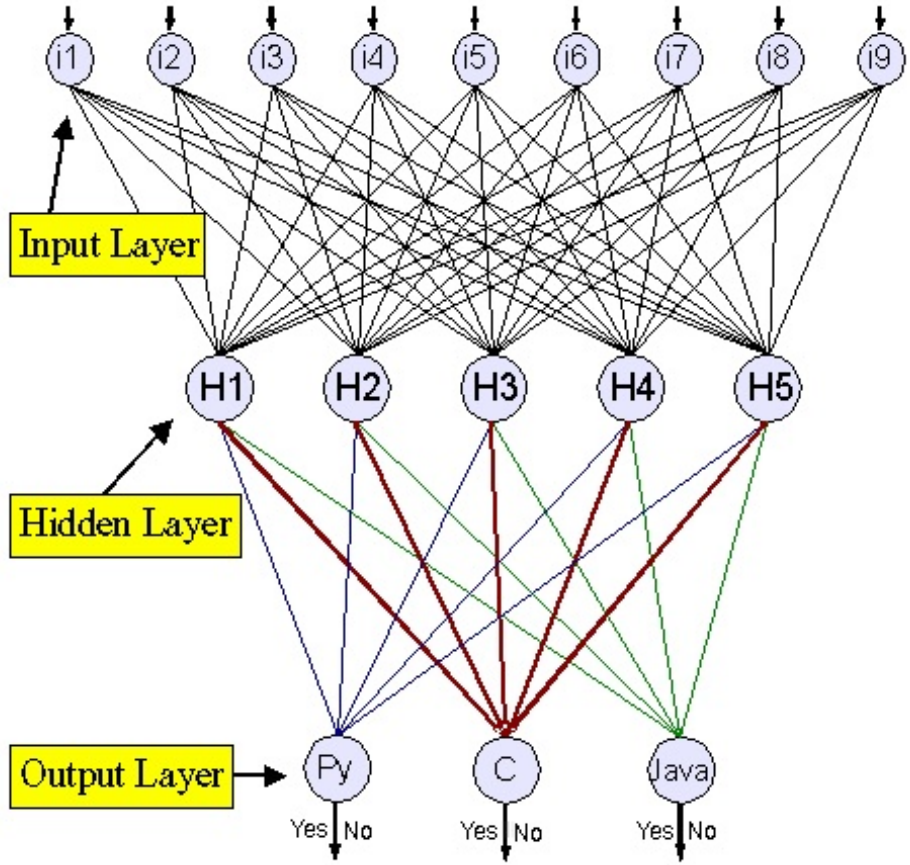
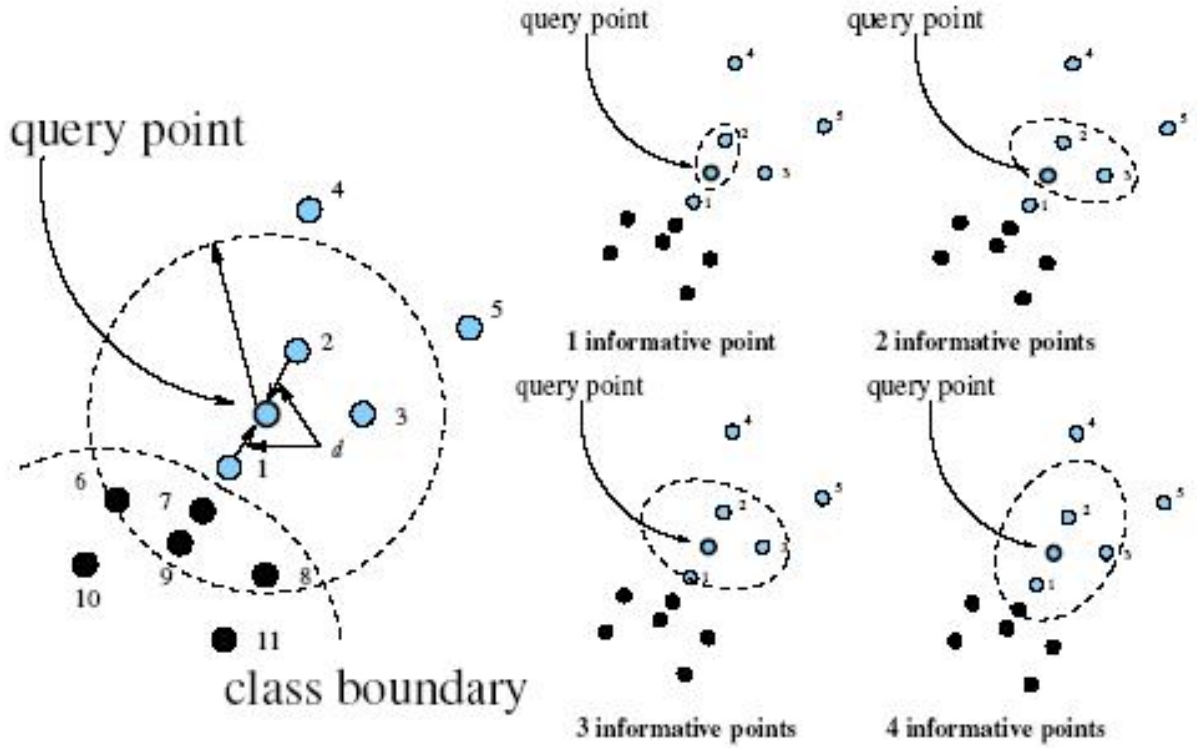
Pattern Recognition : More (Too ?) Flexible Methods
Nearest Neighbours, Trees and Neural Nets



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)

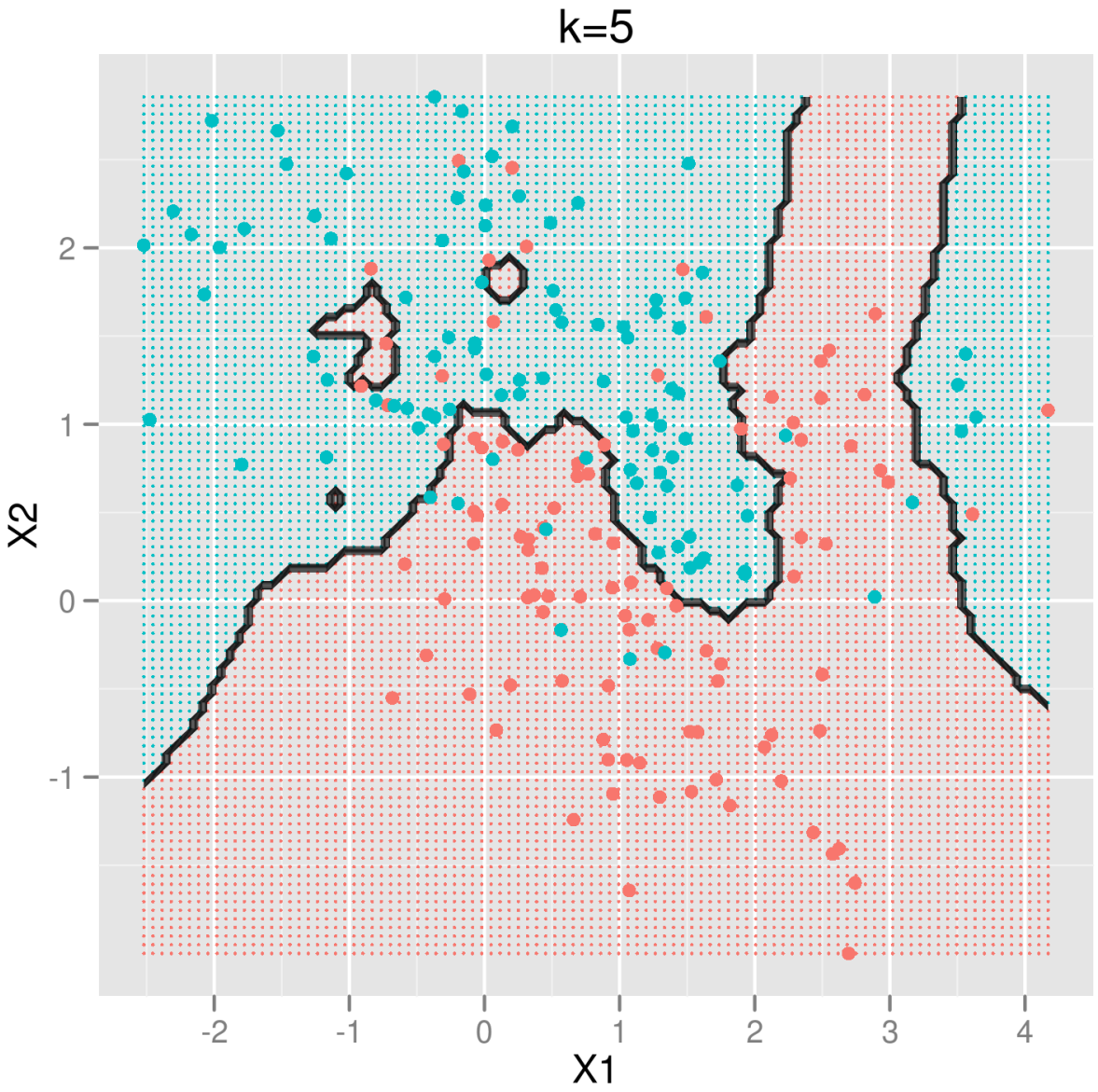
Si $k = k_n \to \infty$ et $k_n = o(n)$, the k- classifier is consistent

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

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*

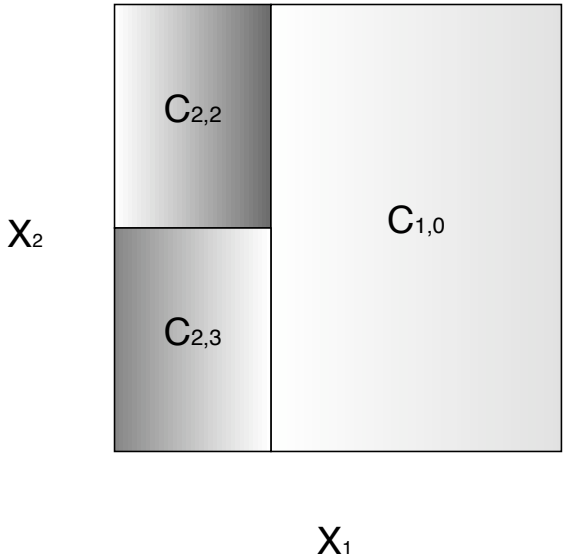
$$\sum_{i=1}^{N} \mathbb{I}\{X_i \in C_{1,0},\ Y_i \neq \bar{Y}_{C_{1,0}}\} + \mathbb{I}\{X_i \in C_{1,1},\ Y_i \neq \bar{Y}_{C_{1,1}}\}$$

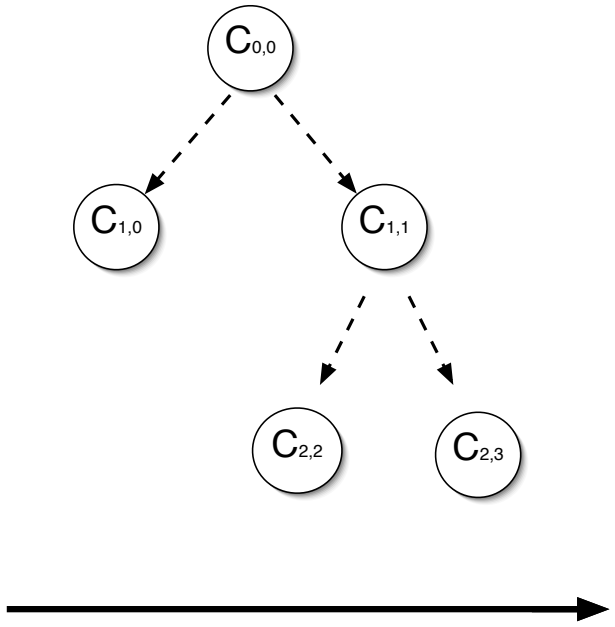
- Consider regions of the form

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

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

"Growing the tree"





"Growing the tree"

