Institut de Mathématiques de Toulouse, INSA Toulouse

Supervised Learning- Part II

Machine Learning for Data Science
CERFACS- May 2018

Béatrice Laurent-Philippe Besse- Aurélien Garivier

Methods studied in this course :

Part I

- Linear model, model selection, variable selection, Ridge regression, Lasso.
- Logistic regression
- Support Vector Machine

Part II

- Classification And Regression Trees (CART)
- Bagging, Random Forests
- Boosting
- Neural networks, Introduction to deep learning

Outline

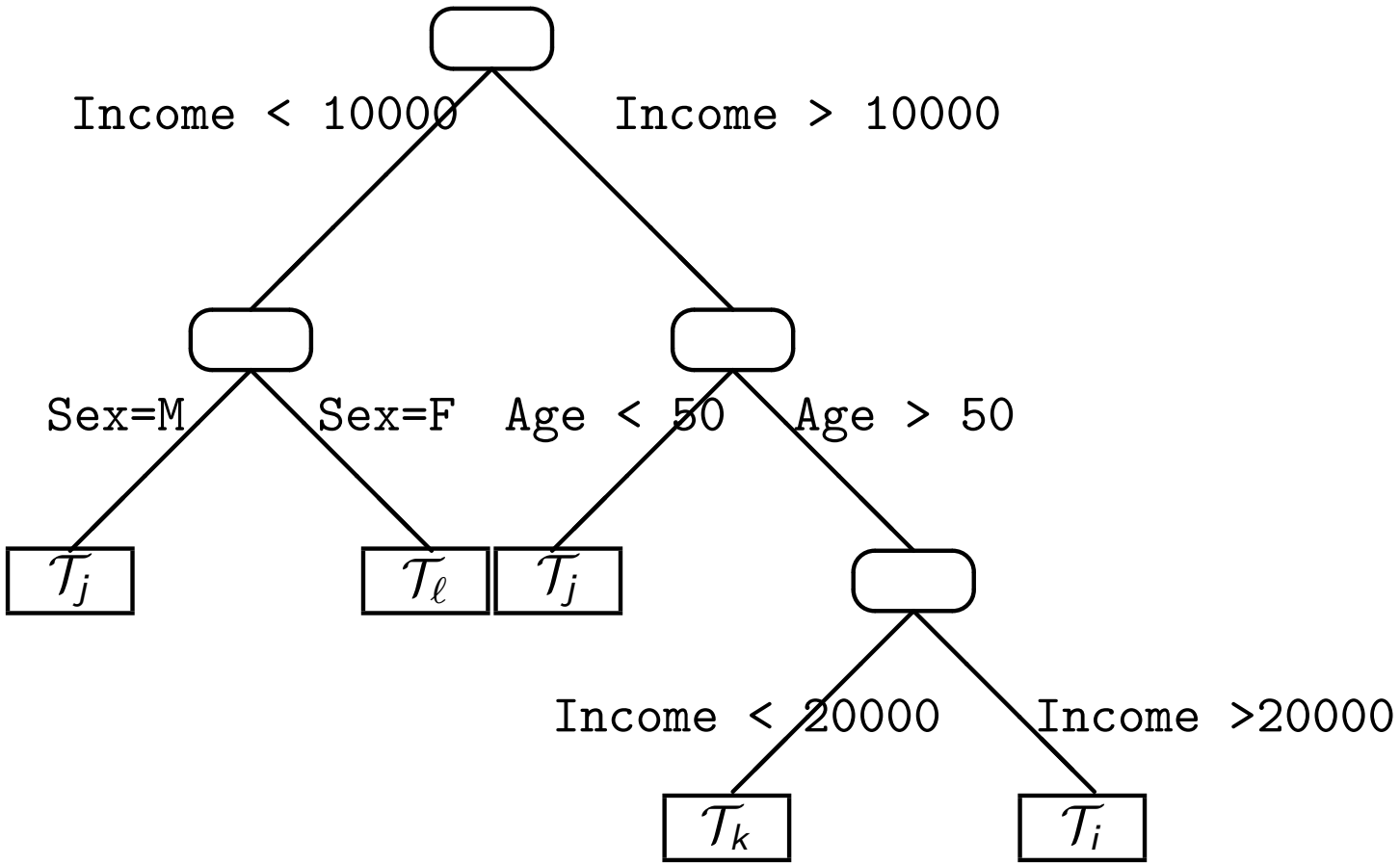
- Classification And Regression Trees (CART)
- Bagging, Random Forests
- Boosting
- Neural networks, Introduction to deep learning

Classification And Regression Trees

Introduction

- Classification and regression trees (CART) : Breiman et al. (1984)
- X^j explanatory variables (quantitative or qualitative)
- Y qualitative with m modalities $\{\mathcal{T}_\ell; \ell = 1 \ldots, m\}$: classification tree
- Y quantitative : regression tree
- Objective : construction of a binary decision tree easy to interpret
- No assumption on the model : non parametric procedure.

