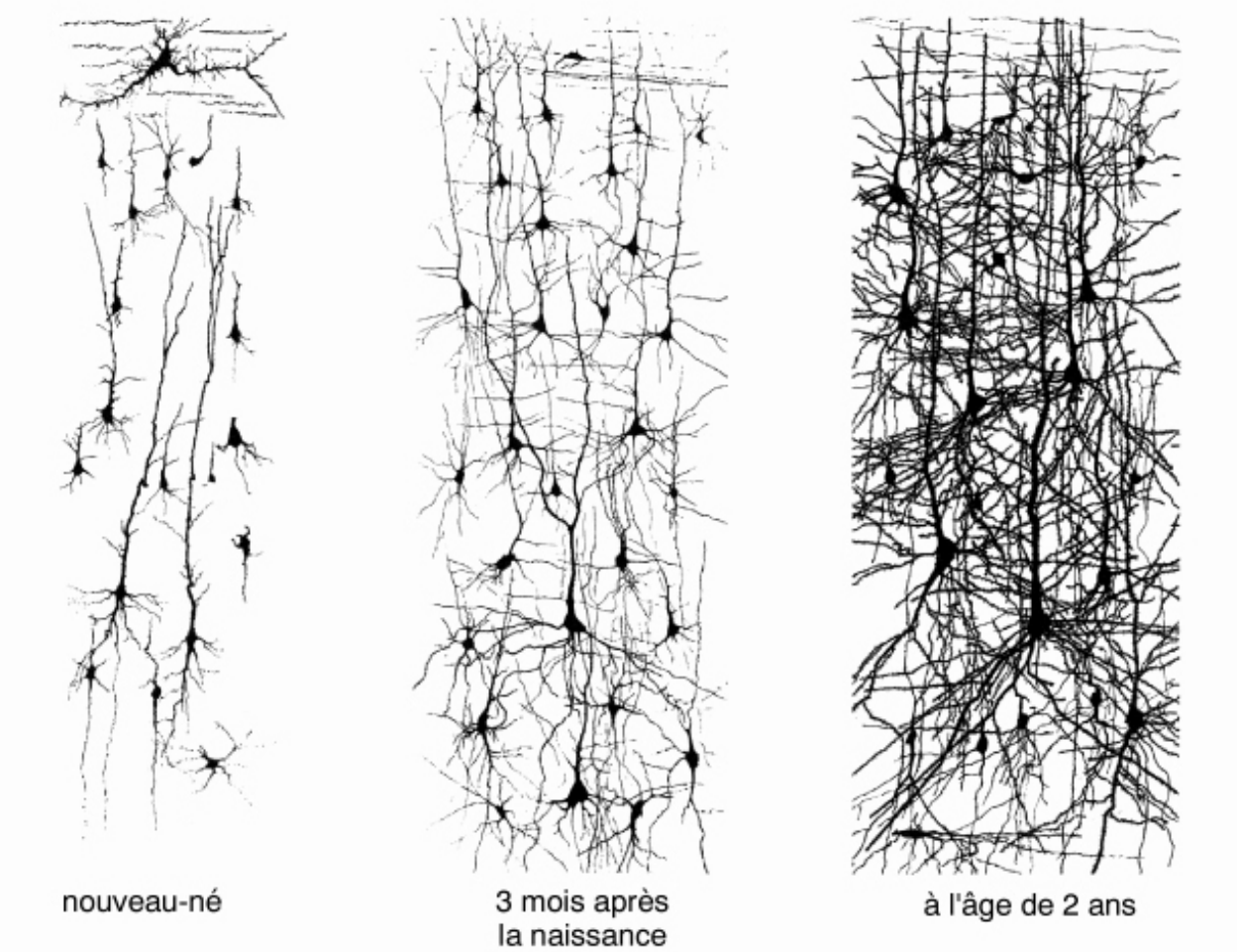
- Heuristics : replace the unknown df by an estimate
- Monte-Carlo approximation
- Higher-order validity