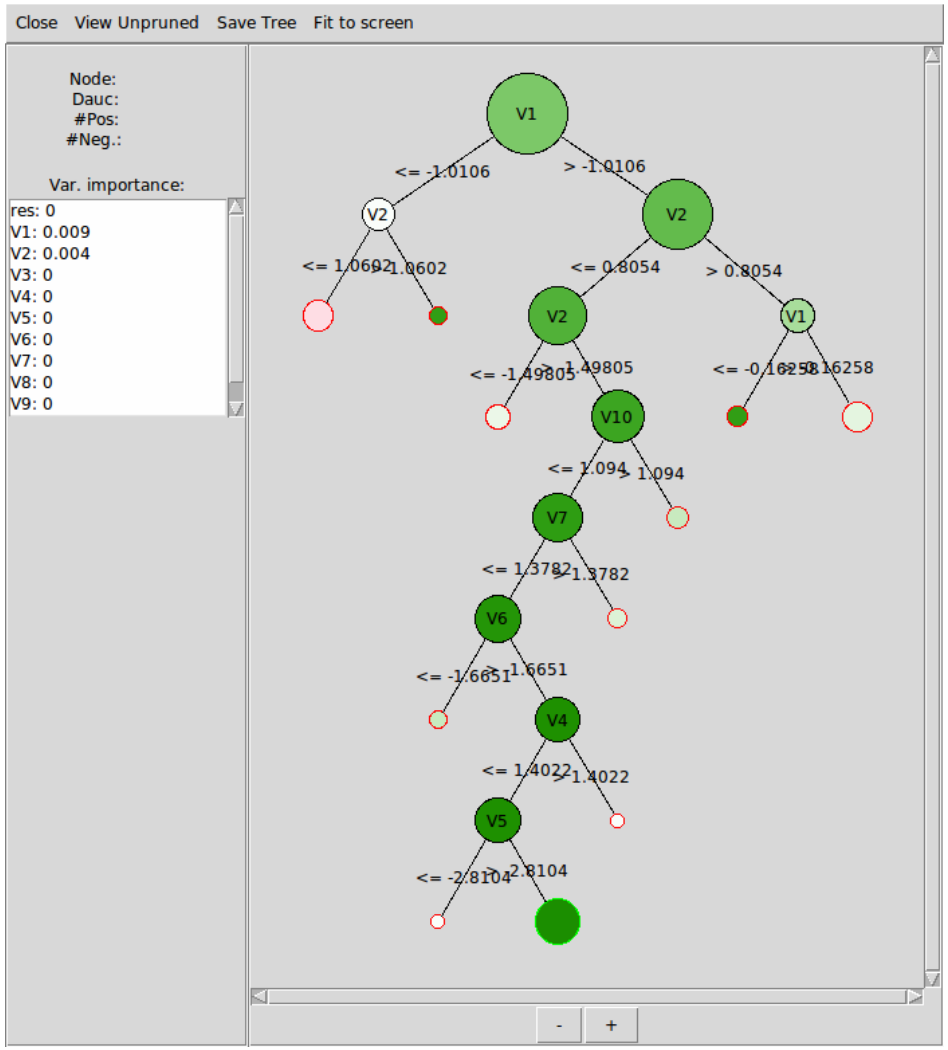
- In order to split node $C_{j,k}$, when not pure and containing at least $n_{\min}$ training data points, iterate the**double loop** :
  1. From $j = 1$ to d, find s (best splitting value for component $X^{(j)}$) so as to minimize the sum of the two impurity measures

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

  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

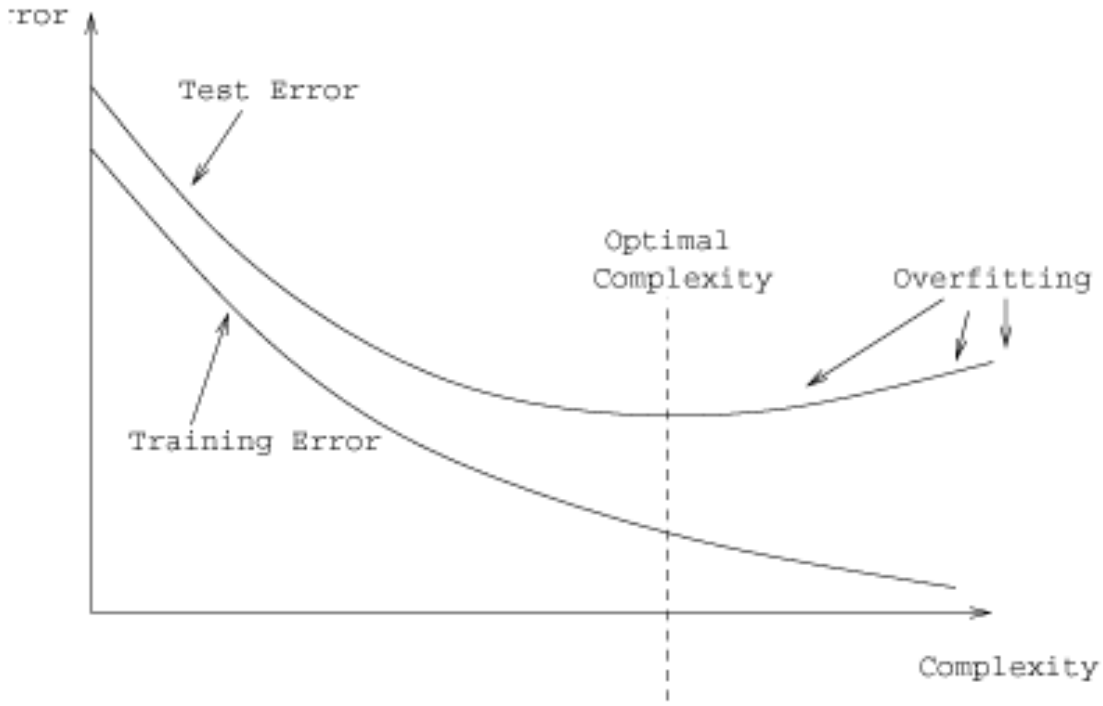
—

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)

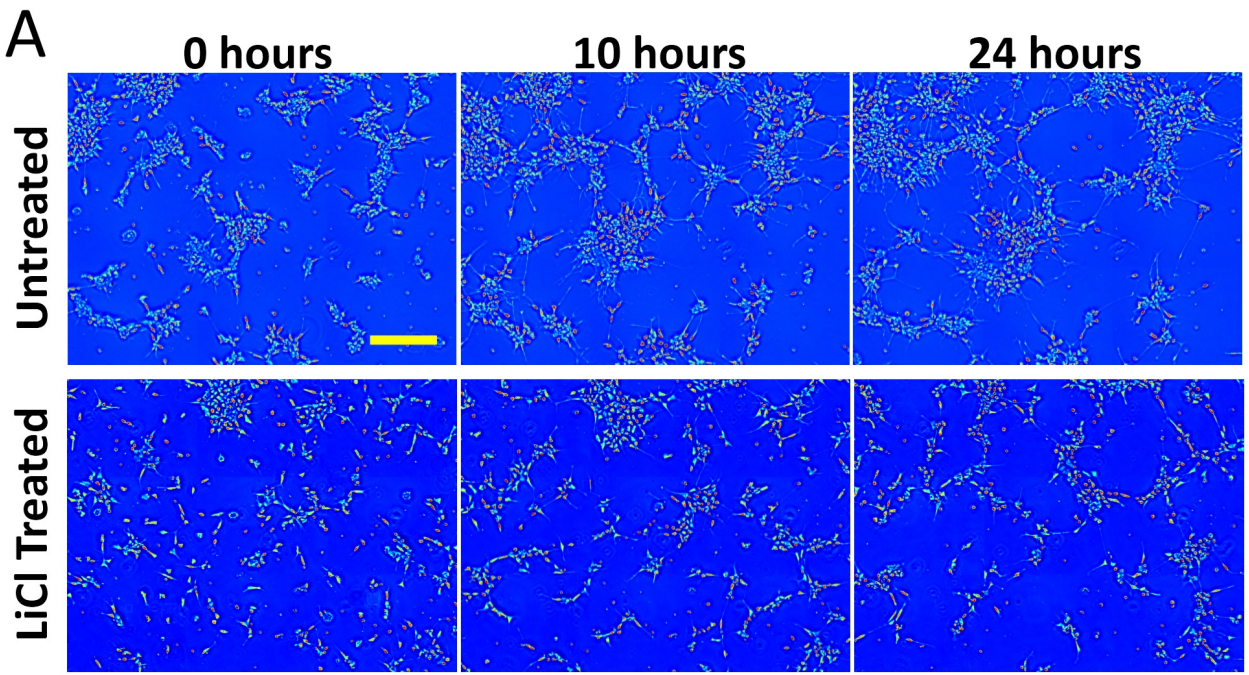
- Bootstrap (the plug-in principle) : estimate the distribution of

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

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

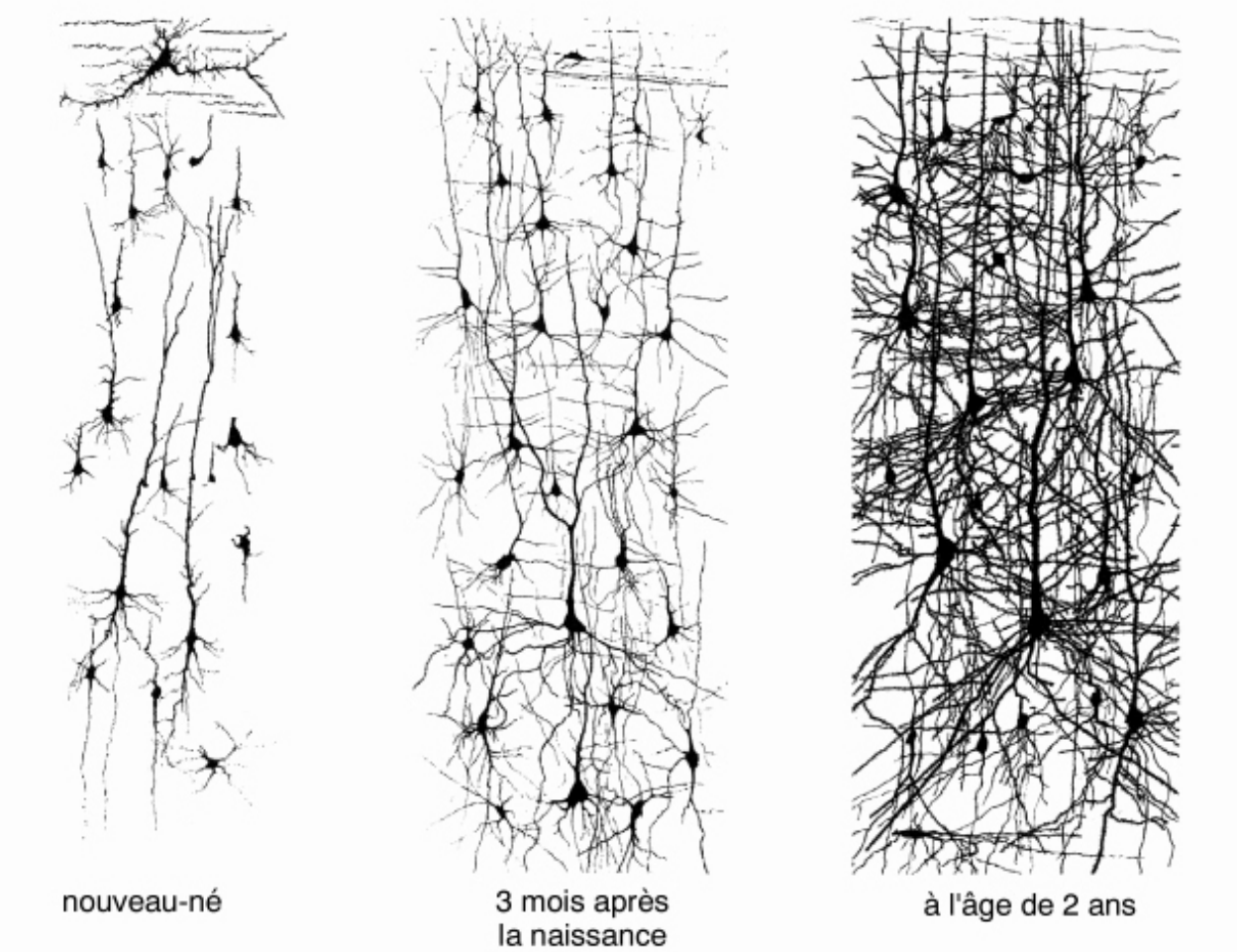
- Heuristics : replace the unknown df by an estimate
- Monte-Carlo approximation
- Higher-order validity

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`

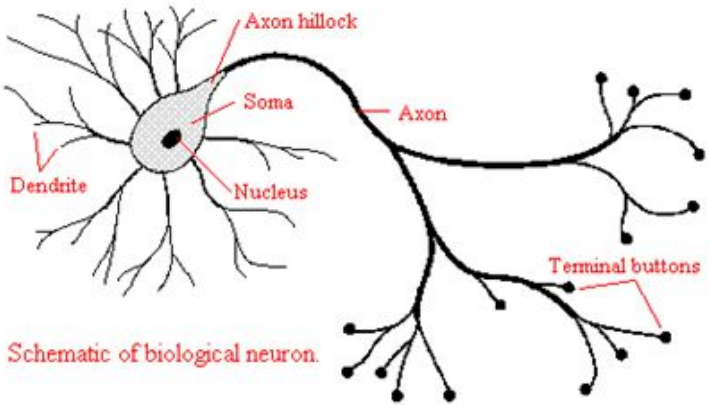
Child neuron network growth

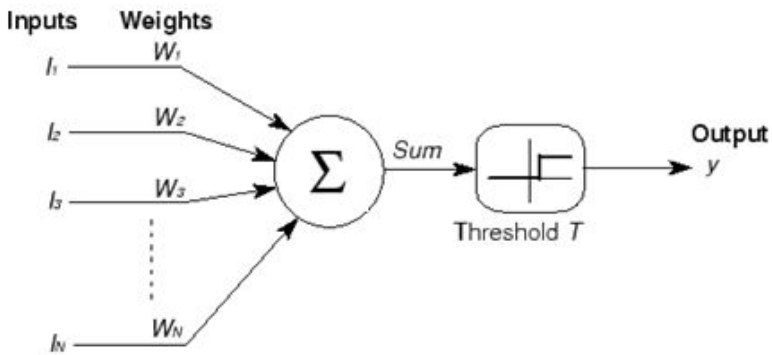


Développement des réseaux de connections entre les neurones chez l'enfant.