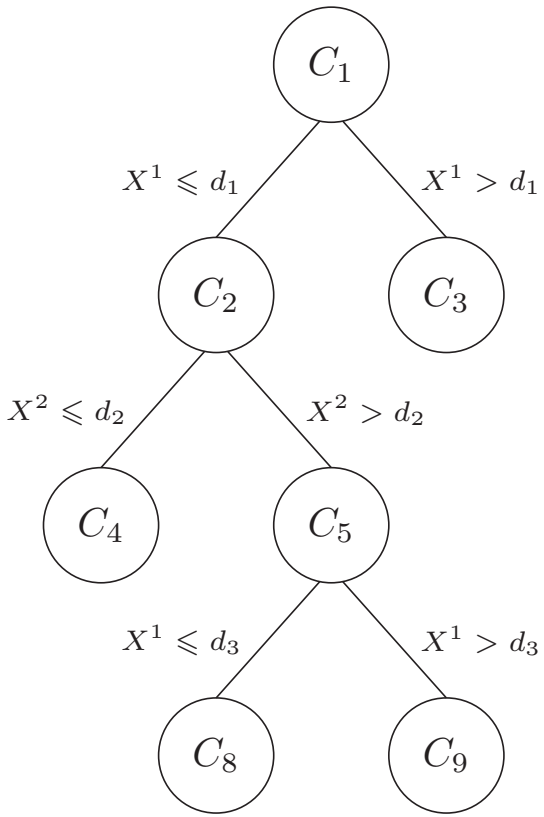
Example of binary classification tree

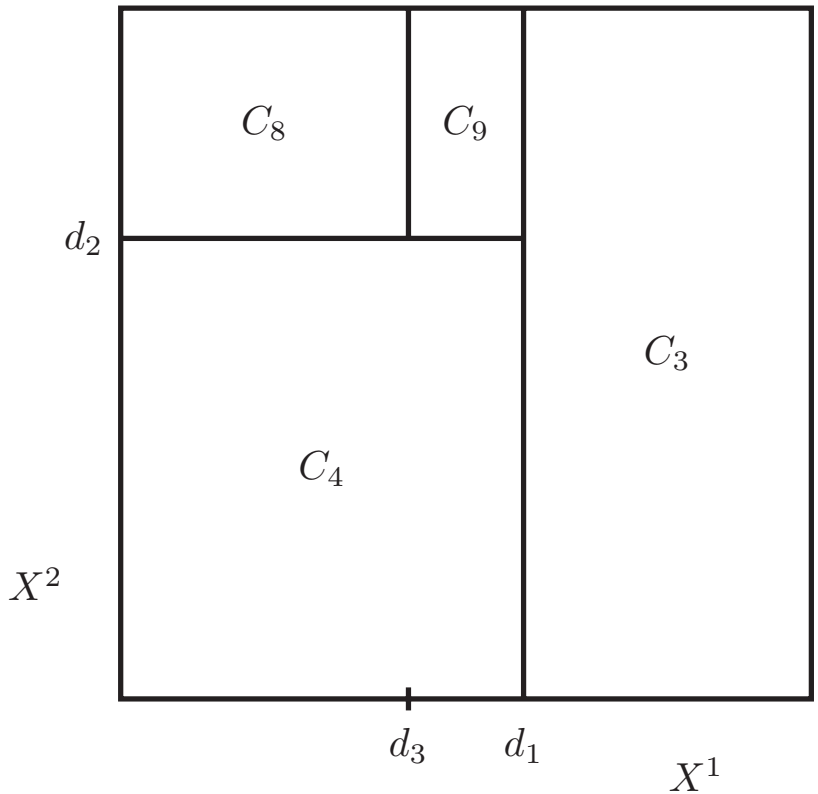


Definitions

- Determine an iterative sequence of *nodes*
- *Root* : initial note : the whole sample
- *Leaf* : Terminal node
- *Node* : choice of one variable and one *division* to proceed to a dichotomie
- *Division* : threshold value or group of modalities