Neuron network growth over 24 hours



In 2014, the group of Gabriel Popescu at Illinois U. visualized a growing net of baby neurons using spatial light interference microscopy (SLIM). Ref : `http://light.ece.illinois.edu/wp-content/uploads/2014/03/Mir_SRep_2014.pdf`
Video : `https://youtu.be/KjKsU_4s0nE`

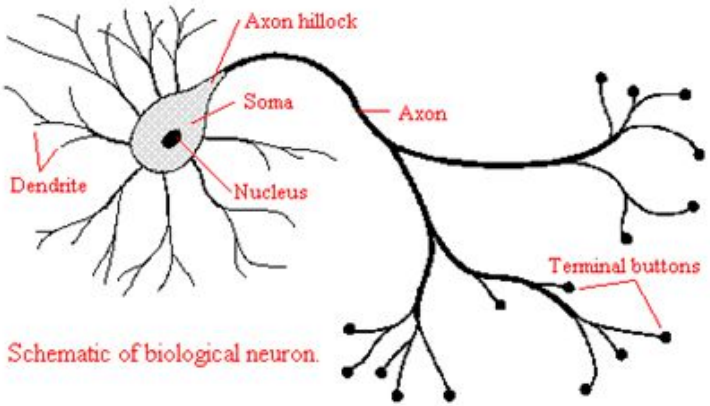
Child neuron network growth

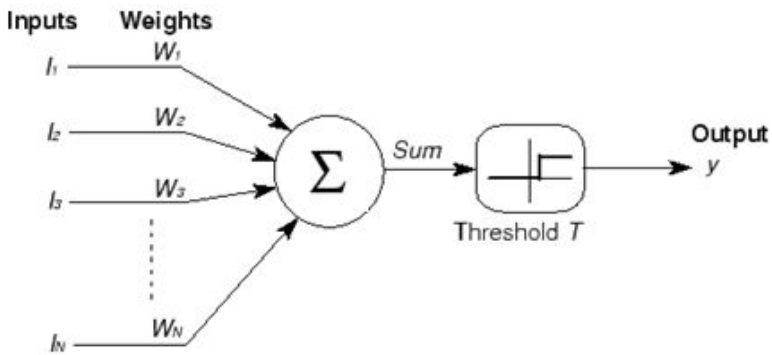


Développement des réseaux de connections entre les neurones chez l'enfant.

Re : Museum de Toulouse `http://www.museum.toulouse.fr/-/connecte-a-vie-notre-cerveau-le-meilleur-des-reseaux-2-3-`