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

Le neurone

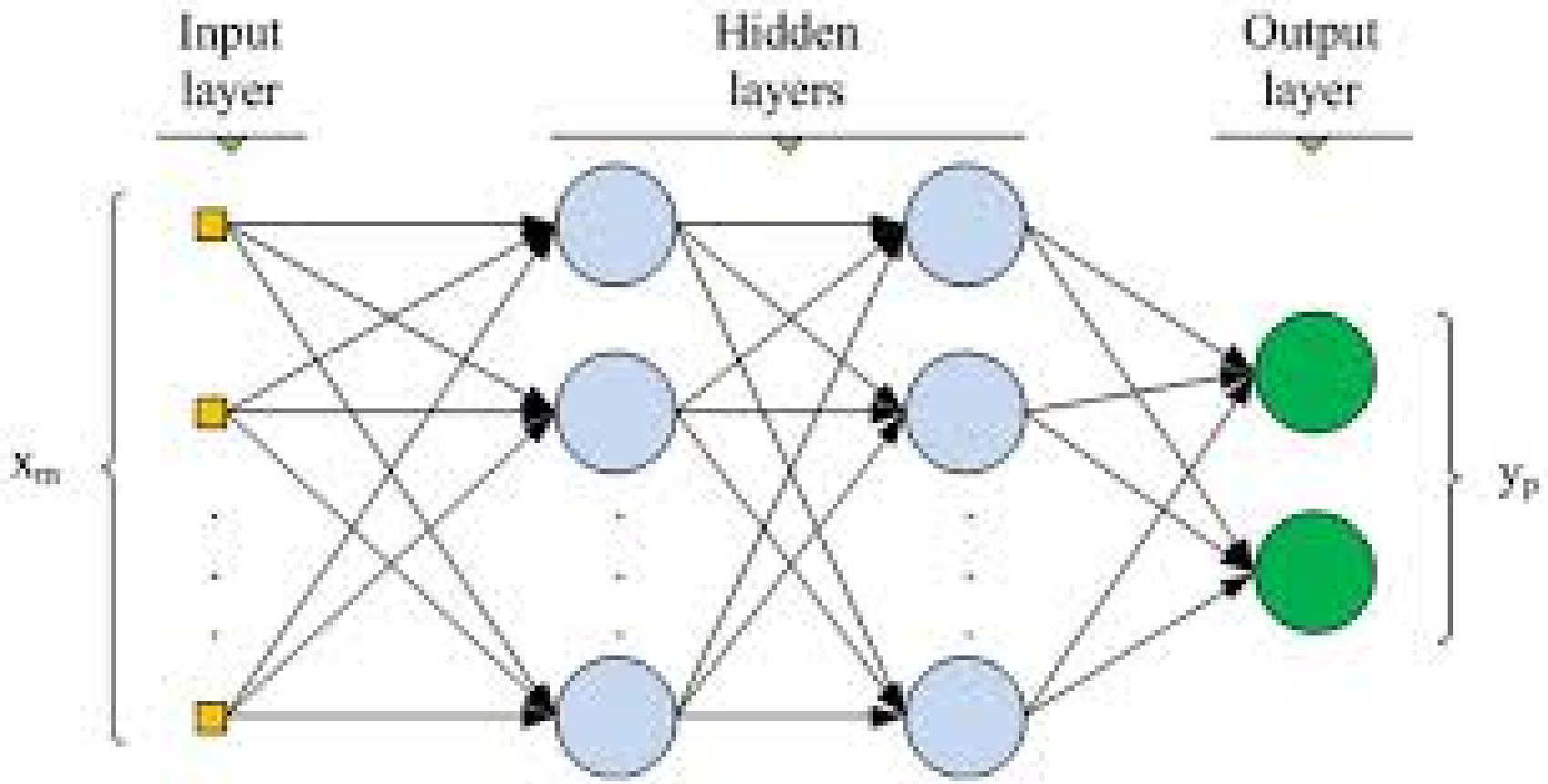




Neurone biologique

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

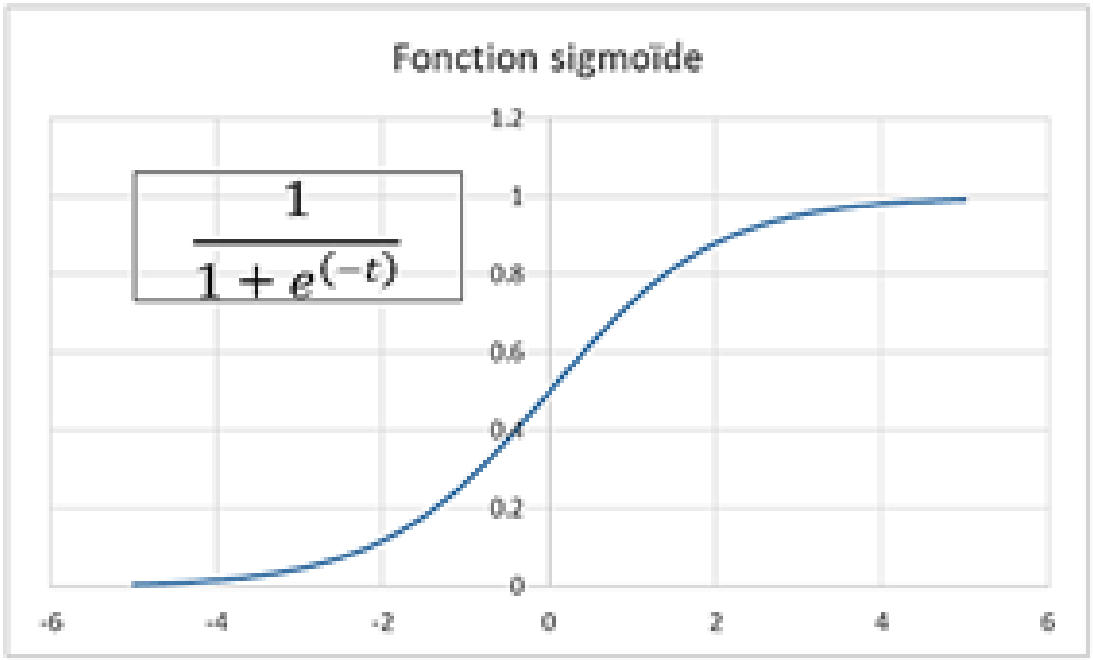
- activation function (e.g. sign)
- weight vector and bias (intercept)

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

Choose g differentiable preferably (cf gradient optimization techniques)

Activation function for the Artificial Neuron

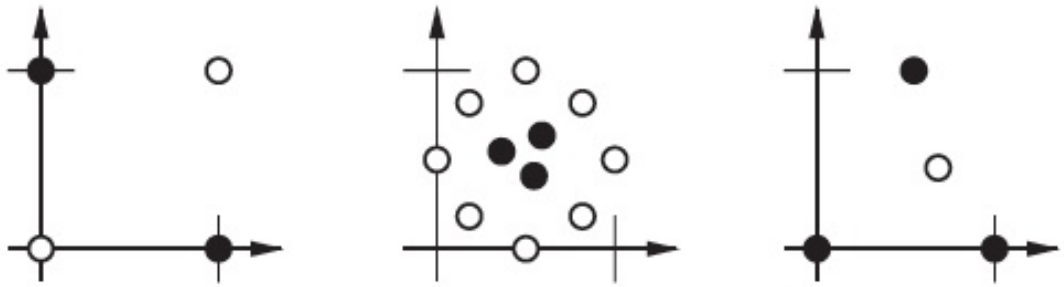
For instance :



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.

Ordinary Gradient Descent

Let $\mathcal{C}(\theta)$ be a function :

- The values θ such that $\frac{\partial C(\theta)}{\partial \theta} = 0$ correspond to minima or maxima of C.
- When $\mathcal{C}$ is strictly convex in θ, gradient descent permits to build iteratively a sequence of values converging to the (unique) solution.
- In addition, even if $\mathcal{C}$ is not strictly convex, it can be used to find a 'good' local minimum approximately.

Idea : refine the value θ iteratively by : $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial \mathcal{C}(\theta)}{\partial \theta}$
After each update, the gradient is re-evaluated at the new point/value and used next to refine the value using the same formula

Global gradient descent

$\mathcal{C}(\theta) = \sum_{n=1}^{N} c_n(\theta)$

1. E = 1000 ;
2. ε= small
3. θ^0 initial value ; $t = 0$;
4. While $(E > \varepsilon)$
   - $\theta^{t+1} \leftarrow \theta^t - \eta_t \sum_{n=1}^{N} \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
   - compute $E = L(\theta^{t+1})$
5. Output the current value of θ

Choosing η_k

Theorem :
If the series $(\sum_k \eta_k)$ diverges and if $(\sum_k \eta_k^2)$ converges, the gradient descent converges to a local minimum.

Local and stochastic gradient descent

$\mathcal{C}(\theta) = \sum_{n=1}^{N} c_n(\theta)$

1. $E = 1000$;
2. ε= small value
3. θ^0 initial value ; $t = 0$;
4. $nb_{cycle} = 0$
5. While $(E \geq \varepsilon)$ and $(nb_{cycle} < 500)$
   - $nb_{cycle} = nb_{cycle} + 1$
   - for $\ell = 1$ to N
     - Draw uniformly at random an index $n \in \{1, \ldots, N\}$
     - $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
   - compute $E = L(\theta^{t+1})$

Stochastic gradient descent with constant minibatch size

$\mathcal{C}(\theta) = \sum_{n=1}^{N} c_n(\theta)$

1. $E = 1000$;
2. ε= small value
3. θ^0 initial value ; $t = 0$;
4. $nb_{cycle} = 0$
5. While $(E \geq \varepsilon)$ and $(nb_{cycle} < 500)$
   - $nb_{cycle} = nb_{cycle} + 1$
     - Draw uniformly at random M times an index $n \in \{1, \ldots, N\}$
     - $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_M(\theta^t)}{\partial \theta^t}$
   - compute $E = L(\theta^{t+1})$

Gradient backpropagation (1/4)

Apply gradient descent to the weights of layers 1 and 2 (reduced here to a single output unit).
Let $\ell = \frac{1}{2}(h(x) - y)^2$
Calculation for a single data point , local algorithm
Gradient w.r.t. output weights :

$$\frac{\partial \ell}{\partial w_j^{(2)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_j^{(2)}} \tag{10}$$

Gradient w.r.t. the hidden layer :

$$\frac{\partial \ell}{\partial w_{ji}^{(1)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_{ji}^{(1)}} \tag{11}$$

Local backpropagation (2/4) : calculations

Calculation for a single data point , local algorithm
Gradient w.r.t. output weights :

$$\frac{\partial \ell}{\partial w_j^{(2)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_j^{(2)}} \tag{12}$$

$$\frac{\partial \ell}{\partial h(x)} = h(x) - y \tag{13}$$

$$\frac{\partial h(x)}{\partial w_j^{(2)}} = \frac{\partial g(w_j^{(2)} z_j + \sum_{k \neq j} w_k^{(2)} z_k)}{\partial w_j^{(2)}} \tag{14}$$

(15)

Local backpropagation (3/4) : calculations

Calculation for a single data point , local algorithm
Gradient w.r.t. output weights :

$$\frac{\partial \ell}{\partial w_{ji}^{(1)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_{ji}^{(1)}} \tag{16}$$

$$\frac{\partial h(x)}{\partial w_{ji}^{(1)}} = w_j^{(2)} \frac{\partial g(\sum_k w_{jk} x_k)}{\partial w_{ji}^{(1)}} \tag{17}$$

$$\frac{\partial h(x)}{\partial w_{ji}^{(1)}} = (1 - g(\sum_k w_{jk} x_k)^2) w_j^{(2)} x_i \tag{18}$$

Local backpropagation (4/4) : calculations

Local descent at x_n drawn uniformly at random :

1. At x_n, compute $h(x_n)$
2. Compute the gradients : $\frac{\partial \ell_n}{\partial w_j^{(2),t}}$ puis $\frac{\partial \ell_n}{\partial w_{ji}^{(1),t}}$
3. Correct the weights by means of the pre-calculated gradients :
   - Correct the layer (1) :
   - For $j = 0$ to M :
     - $w_j^{(1),t+1} \leftarrow w_j^{(1),t} - \eta_t \frac{\partial \ell_n}{\partial w_j^{(\mathbf{1}),t}}$
   - Correct layer 2 : here single output neuron
     - $w^{(2),t+1} \leftarrow w^{(2),t} - \eta_t \frac{\partial \ell_n}{\partial w^{(\mathbf{2}),t}}$

Regularization and early stopping

- **Early Stopping**
  - A first regularisation heuristic has been proposed in the 90's : stop *a priori* early the learning procedure to avoid overfitting : avoid getting close to a minimum !
- **Regularization**
  - Define :
    $\mathcal{L}(W, \mathcal{S}_{app} = \sum_n \ell(h(x_n), y_n) + \lambda_2 ||w^{(2),*}||^2 + \lambda_1 ||w^{(1),*}||^2$
  - Do not regularise $w_0^{(2)}$, $w_{j0}^{(1)}$ and $w_{0i}^{(2)}$. These components are not considered by the regularization scheme.

In pratice, note that : $||w^{(1),*}||^2 = \sum_{ji, j\neq 0, i\neq 0} (w_{ji}^{(1)})^2$.