Example of dyadic partition of the space





Rules

- We have to choose :
  1. Criterion for the "best" division among all *admissibles* ones (partition based on the values of one variable)
  2. Rules for a terminal node : *leaf*
  3. Rules to assign to a class $\mathcal{T}_\ell$ or one value for Y
- *Admissible* divisions : descendants $\neq \emptyset$
- X^j real or ordinal with c_j possible values : $(c_j - 1)$ possible divisions
- X^j nominal : $2^{(c_j-1)} - 1$ possible divisions
- Heterogeneity function D_κ of one node
  1. Null : a single modality for Y or Y is constant
  2. Maximal : all the modalities for Y or large variance

Division criterion

Optimal division

- Notations
  - κ : a node
  - κ_L and κ_R the two son nodes
- The algorithm retains the division which minimizes

$$D_{\kappa_L} + D_{\kappa_R}$$

- For each node κ in the construction of the tree :

$$\max_{\{Divisions\ of\ X^j; j=1,p\}} D_\kappa - (D_{\kappa_L} + D_{\kappa_R})$$

Stopping rule and affectation

Leaf and affectation

- A Node is a terminal node or a leaf, if it is :
  - Homogeneous
  - Number of observations below some threshold
- Affectation
  - Y quantitative, the value is the mean of the observations in the leaf
  - Y qualitative, each leaf is affected to one class $\mathcal{T}_\ell$ of Y by considering the conditional mode :
    - the mostly represented class in the node
    - The less cosltly class if cost for wrong classification are given

Heterogeneity criterion in regression

Y quantitative : heterogeneity in regression

Heterogeneity of the node κ :

$$D_\kappa = \sum_{i \in \kappa} (y_i - \overline{y}_\kappa)^2 = |\kappa| \frac{1}{|\kappa|} \sum_{i \in \kappa} (y_i - \overline{y}_\kappa)^2$$

where $|\kappa|$ is the cardinality of the node κ

Minimize the intra-class variance

The son nodes κ_L and κ_R minimize :

$$\frac{1}{n} \sum_{i \in \kappa_L} (y_i - \overline{y}_{\kappa_L})^2 + \frac{1}{n} \sum_{i \in \kappa_R} (y_i - \overline{y}_{\kappa_R})^2.$$

Heterogeneity criterion in classification

Y qualitative : $Y \in \{\mathcal{T}_\ell, l = 1, \ldots, m\}$.
Node κ.

- $n_\kappa^\ell = \text{Card } \{(X_i, Y_i), X_i \in \kappa, Y_i \in \mathcal{T}_\ell\}$.
- $n_\kappa = \text{Card } \{(X_i, Y_i), X_i \in \kappa\}$.
- p_κ^ℓ : probability that an observation is in class $\mathcal{T}_\ell$ given that it is in node κ.
- Estimated by $\frac{n_\kappa^\ell}{n_\kappa}$.

Heterogeneity criterion in classification

Y qualitative : heterogeneity in classification

Heterogeneity of node κ :

- Shannon Entropy with the notation $0 \log(0) = 0$
  p_κ^ℓ : proportion of the class $\mathcal{T}_\ell$ of Y in the node κ.

$$E_\kappa = -\sum_{\ell=1}^{m} p_\kappa^\ell \log(p_\kappa^\ell)$$

Maximal in $(\frac{1}{m}, \ldots, \frac{1}{m})$, minimal in $(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$.

$$D_\kappa = -|\kappa| \sum_{\ell=1}^{m} p_\kappa^\ell \log(p_\kappa^\ell)$$

- Gini concentration : $D_\kappa = |\kappa| \sum_{\ell=1}^{m} p_\kappa^\ell (1 - p_\kappa^\ell)$