Le neurone

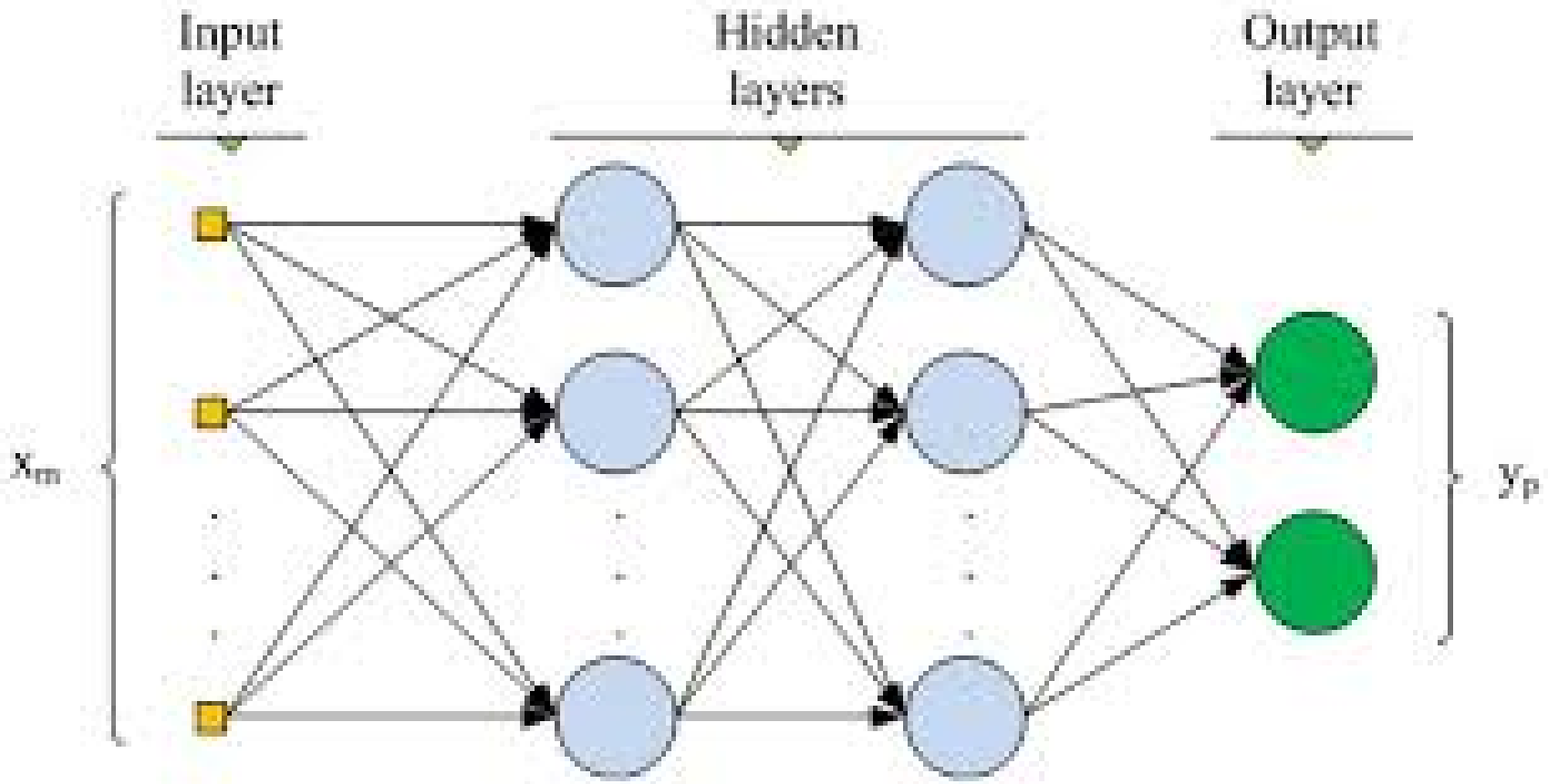




Neurone biologique

Neurone artificiel

Network of Artificial Neurons (multilayer perceptron, MLP)



From the Artificial Neuron model to Neural Networks 1/2

- Artificial Neuronl : Mc Cullogh et Pitts, 1943
- Learning the Artificial Neuron model : the Perceptron by Rosenblatt, 1957
- Minsky and Papert : limitation of the Perceptron, 1959
- Learning a multi-layer perceptron by gradient backpropagation, Y. Le Cun, 1985, Hinton and Sejnowski, 1986.
- Multi-Layer Perceptron = a universal approximant, Hornik et al. 1991
- Convolutional networks, 1995, Y. Le Cun and Y. Bengio
- Between 1995 et 2008, the domain is flat (non convexity, computationally demanding, no theory)

From the Artificial Neuron model to Neural Networks 1/2

- Democratization of GPU's (graphical processing units) 2005
- Large image databases : Imagenet, Fei-Fei et al. 2008 (now much more than 10^6 images)
- Deeper and deeper neural networks, learned by means of massive databases
- Initialization with unuspervised learning (autoencoder)
- Word2vec (Mikolov et al. 2013)
- Dropout (Srivastava et al. 2014)