Model Selection

The MLP has several hyperparameters :

- Nb of hidden layers
- Sizes of the hidden layers
- parameter λ
- nb_{cycle}, ε
- $\eta_t = \frac{\gamma}{1+t}$

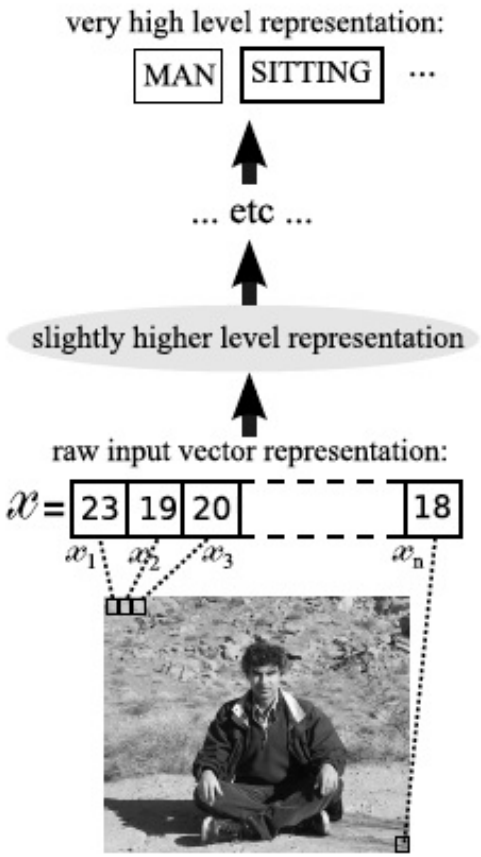
In general, they are selected by means of CROSS VALIDATION.

Advantages and drawbacks of Neural Networks "feedforward"

- **Pros**
  - Flexibility regarding the output : arbitrary number of classes, etc..
  - Fitting methos known since the mid 80's
  - Stochastic gradient descent is appropriate to deal with BIG DATA
  - GPU architectures can be used
  - PLUG and PLAY : the same paradigm can be used successively
- **Cons**
  - Non convex loss : no global minimum
  - Implementing the gradient descent requires many adjustments in practice
  - No theoretical validity framework
  - *Ad hoc* implementations - The 'art' of neural nets

Deep Nets - Deep Learning

Image Y. Bengio



Learning deep neural networks

From 3 layers, one uses the term "deep learning", this type of network is useful to analyze complex data such as textual data or images.

The why of the use of several hidden layers ?

The fact that a NN with a single hidden layer is a universal approximant does not mean that it provides the best representation/approximation.

Learning deep neural networks

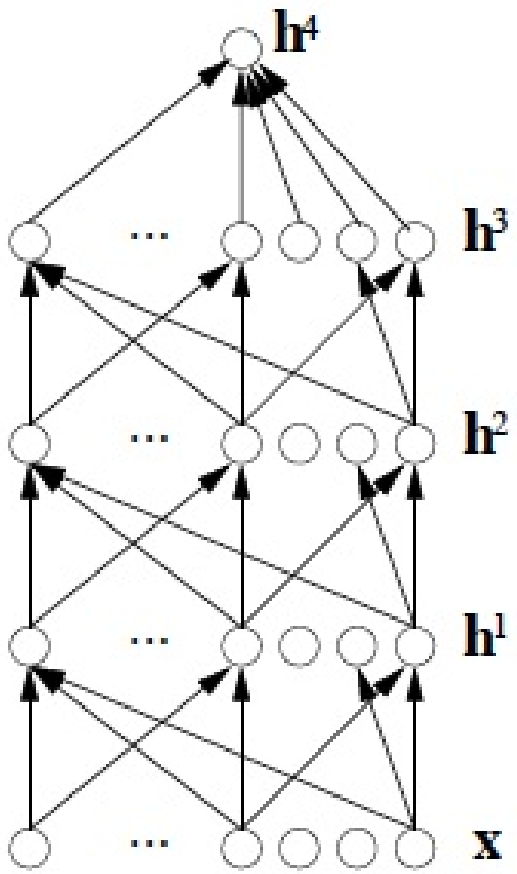
Despite the danger of overfitting,

two good reasons for considering deep neural nets

- advances in memory and computation (GPU)
- availability of massive databases (Imagenet, Fei-Fei, 2008)

Gradient backpropagation does not work well for deep nets (Bengio et al. 2007 ; Erhan et al. 2009).
In absence of a good initialization, it often outputs bad local minima.

Learning deep neural networks



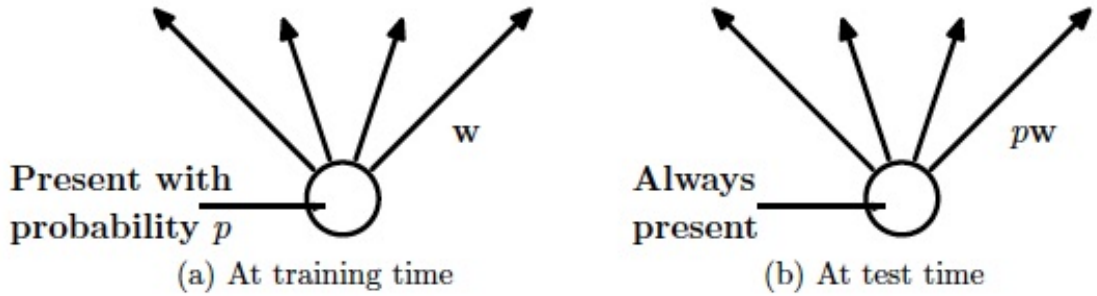
Learning deep neural networks

Two major improvements

- Dropout
- Auto-encoders

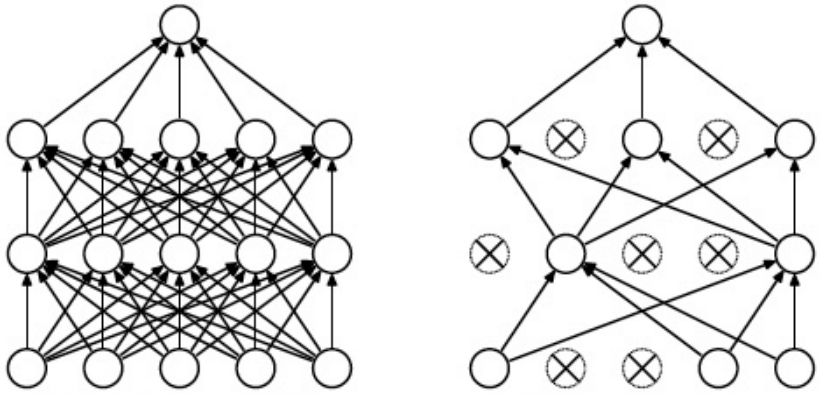
Avoid overfitting deep nets by *dropout 1/3*

For very deep nets (>>2 layers) :



- During the learning stage, at each gradient modification : each unit (neuron) is considered with probability p, meaning that certain neurons are not present and are consequently not corrected systematically.
- During the prediction (test stage), each unit is present with a factor p applied to its weights.

Avoid overfitting deep nets by *dropout 2/3*



(a) Standard Neural Net (b) After applying dropout.

Interpretation :

When m neurons are involved, it is like one learns 2^m sparse neural nets and during the test, they are aggregated to form the neural network used for prediction.

Neurons cannot adjust w.r.t. the others

Avoid overfitting deep nets by *dropout 3/3*

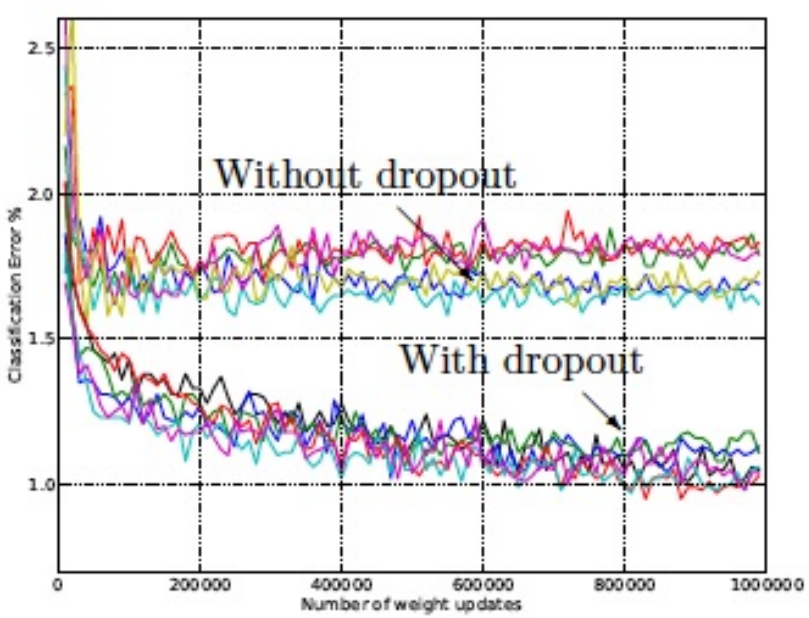


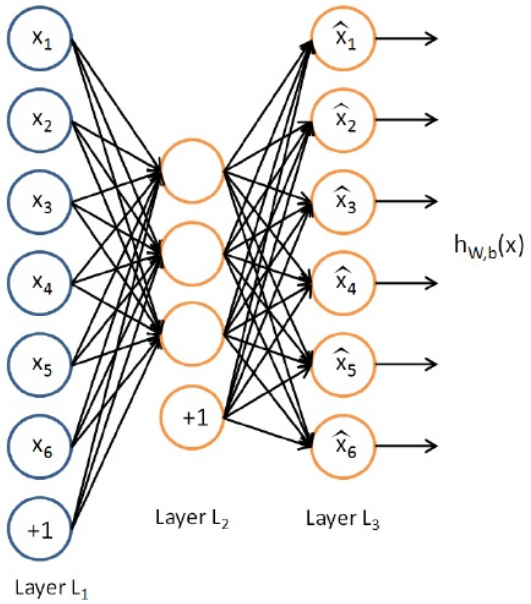
Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Deep Learning

Deep nets are often initialized by means of unsupervised learning, through autoencoders or Restricted Boltzman Machines (RBM). We will see how later...

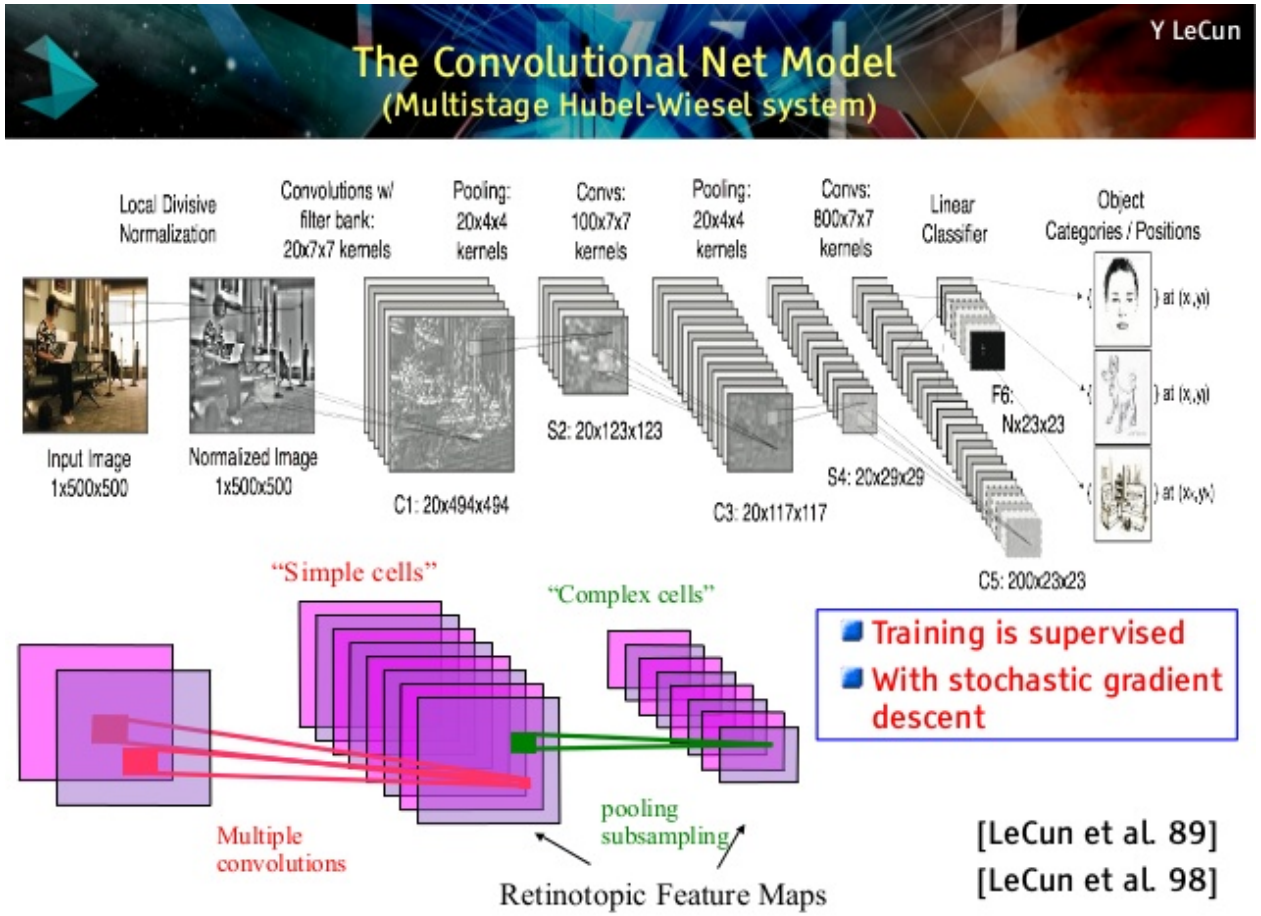
Autoencoders

An autoencoder is a network with one input layer, one or more hidden layers and one output layer. This type of network aims at providing an internal representation (the layer in the middle) by learning how to predict the input from itself : $x \approx g(x)$.