Pruning and optimal tree

Pruning : notations

- We look for a parcimonious tree
- *Complexity* of a tree : K_A = numbers of leaves in A
- Adjustment error of A :

$$D(A) = \sum_{\kappa=1}^{K_A} D_\kappa$$

D_κ : heterogeneity of leaf κ

Sequence of embedded trees

- Adjustment error penalized by the complexity :

$$Crit_\gamma(A) = D(A) + \gamma \times K_A$$

- When $\gamma = 0$: $A_{\max}$ (maximal tree) minimizes $Crit_\gamma(A)$
- When γ increases, the division of A_H, for which the improvement of D is smaller than γ, is cancelled ; hence
  - two leaves are gathered (prunned)
  - there father node becomes a terminal node
  - A_K becomes A_{K-1}.
- After iteration of this process, we get a sequence of trees :

$$A_{\max} \supset A_K \supset A_{K-1} \supset \cdots A_1$$

Breiman sub-sequence

- A_K is the sub tree of $A_{\max}$ (maximal tree) obtained by pruning the nodes κ such that $D(\kappa) = D(\kappa_L) + D(\kappa_R)$.
- For each node in A_K, $D(\kappa) > D(\kappa_L) + D(\kappa_R)$ and $D(\kappa) > D(A_K^\kappa)$ where A_K^κ is the subtree of A_K from node κ.
- For γ small, $D(\kappa) + \gamma > D(A_K^\kappa) + \gamma|A_K^\kappa|$. This holds while $\gamma < (D(A_K^\kappa) - D(\kappa))/(|A_K^\kappa| - 1) = s(\kappa, A_K^\kappa)$ for all node κ of A_K.

$$\gamma_K = \inf_{\kappa \text{ node of } A_K} s(\kappa, A_K^\kappa)$$

- $Crit_{\gamma_K}(\kappa) = Crit_{\gamma_K}(A_K^\kappa)$ and the node κ becomes preferable to the subtree A_K^κ.
- $A_{K-1} = A_{\gamma_K}$ is the subtree obtained by pruning the branches from the nodes minimizing $s(\kappa, A_K^\kappa)$: this gives the second tree in the sub-sequence
- We iterate this process.

Optimal tree

Algorithm to select the optimal tree

- Maximal tree $A_{\max}$
- Imbedded sequence $A_{\max}, A_K \dots A_1$ associated with an increasing sequence of values $\gamma_K \leq \dots \leq \gamma_1$
- V-fold cross validation error :
  for $v = 1, \dots, V$ **do**
  - Estimation of the sequence of trees associated to (γ_κ) with all the folds except v
  - Estimation of the error with the fold v.

  EndFor
- Sequence of the mean of these errors for each value of $\gamma_K, \dots, \gamma_1$
- γ_{Opt} optimal value for the tuning parameter minimizing the mean of the errors
- Tree associated to γ_{Opt} in $A_K \dots A_1$

Advantages

- Trees are easy to interpret
- Efficient algorithms to find the pruned trees
- Tolerant to missing data

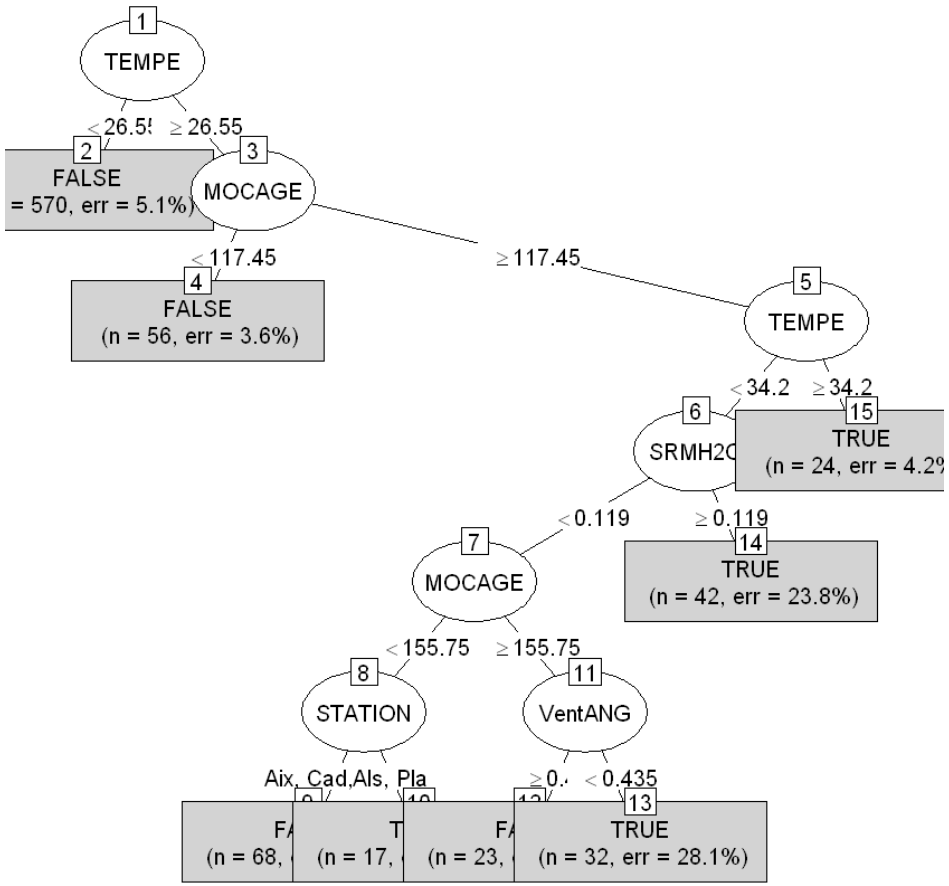
$\Longrightarrow$ Success of CART for practical applications