Artificial Neuron

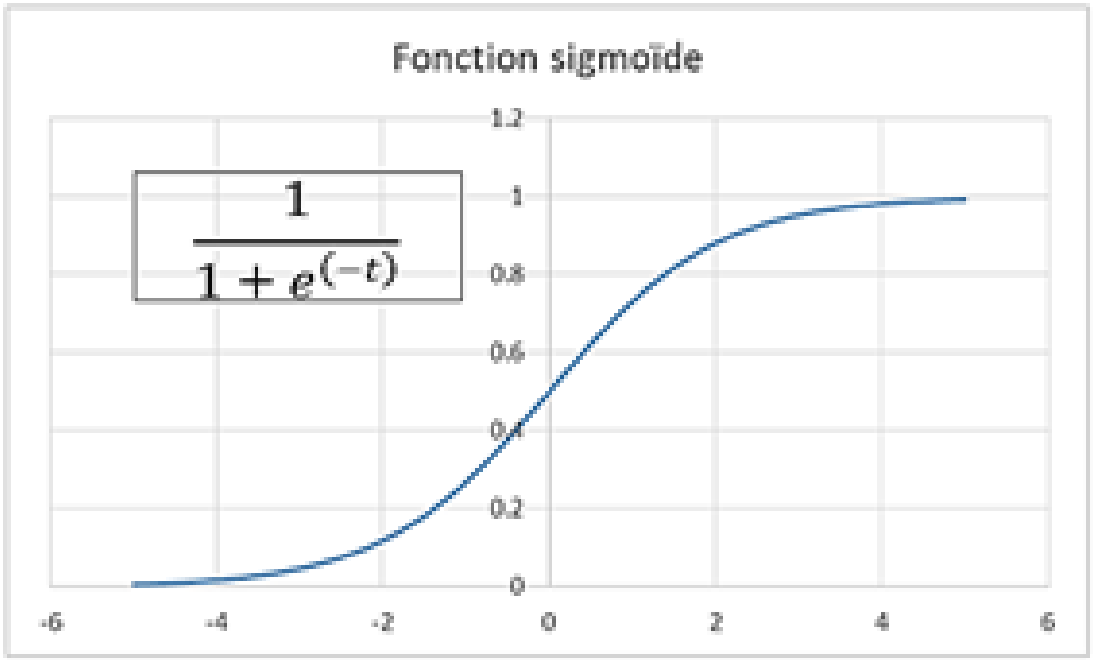
- activation function (e.g. sign)
- weight vector and bias (intercept)

$$f(x) = g(w^T x + b) \tag{1}$$

Choose g differentiable preferably (cf gradient optimization techniques)

Activation function for the Artificial Neuron

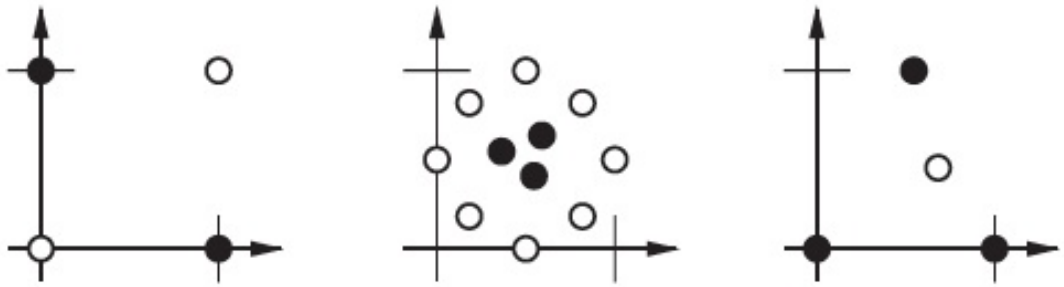
For instance :



One also uses hyperbolic tangent *tanh* (values in the range $(-1,\ 1)$).

Limitation of the Artificial Neuron model

Limited to linearly separable data :



Add a processing intermediary layer

Now, compute :

$$f(x) = g(\Phi(x)^T w + b)$$

Feature map or latent representation.
Flexibility of neural networks : the feature map Φ is learned from the training data.

Universal Approximation

In 1991, Hornik et al. prove that MLP's with one hidden layer and $p+1$ input is dense in the space of real valued continuous functions on $\mathbb{R}^p$. A MLP with one hidden layer is a **universal approximant**.