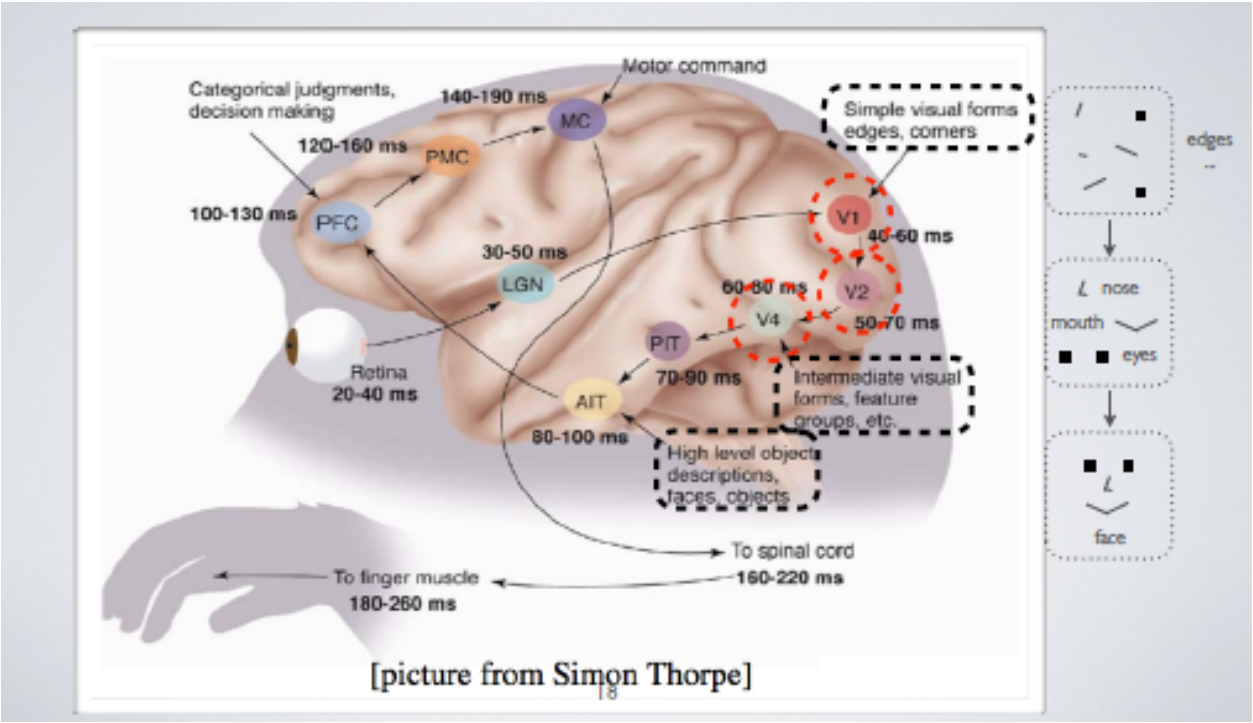


Convolutional Neural Networks for Images

Y. Le Cun.



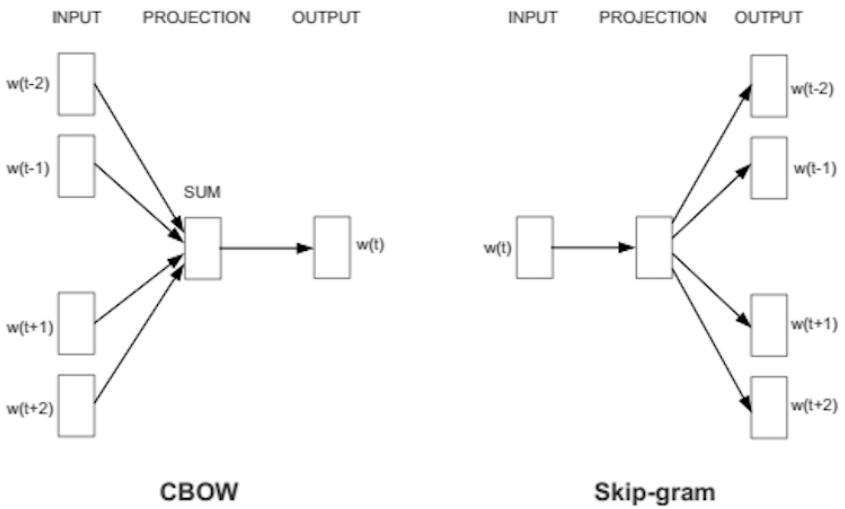
Visual cortex



[picture from Simon Thorpe]

Skip-gram model for word embedding

Mikolov et al. 2013.



References Ensemble Learning

- Y. Amit, D. Geman, and K. Wilder, Joint induction of shape features and tree classifiers, IEEE Trans. Pattern Anal. Mach. Intell., 19, 1300-1305, 1997.
- Breiman, L., Bagging predictors. Mach Learn (1996) 24 : 123.
- Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory, 1995.
- J. Friedman, T. Hastie and R. Tibshirani, Additive logistic regression : a statistical view of boosting. Ann. Statist. Vol. 28, No. 2 (2000), 337-407.
- Breiman, L., Random Forests. Mach. Learn. (2001), Vol. 45, No. 1, pp 5–32
- Tutorial : Ensemble Methods in Machine Learning. T.G. Dietterich, available at :
  http ://web.engr.oregonstate.edu/ tgd/publications/mcs-ensembles.pdf

References - Réseaux de Neurones

- Le cours de Hugo Larochelle (youtube)
- Notes de cours IT6266, Université de Montréal, Equipe de Yoshua Bengio.
- Learning Deep Architectures for AI, Yoshua Bengio, Foundations Trends in Machine Learning, 2009
- Dropout : A simple way to prevent overfitting, Srivastava et al. JMLR 2014
- Pattern Recognition and Machine Learning, C. Bishop, Springer, 2006.
- `http://deeplearning.net/tutorial/` : pour tout document y compris implémentations...

References - Neural Networks

- The online course of Hugo Larochelle (youtube)
- Lecture notes IT6266, Université de Montréal, Equipe de Yoshua Bengio.
- Learning Deep Architectures for AI, Yoshua Bengio, Foundations Trends in Machine Learning, 2009
- Dropout : A simple way to prevent overfitting, Srivastava et al. JMLR 2014
- Pattern Recognition and Machine Learning, C. Bishop, Springer, 2006.
- `http://deeplearning.net/tutorial/` : pour tout document y compris implémentations...

Ensemble Learning

—

Bagging, Boosting and Random Forests

Agenda

- Ensemble Learning - Consensus
- Bagging - Increase stability
- Boosting - "Best-off-the-shelf"
- "Random Forests"

Consensus methods

- Rather than fitting a classifier, combine the predictions of an **ensemble** of classifiers

$$C_1(X),\ \ldots,\ C_M(X).$$

Amit & Geman (1997)

- Majority vote :

$$sign\left(\sum_{m=1}^{M} C_m(X)\right)$$

- Variant - weighted majority vote : $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$

$$sign\left(\sum_{m=1}^{M} \alpha_m C_m(X)\right)$$

- Extension to multiclass, to regression
- An old challenge : "ranking" and consensus (preference data)

Bagging - Boostrap

- **Bootstrap aggregating** technique - Breiman (1996)
- Applicable to any learning algorithm $\mathcal{L}$
- From the training dataset $\mathcal{D}_n$:
  1. Generate independently $B \geq 1$ bootstrap samples $\mathcal{D}_n^{*(b)}$ (sampling with replacement in $\mathcal{D}_n$)
  2. For $b : 1$ to B, implement algorithm $\mathcal{L}$ based on $\mathcal{D}_n^{*(b)}$, producing classifier $C^{*(b)}$
  3. Aggregate the bootstrap predictions by calculating the majority vote :

$$C_{bag}(X) = sign\left(\sum_{b=1}^{B} C^{*(b)}(X)\right)$$

- Variant : if $C^{*(b)}(X) = sign(f^{*(b)}(X))$,

$$\widetilde{C}_{bag} = sign\left(\sum_{b=1}^{B} f^{*(b)}(X)\right)$$

Bagging - Comments

- Bagging can **significatively reduce the variability** of unstable algorithms (e.g. decision trees)
- Variance reduction may lead to a lower prediction error
- In regression : $f_{bag}(x) = \mathbb{E}[f^*(x)]$ (expectation w.r.t. $\mathcal{D}_n$)

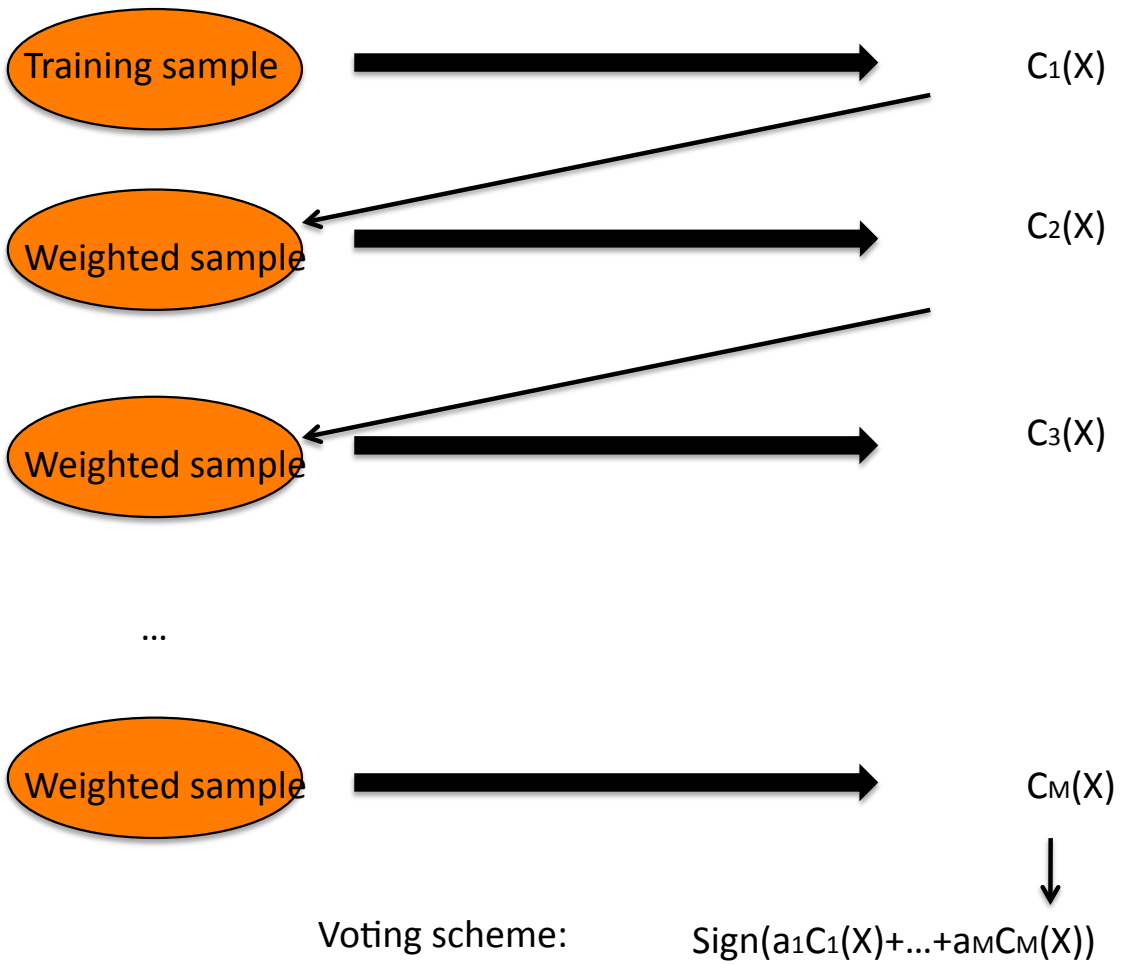
$$\mathbb{E}\left[(Y - f^*(x))^2\right] = \mathbb{E}\left[(Y - f_{bag}(x))^2\right] + \mathbb{E}\left[(f_{bag}(x) - f^*(x))^2\right] \geq \mathbb{E}\left[(Y - f_{bag}(x))^2\right]$$

- In classification :
  Bagging a good classifier improves it, but ...
  bagging a bad one may deteriorate it !

Boosting

- AdaBoost - Freund & Schapire (1995)
- The ingredient for 'slow learning', resisting to the overfitting phenomenon : a "weak" classification method $\mathcal{L}$
- Heuristic :
  - apply $\mathcal{L}$ to weighted versions of the original training dataset
  - increase the weights of the observations misclassified by the current predictive rule
  - aggregate the classifiers in a non uniform fashion (a good predictor should not be built from a few outliers)
- AdaBoost surpasses its competitors when applied to many benchmark datasets
- Statistical interpretation : five years later...