Warnings

- Variable selection : the selected tree only depends on few explanatory variables, trees are often (wrongly) interpreted as a variable selection procedure
- High instability of the trees : not robust to the learning sample, curse of dimensionality ..
- Prediction accuracy of a tree is often poor compared to other procedures

$\Longrightarrow$ Aggregation of trees : bagging, random forests

Example for Ozone data



Ozone : Classification tree pruned by cross-validation

Outline

- Classification And Regression Trees (CART)
- Bagging , Random Forests
- Boosting
- Neural networks, Introduction to deep learning

Introduction

- Combination or aggregation of models (almost) without overfitting
- Bagging is for bootstrap aggregating : Breiman, 1996
- Random forests : Breiman, 2001
- From Boosting (Freund et Shapiro,1996) deterministic and adaptative to *extreme gradient boosting*
- Allows to aggregate any modelisation method
- Efficient methods : Fernandez-Delgado et al. (2014), *Kaggle*

- Bagging is appropriate for unstable algorithms, with small bias and high variance (CART)
- Boosting is appropriate for algorithms with high bias and small variance

Bagging

Bootstrap aggregating

- Let Y a quantitative or qualitative variable to explain
- $X^1, \ldots, X^p$ the explanatory variables
- $f(\mathbf{x})$ a model function of $\mathbf{x} = \{x^1, \ldots, x^p\} \in \mathbb{R}^p$
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ learning sample with distribution F and size n
  - $\widehat{f}_{\mathbf{z}}$: predictor associated to the sample $\mathbf{z}$ with $f(.) = E_F(\widehat{f}_{\mathbf{z}})$
  - B independent samples $\{\mathbf{z}_b\}_{b=1,B}$
  - Y quantitative : $\widehat{f}_B(.) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(.)$ (mean)
  - Y qualitative : $\widehat{f}_B(.) = \arg\max_j \text{card}\left\{b \mid \widehat{f}_{\mathbf{z}_b}(.) = j\right\}$ (majority vote)
- Principle : Take the mean of independent predictions to reduce the variance
- B independent samples replaced by B bootstrap replications

Bagging : algorithm

- Let $\mathbf{x}_0$ to point where we want to predict and
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ a learning sample
- **For** $b = 1$ **to** B
  - Generate a bootstrap sample $\mathbf{z}_b^*$ (with size $m_n \leq n$)
  - Estimate $\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ with the bootstrap sample
- Compute the mean estimation $\widehat{f}_B(\mathbf{x}_0) = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ or the result of the majoritary vote

- The B boostrap samples are built on the same learning sample $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$:
  $\Longrightarrow$ the estimators $\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ are not independent
- Regression case : If $\text{Corr}(\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0), \widehat{f}_{\mathbf{z}_{b'}}(\mathbf{x}_0)) = \rho(x_0)$,

$$\begin{aligned}
\text{E}(\widehat{f}_B(\mathbf{x}_0)) &= f(\mathbf{x}_0) \\
\text{Var}(\widehat{f}_B(\mathbf{x}_0)) &= \rho(x_0)\text{Var}(\widehat{f}_b(\mathbf{x}_0)) + \frac{(1-\rho(x_0))}{B}\text{Var}(\widehat{f}_b(\mathbf{x}_0)) \\
&\to \rho(x_0)\text{Var}(\widehat{f}_b(\mathbf{x}_0)) \text{ as } B \to +\infty
\end{aligned}$$

$\Longrightarrow$ Importance to find low correlated predictors $(\widehat{f}_b(\mathbf{x}_0))_{1 \leq b \leq B}$.
$\hookrightarrow$ **Random forests**