Universal Approximation

In 1991, Hornik et al. prove that MLP's with one hidden layer and $p+1$ input is dense in the space of real valued continuous functions on $\mathbb{R}^p$. A MLP with one hidden layer is a **universal approximant**.
Some other flexible/rich classes of decision functions (more next week !) :

- Linear regressor : NO
- SVM with a universal kernel, e.g. Gaussian kernel : YES
- Random Forests : YES
- Boosting stumps : YES

Example of a multi-layer neural network "feedforward"

Consider a MLP with an output layer of size $K = 1$, a hidden layer of size $M + 1$, an input vector of size $p + 1$

Class of fonctions $\mathcal{H}_{mlp} = \{h_{mlp} : \mathcal{R}^{p+1} \to \mathcal{Y}\}$

for the regression problem (continuous output Y)

$$h_{MLP}(x) = \sum_{j=0}^{M} w_j^{(2)} z_j \quad (2)$$

$$z_j = \tanh(a_j) \quad (3)$$

$$a_j = \sum_{i=0}^{p} w_{ji}^{(1)} x_i \quad (4)$$

About choosing the activation function

Hyperbolic tangent is chosen here as activation function, differentiable.

$$h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \quad (5)$$

$$h'(a) = 1 - h(a)^2 \quad (6)$$

This choice is appealing from a computational perspective since the derivative can be expressed in terms of $h(a)$. A similar property holds for the sigmoid :

$$g(a) = \frac{1}{1 + \exp(-\frac{1}{2}a)}.$$

Architecture of a multi-layer neural network "feedforward"

- The single ouptut of a regressor MLP predicts a real value
- For classification with K classes, one chooses K outputs with the sigmoid function or the softmax function $softmax(z) = (softmax_1(z), \ldots, softmax_K(z))$, with

$$softmax_i(z) = \frac{\exp(z_i)}{\sum_{j=1^K} \exp(z_j)}$$

- For a multi-output regression with K outputs, take K linear outputs for the architecture

Architecture of a multi-layer neural network "feedforward"

Consider a MLP with an output layer of size $K = 1$, a hidden layer of size $M + 1$, an input vector of size $p + 1$ for a regression task

Class of functions $\mathcal{H}_{mlp} = \{h_{mlp} : \mathcal{R}^{p+1} \rightarrow \mathcal{Y}\}$

$$h_c(x) = g(\sum_{j=0}^{M} w_{jc}^{(2)} z_j) \quad (7)$$

$$z_j = g(a_j) \quad (8)$$

$$a_j = \sum_{i=0}^{p} w_{ji}^{(1)} x_i \quad (9)$$

with $g(t) = \frac{1}{1+\exp(-1/2t)}$.

Learning from Training Data

$$\mathcal{L}(W;\mathcal{S}) = \sum_{n=1}^{N} \ell(h(x_n), y_n))$$

Regression :

$$\ell(h(x_n), y_n) = (h(x_n) - y_n)^2$$

Classification (maximize the likelihood) : Interpret $f_c(x) = p(y = c|x)$ (multiple outputs : one may use the softmax function)

$$\ell(h(x), y) = -\log f_y(x)$$

To be notice : $\mathcal{L}$ is non convex and has many local minima

- Our best : find a good local minimum
- For this reason MLP had been abandoned for a long time, SVM/SVR were preferred, easier models to optimize

Gradient backpropagation

- When applying gradient descent, the error is backpropagated through all the layers, starting from the last one,
- One uses the **chain rule** for differentiation : $\frac{\partial L(W)}{\partial w_{ji}^{(1)}} = \frac{\partial L(W)}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}}$ to modify the weights of the hidden layer.
- Once all the modifications are computed, the network is updated.
- Backpropagation can be applied locally or globally

Gradient backpropagation

References :

Y. LeCun : Une procédure d'apprentissage pour réseau à seuil asymmétrique (a Learning Scheme for Asymmetric Threshold Networks), Proceedings of Cognitiva 85, 599-604, Paris, France, 1985.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back-propagating errors. Nature, 323, 533–536.