Boosting - General Scheme



The algorithm "Adaptive Boosting"

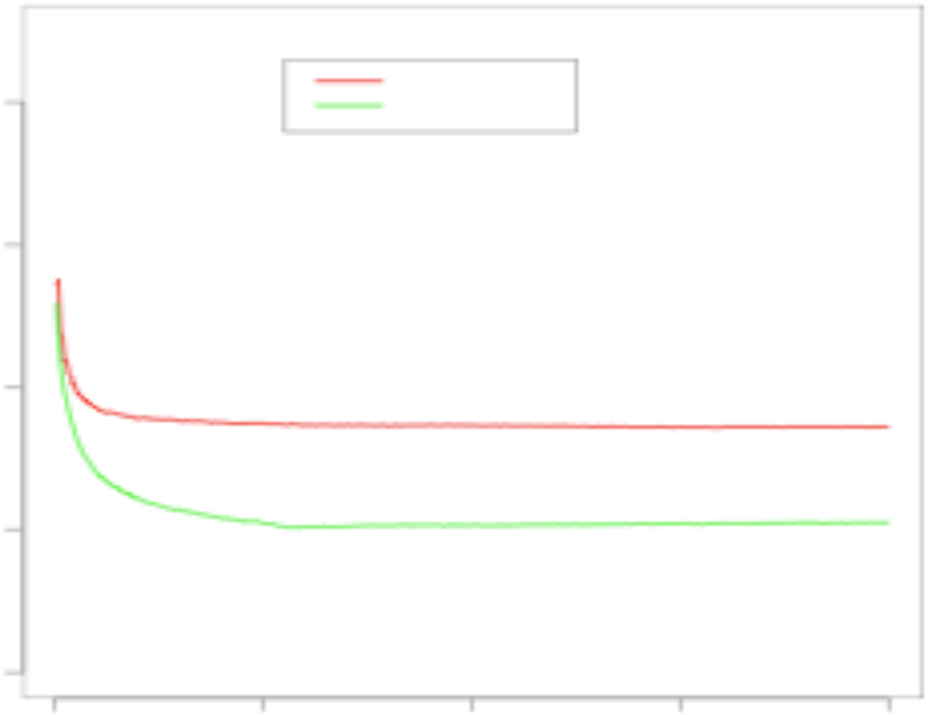
- Initialization : uniform weights, $\omega_i = 1/n$ assigned to each example (X_i, Y_i), $1 \leq i \leq n$
- From $m : 1$ to M,
  1. By means of algorithm $\mathcal{L}$, fit a weak classifier C_m based on the weighted labelled sample $\{(X_i, Y_i, \omega_i) : \ 1 \leq i \leq n\}$
  2. Compute the weighted prediction error rate

$$err_m = \sum_{i=1}^{n} \omega_i \mathbb{I}\{Y_i \neq C_m(X_i)\}$$

     and $a_m = \log((1 - err_m)/err_m)$
  3. Update the weights :
     - $\omega_i \leftarrow \omega_i \exp\left(a_m \mathbb{I}\{Y_i \neq C(X_i)\}\right)$
     - $\omega_i \leftarrow \omega_i / \sum_{j=1}^{n} \omega_j$
- Output : $C_{Boost}(X) = sign\left(\sum_{m=1}^{M} a_m C_m(X)\right)$

AdaBoost resists the overfitting phenomenon !

- Typical weak classifiers : stumps (binary tree with depth 1)
- As M increases, the test error decreases and stabilizes



Practical Aspects

- How to implement $\mathcal{L}$ based on a **weighted** sample ?
  - modify the criterion to be optimized explicitly (ex : CART, SVM, k-NN, *etc.*)
  - draw at random a training sample with distribution $\sum_i \omega_i \delta_{(X_i, Y_i)}$
- When should we stop the iterations ?
  - plot the test error as a function of M
  - stop when it stabilises (but it is no more a test error...)

A statistical interpretation of Boosting

- Friedman, Hastie & Tibshirani (2000)
- Stagewise forward additive modelling
- Exponential loss : $C(X) = sign(f(X))$

$$L_e(f) = \mathbb{E}[\exp(-Yf(X))]$$

- Optimal solution :

$$f^*(X) = \frac{1}{2} \log \left(\frac{\eta(X)}{1 - \eta(X)} \right)$$

Forward stagewise additive modelling

- Heuristic : refine the current predictive rule $f_{m-1}(x)$ by adding $\alpha_m C_m(x)$, with $\alpha_m \in \mathbb{R}$ and $C_m(x) \in \{-1, +1\}$
- How to choose α_m and $C_m(x)$ so as to minimise the exponential version of the empirical risk ?

$$\underset{\alpha,\, C}{\arg\min} \sum_{i=1}^{n} \exp\left(-Y_i(f_{m-1}(X_i) + \alpha C(X_i))\right) = ?$$

- Set $\omega_i = \exp(-Y_i f_{m-1}(X_i))$, the empirical risk then writes :

$$\sum_{i=1}^{n} \omega_i \exp\left(-Y_i \alpha C(X_i)\right)$$

- For any $\alpha > 0$, the classifier with minimum risk is also that which minimises the weighted risk :

$$\sum_{i=1}^{n} \omega_i \mathbb{I}\{Y_i \neq C(X_i)\}$$

Forward stagewise additive modelling

- Let $C_m(X)$ be the solution to this weighted classification problem :

$$err_m = \sum_{i=1}^{n} \omega_i \mathbb{I}\{Y_i \neq C_m(X_i)\}$$

- It remains to minimize in α :

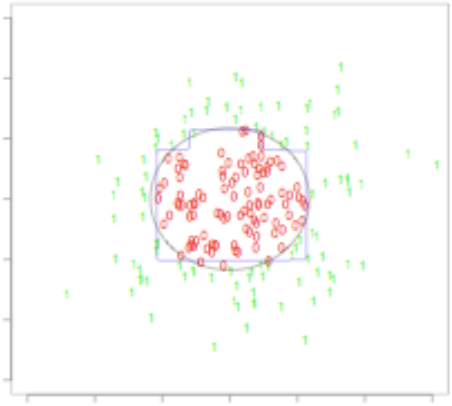
$$e^{\alpha} err_m + e^{-\alpha}(1 - err_m),$$

and get $\alpha_m = (1/2) \cdot \log((1 - err_m)/err_m)$

- Many variants : other losses, weight thresholding, *etc.*

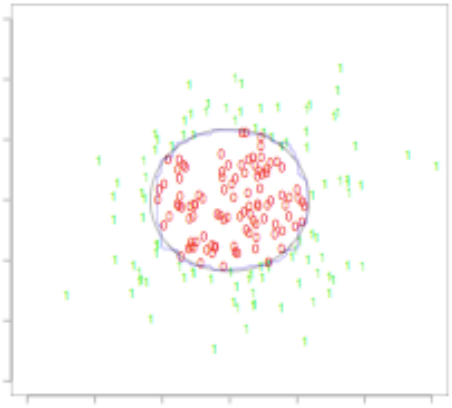
Aggregation produces smooth decision regions

Decision Boundary: Tree



When the nested spheres are in R^{10}, CART™ produces a rather noisy and inaccurate rule $\hat{C}(X)$, with error rates around 40%.

Decision Boundary: Boosting



Bagging and Boosting average many trees, and produce smoother decision boundaries.

Random Forest

- Ingredients : bagging + randomization
- Randomize the collection of predictive variables (*i.e.* X's components) : before splitting the nodes of a bootstrap decision tree
- Typical weak classifier : decision tree with small depth
- Aggregation preserves consistency...
  but no theoretical explanation for the observed performance !
- Heuristic : randomization (of the predictive variables) "enriches" the rule
- Randomization of the data when massive, to scale up the algorithms

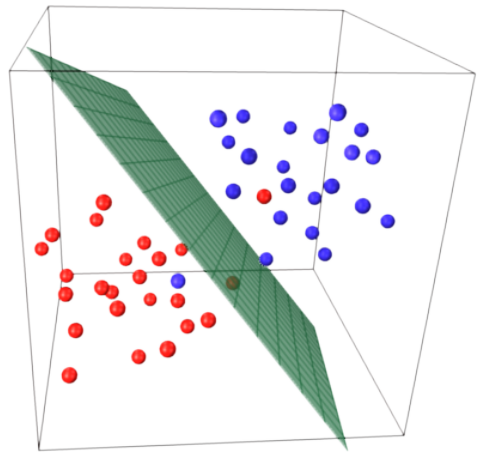
Linear separation

Definition

Let $\mathbf{x} \in \mathbb{R}^p$

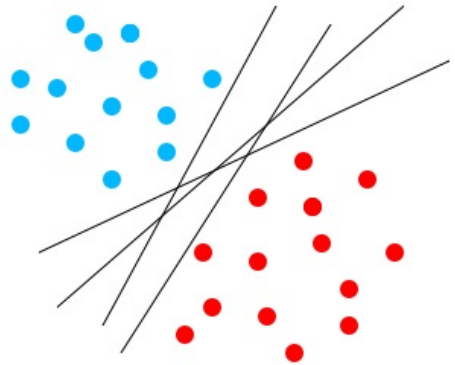
$f(\mathbf{x}) = \text{signe}(\mathbf{w}^T\mathbf{x} + b)$

The equation : $\mathbf{w}^T\mathbf{x} + b = 0$ defines an hyperplane in the Euclidean space $\mathbb{R}^p$



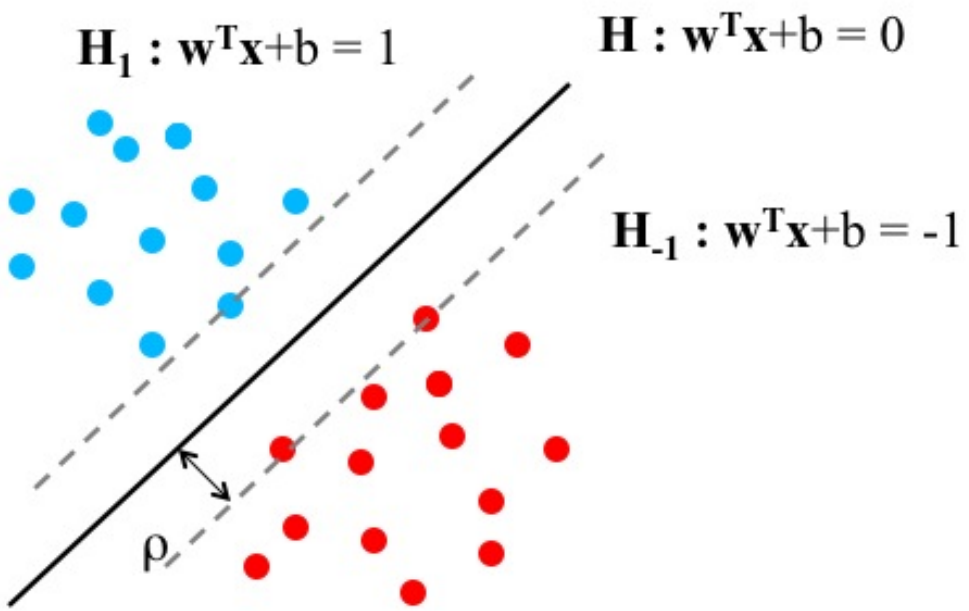
Example : training data in 3D and linear separator

Case of data linearly separable



Example in 2D : what line should be chosen ?

Margin criterion



Margin criterion

Geometrical notion of margin

- To separate the data, consider a triplet of hyperplanes :
  - H : $\mathbf{w}^T\mathbf{x} + b = 0$, $H_1 : \mathbf{w}^T\mathbf{x} + b = 1$, $H_{-1} : \mathbf{w}^T\mathbf{x} + b = -1$
- The *geometrical margin*, $\rho(\mathbf{w})$ is the smallest distance between the data and the hyperplane H, here half of the distance between H_1 and H_{-1}
- A simple calculation yields : $\rho(\mathbf{w}) = \frac{1}{||\mathbf{w}||}$.

New cost function to be optimized

How to determine $\mathbf{w}$ and b ?

- Maximize the margin $\rho(\mathbf{w})$ while separating the data on both sides of H_1 and H_{-1}
- Separate the blue data $(y_i = 1) : \mathbf{w}^T \mathbf{x}_i + b \geq 1$
- Separate the red data $(y_i = -1) : \mathbf{w}^T \mathbf{x}_i + b \leq -1$

Linear SVM : separable case

Optimisation in the primal space

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimiser}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{sous la contrainte} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1, \ i = 1, \ldots, n. \end{aligned}$$

Reference

Boser, B. E. ; Guyon, I. M. ; Vapnik, V. N. (1992). "A training algorithm for optimal margin classifiers". Proceedings of the fifth annual workshop on Computational learning theory - COLT '92. p. 144.

Quadratic optimization program with linear constraints

Typical problem (notations are changing !)

- un problème d'optimisation $(\mathcal{P})$ est défini par

$$\begin{array}{rl} \text{minimiser sur } \mathbb{R}^n & J(\mathbf{x}) \\ \text{avec} & h_i(\mathbf{x}) = 0\,, 1 \leq i \leq p \\ & g_j(\mathbf{x}) \leq 0\,, 1 \leq j \leq q \end{array}$$

- rappel de vocabulaire :
  - les h_i sont les **contraintes d'égalité** (notées $\mathbf{h}(\mathbf{x}) = 0$)
  - les g_j sont les **contraintes d'inégalité** (notées $\mathbf{g}(\mathbf{x}) \leq 0$)
  - l'**ensemble des contraintes** est

$$\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^n | h_i(\mathbf{x}) = 0\,, 1 \leq i \leq p \text{ et } g_j(\mathbf{x}) \leq 0\,, 1 \leq j \leq q\}$$

ensemble des points admissibles ou **réalisables**

Quadratic optimization program with linear constraints

Typical problem :
$\min_x f(x)$
s.c. $g(x) \leq 0$

- Her, $g(x)$ linear
- f strictly convex

1. Lagrangian : $J(x, \lambda) = f(x) + \lambda g(x)$, $\lambda \geq 0$

Quadratic optimization program with linear constraints

Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}||\mathbf{w}||^2 + \sum_i \alpha_i(1 - y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}))$$

$$\forall i, \alpha_i \geq 0$$

Karush-Kuhn-Tucker conditions

At the extremum, one has

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0$$

$$\nabla_b \mathcal{L}(b) = -\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\forall i, \alpha_i \geq 0$$

$$\forall i, \alpha_i [1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)] = 0$$

Obtaining the α_i's : solving in the dual space

$$\mathcal{L}(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

- Maximize $\mathcal{L}$ under the constraints $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0, \forall i = 1, \ldots, n$
- Use a quadratic solver

Linear SVM or Optimal Margin Hyperplane

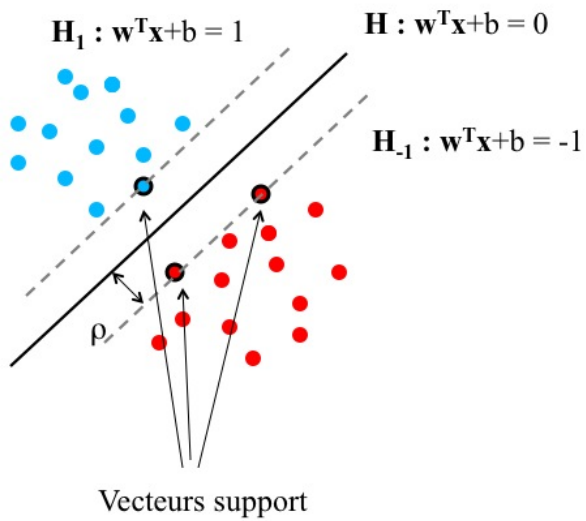
Suppose that the Lagrange multipliers α_i have been determined :

Equation of a linear SVM

$$f(\mathbf{x}) = \text{signe}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b)$$

To predict the label a point$\mathbf{x}$, the classifier combines linearly the labels y_i of the support points with weights of type $\alpha_i \mathbf{x}_i^T \mathbf{x}$ depending on the **similarity** between $\mathbf{x}$ and the support data in the inner product (cosine similarity).

"Support" Vectors



The training data points $\mathbf{x}_i$ such that $\alpha_i \neq 0$ are on one hyperplane or the other, H_1 or H_{-1}.
Only these data points, referred to as *support vectors* are explicitely involved in the definition of $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$
NB : b is obtained by choosing a support vector ($\alpha_i \neq 0$)

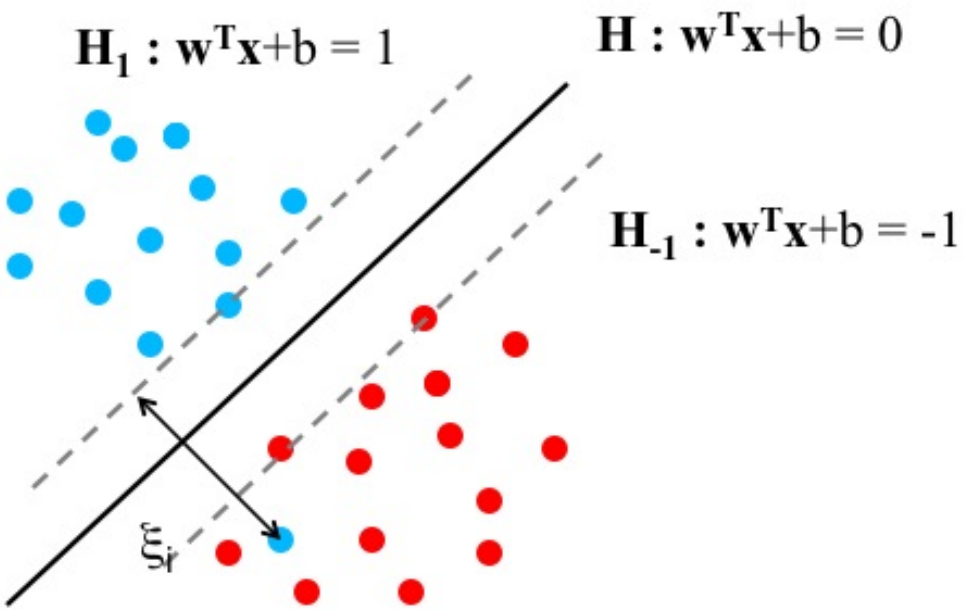
Realistic case : linear SVM for non linearly separable data

Introduce a slack variable ξ_i for each data point :

Problem in the primal space

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{sous les contraintes} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1 - \xi_i \; i = 1, \ldots, n. \\ & \xi_i \geq 0 \; i = 1, \ldots, n. \end{aligned}$$

Realistic case : linear SVM for non linearly separable data



Realistic case : linear SVM for non linearly separable data

Problem in the dual space

$$\max_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{sous les contraintes} \quad 0 \leq \alpha_i \leq C \; i = 1, \ldots, n.$$

$$\sum_i \alpha_i y_i \; i = 1, \ldots, n.$$

Karush-Kuhn-Tucker (KKT) conditions

Let α^* be the solution of the dual problem :

$$\forall i, [y_i f_{w^*,b^*}(x_i) - 1 + \xi_i^*] \leq 0 \tag{19}$$

$$\forall i, \alpha_i^* \geq 0 \tag{20}$$

$$\forall i, \alpha_i^* [y_i f_{w^*,b^*}(x_i) - 1 + \xi_i^*] = 0 \tag{21}$$

$$\forall i, \mu_i^* \geq 0 \tag{22}$$

$$\forall i, \mu_i^* \xi_i^* = 0 \tag{23}$$

$$\forall i, \alpha_i^* + \mu_i^* = C \tag{24}$$

$$\forall i, \xi_i^* \geq 0 \tag{25}$$

$$\mathbf{w}^* = \sum_i \alpha_i^* y_i \mathbf{x}_i \tag{26}$$

$$\sum_i \alpha_i^* y_i = 0 \tag{27}$$

(28)

Different situations

Let α^* be the solution of the dual problem :

- if $\alpha_i^* = 0$, then $\mu_i^* = C > 0$ and thus, $\xi_i^* = 0$: x_i is well classified
- if $0 < \alpha_i^* < C$, then $\mu_i^* > 0$ and thus, $\xi_i^* = 0$: x_i is such that : $y_i f(x_i) = 1$
- if $\alpha_i^* = C$, then $\mu_i^* = 0$, $\xi_i^* = 1 - y_i f_{w^*,b^*}(x_i)$

NB : one computes b^* by choosing i such that $0 < \alpha_i^* < C$

Realistic case : linear SVM for non linearly separable data

A few remarks are in order

- certain support vectorsare thus on the 'wrong sides' of the hyperplanes H_1 or H_{-1}
- C is an hyperparameter that control the trade-off between model complexity (nb of support vectors here) and goodness-of-fit.

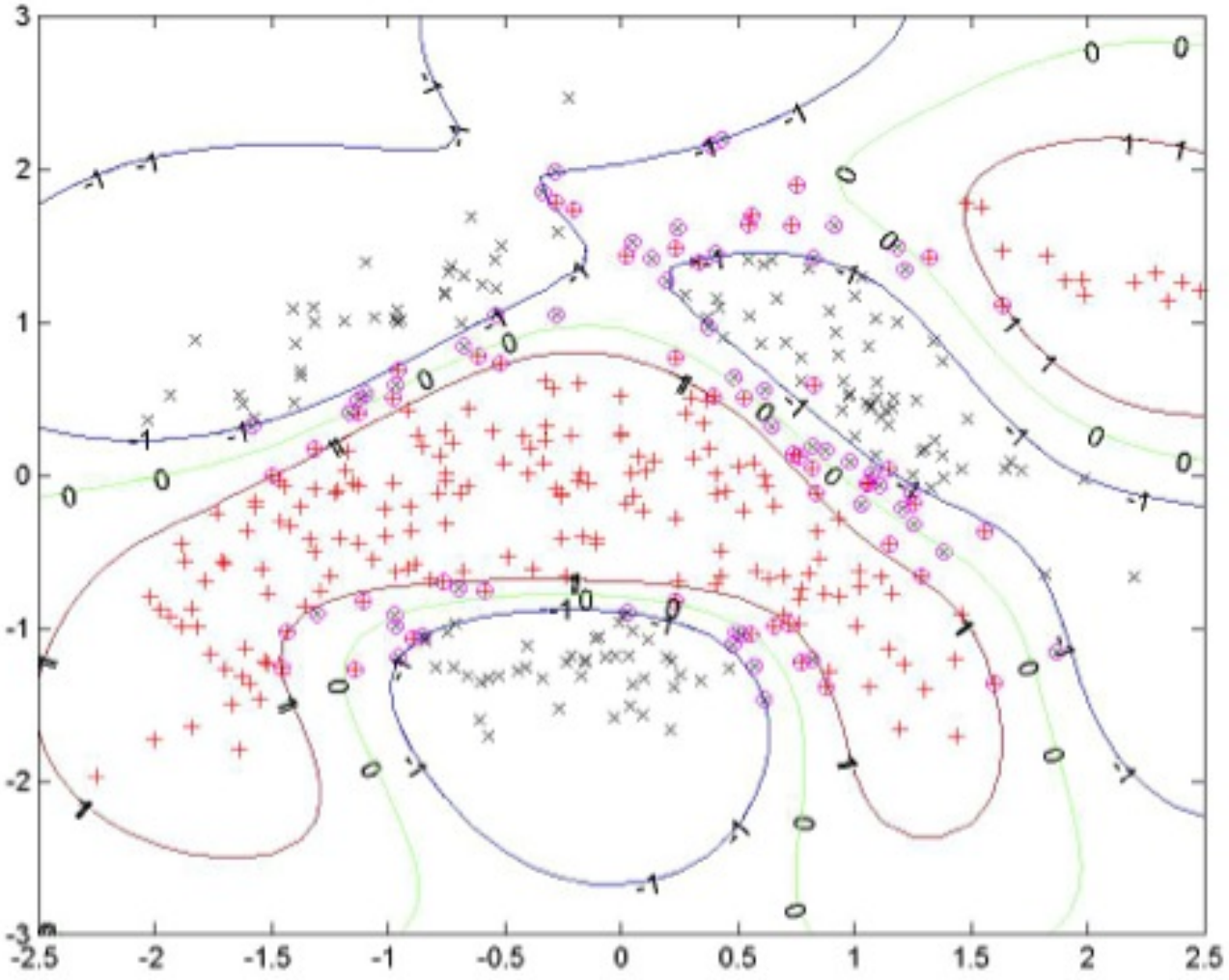
SVM : regularization approach

Optimization in the primal space

$$\min_{\mathbf{w},b} \quad \sum_{i=1}^{n}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))_+ + \lambda\frac{1}{2}\|\mathbf{w}\|^2$$

With : $(z)_+ = max(0, z)$
$f(\mathbf{x}) = \text{signe}(h(\mathbf{x}))$
Cost function : $L(\mathbf{x}, y, h(\mathbf{x})) = (1 - yh(\mathbf{x}))_+$
$yh(\mathbf{x})$ is called 'classifier's margin'

Support Vector Machine : the non linear case



Preliminary remark

The problem of finding the hyperplane with optimal margin involves the training data through inner products only.

$$\begin{aligned} \max_{\alpha} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{sous les contraintes} \quad & 0 \leq \alpha_i \leq C \; i = 1, \ldots, n. \\ & \sum_i \alpha_i y_i \; i = 1, \ldots, n. \end{aligned}$$

Remark 1 : learning/training

If the data are transformed by means of a function φ (non linear) and if the inner products $\varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$ can be computed, then a non linear separating function can be learned.

$$\max_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$$

$$\text{sous les contraintes} \quad 0 \leq \alpha_i \leq C \ i = 1, \ldots, n.$$

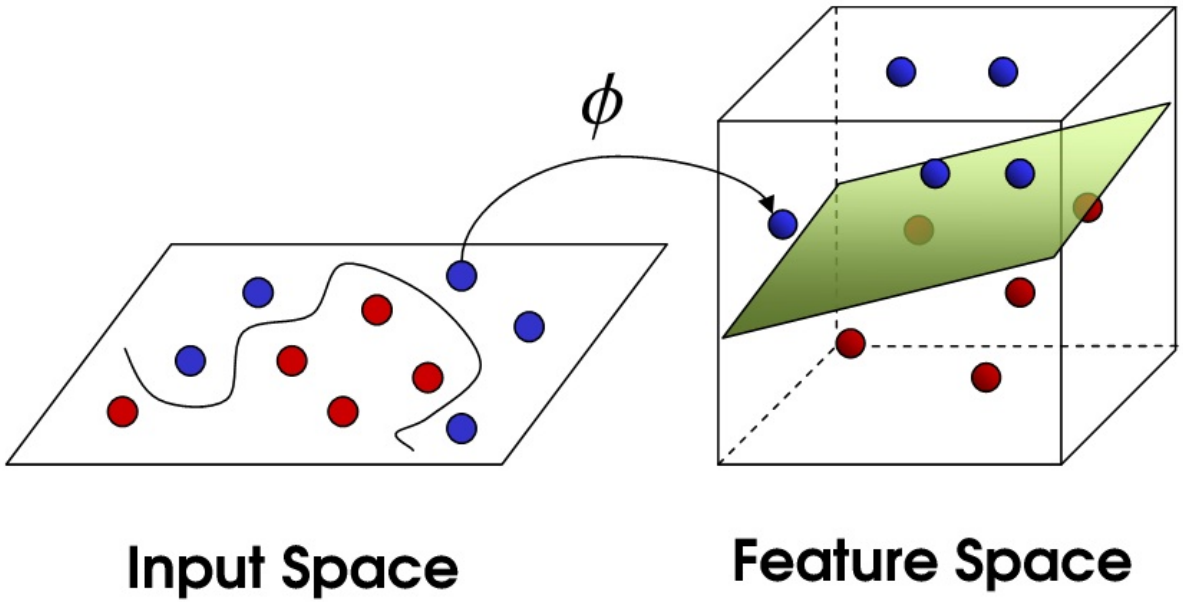
$$\sum_i \alpha_i y_i \ i = 1, \ldots, n.$$

To predict the lable of a new data point $\mathbf{x}$, only $\varphi(\mathbf{x})^T\varphi(\mathbf{x}_i)$ is required.

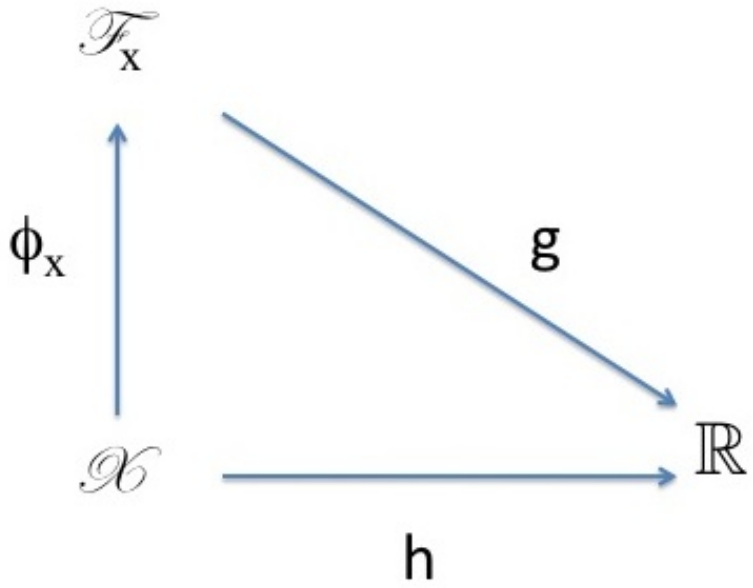
The kernel trick

If one replaces $\mathbf{x}_i^T \mathbf{x}_j$ by its image by a function k : $k(\mathbf{x}_i, \mathbf{x}_j)$ such that there exist a feature space $\mathcal{F}$ and a feature map $\varphi : \mathcal{X} \to \mathcal{F}$ and $\forall(\mathbf{x}, \mathbf{x}') \in \mathcal{X}, k(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T \varphi(\mathbf{x}')$, one may apply then the same optimization algorithm (solving in the dual) and obtain :
$f(\mathbf{x}) = \text{signe}(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b)$
Such functions do exist and are called *kernels*.

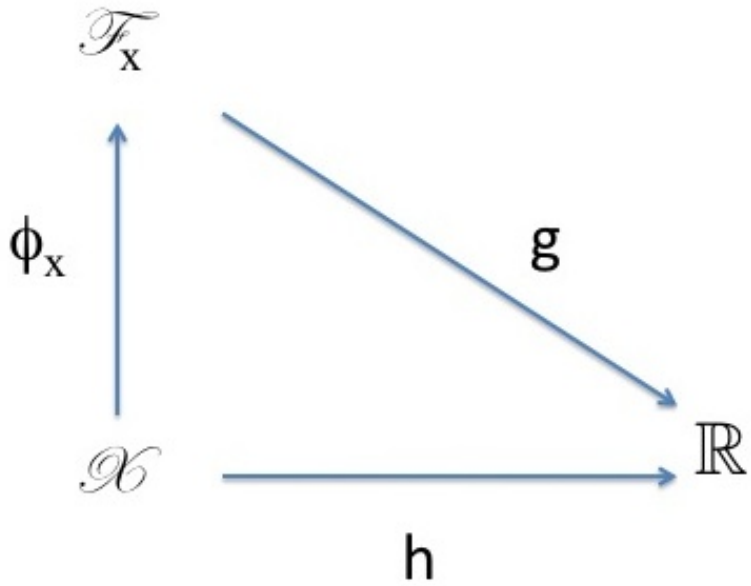
Kernel trick and feature map 1/2



Kernel trick and feature map 2/2



Kernel trick and feature map 2/2



Function h of the type :
$h(\mathbf{x}) = \sum_{i=1}^{n} \beta_i \varphi(x)^T \varphi(x_i) = \sum_{i=1}^{n} \beta_i k(x, x_i)$,
with $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a positive definite kernel.

Kernels

Definition

Let $\mathcal{X}$ be an ensemble. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, a symmetric function. The function k is called a positive kernel iff for any set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ in $\mathcal{X}$ and any column vector $\mathbf{c}$ in $\mathbb{R}^m$,
$\mathbf{c}^T K \mathbf{c} = \sum_{i,j=1}^{m} c_i c_j k(x_i, x_j) \geq 0$

Moore-Aronzajn's theorem

Moore-Aronzajn

Let K be a positive definite kernel. Then, there exists a unique Hilbert space $\mathcal{F}$ for which k is an auto-reproducing kernel :
$\forall x \in \mathcal{X}, \langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{F}} = f(x)$
In particular : $\langle k(\cdot, x), k(\cdot, x') \rangle_{\mathcal{F}} = k(x, x')$

Moore-Aronzajn's theorem

Moore-Aronzajn

Let K be a positive definite kernel. Then, there exists a unique Hilbert space $\mathcal{F}$ for which k is an auto-reproducing kernel :
$\forall x \in \mathcal{X}, \langle f(\cdot), k(\cdot, x)\rangle_{\mathcal{F}} = f(x)$
In particular : $\langle k(\cdot, x), k(\cdot, x')\rangle_{\mathcal{F}} = k(x, x')$

NB :*It means that one can always choose* $\varphi(x) = k(\cdot, x)$

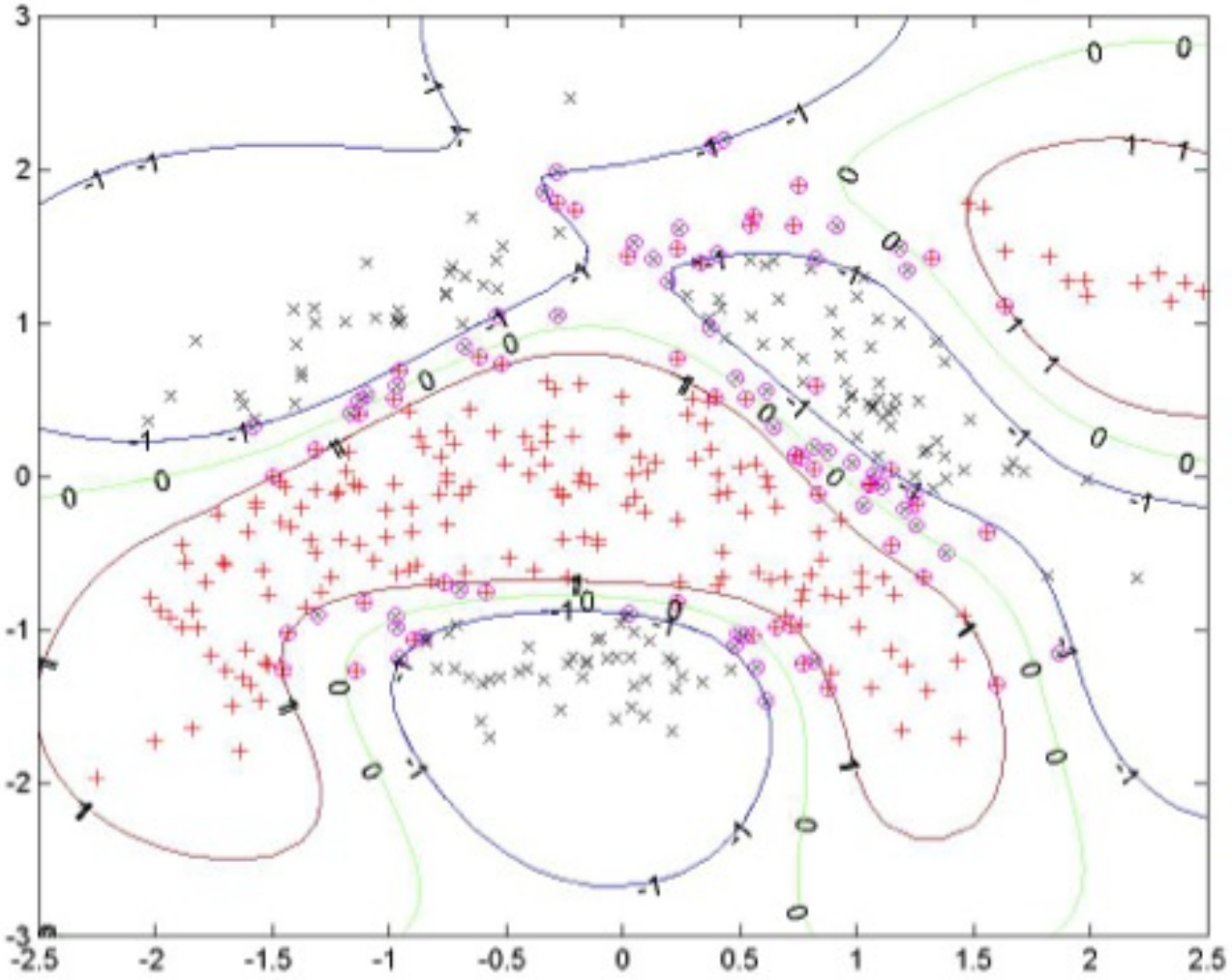
Kernels

Kernels between vectors

$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$

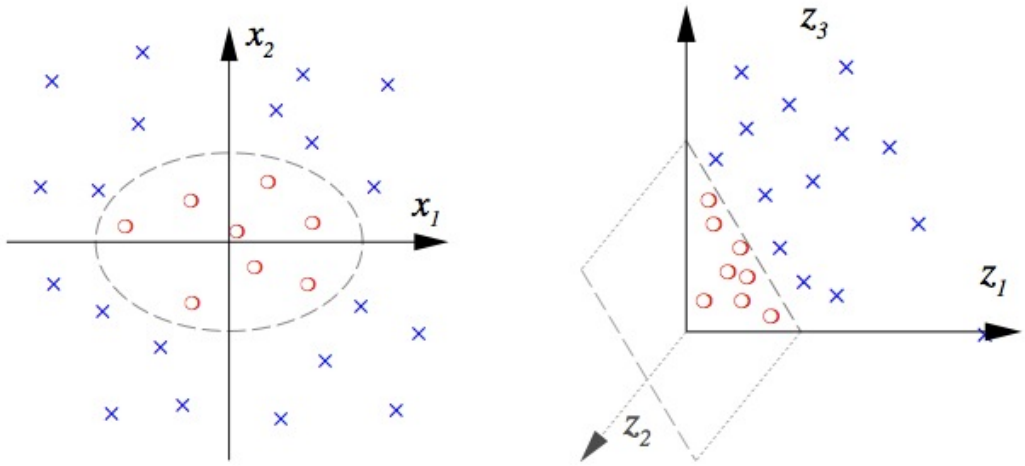
- Linear : $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Polynomial : $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$
- Gaussian : $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$

Support Vector Machine : non linear separation with Gaussian kernel



Example : polynomial kernel

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



Example : polynomial kernel

Kernel trick

Notice that $\varphi(\mathbf{x}_1)^T\varphi(\mathbf{x}')$ can be computed without working in $\mathbb{R}^3$

Define $k(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T\varphi(\mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^2$

Construction of a kernel

- Combine known kernels
- Specific kernels for certain types of data :
  - **Structured data** : ensembles, graphs, trees, sequences, ...
  - Unstructured data with an underlying structure : text, images, documents, signals, biological objects
- **Selection of a kernel** :
  - Hyperparameter learning : Chapelle et al. 2002
  - Multiple Kernel Learning : given $k_1, \ldots, k_m$, learn a convex combination $\sum_i \beta_i k_i$ (see SimpleMKL Rakotomamonjy et al. 2008, unifying view in Kloft et al. 2010)