Bagging : practical use

- *Bootstrap out-of-bag* estimation of the prediction error : control of the quality and of overfitting
- CART to built a sequence of binary trees
- Three pruning strategies are possible :
  1. keep a whole tree for each of the samples
  2. tree with at most q leaves
  3. whole tree pruned by cross-validation
- First strategy compromise between computations and prediction accuracy : small bias for each tree and reduced variance by aggregation

Bagging : limitations

- Computation time and control of the error
- Storage of all the models to aggregate
- Black box model

Outline

- Classification And Regression Trees (CART)
- Bagging, Random Forests
- Boosting
- Neural networks, Introduction to deep learning

Random forests

Random forests : principle

- Random forests means initially any aggregation method for classification or regression trees
- Now, it refers to the **Random input** method introduced by Breiman and Cutler (2005)
  `http://www.stat.berkeley.edu/users/breiman/RandomForests/`
- Improvement of the bagging of binary trees
- Variance of B correlated variables : $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$
- Add a randomisation to get more independent trees
- Random choice of variables
- Interesting in high dimension

Random forests : algorithm

- Let $\mathbf{x}_0$ the point where we want to predict,
  $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ a learning sample **for** $b = 1$ **to** B
  - Generate a bootstrap sample $\mathbf{z}_b^*$
  - Estimate a tree with randomization of the variables :
  - For each node, random sample of $m < p$ predictors to built the subdivision
- Compute the mean estimation $\widehat{f}_B(\mathbf{x}_0) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$
  or the majority vote

Random forests : utilisation

- Pruning : tree with q leaves, or complete tree,
- Random selection of m predictors : $m = \sqrt{p}$ for classification, $\frac{p}{3}$ for regression.
- Small m increases the variability of each tree but reduces the correlation between them : Each model is less accurate but l'aggregation is performing
- Iterative computation of the out-of-bag error.
- Good accuracy, easily implementable, parallelisable but not easy to interpret

To help interpretation

- Index of importance for each variable
  - Mean Decrease Accuracy
  - Mean Decrease Gini

Other possible applications

- Proximity or similarity between observations
- Anomaly detection with `IsolationForest`
- Imputation of missing data with `missForest`
- Survival analysis with `survival forest`
- ...

Outline

- Classification And Regression Trees (CART)
- Bagging , Random Forests
- Boosting
- Neural networks, Introduction to deep learning

Boosting principle

Boosting : principle

- Improve the performances of a weak classifier (Schapire, 1990 ; Freund and Schapire, 1996)
- AdaBoost (Adaptative boosting) prediction of a binary variable
- Reduction of the variance but also the bias
- Aggregation of a family of recurrent models : *Each model is an adaptive version of the previous one giving more weight, in the next estimate, to the badly adjusted observations*
- Variants : type of variable to predict (binary, k classes, real), loss fonction

Basic algorithm

AdaBoost

- AdaBoost is a boosting method to combine several binary classifiers $f_1, \ldots, f_k$ with values in $\{-1, 1\}$.
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ a learning sample, $y_i \in \{-1, 1\}$.
- The aim is to minimize the empirical convexified risk for the exponential loss function over linear combination of the classifiers

$$\hat{f} = \underset{f \in \text{span}(f_1, \ldots, f_k)}{\text{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i f(\mathbf{x}_i)) \right\}.$$

AdaBoost

- To approximate the solution, Adaboost computes a sequence of functions $\hat{f}_m$ for $m = 0, \ldots M$ with

$$\begin{aligned} \hat{f}_0 &= 0 \\ \hat{f}_m &= \hat{f}_{m-1} + \beta_m f_{j_m} \end{aligned}$$

where (β_m, j_m) minimizes the empirical risk

$$\underset{\beta \in \mathbb{R}, j=1,\ldots,p}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i(\hat{f}_{m-1}(\mathbf{x}_i) + \beta f_j)) \right\}.$$

- The final classification rule is given by

$$\hat{f} = \operatorname{sign}(\hat{f}_M).$$

AdaBoost

- We denote
$$w_i^{(m)} = \frac{1}{n} \exp(-y_i \hat{f}_{m-1}(\mathbf{x}_i))$$
- We assume that for all $j = 1, \ldots, k$,
$$\widehat{\mathcal{E}_m}(j) = \frac{\sum_{i=1}^n w_i^{(m)} \mathbb{1}_{f_j(\mathbf{x}_i) \neq y_i}}{\sum_{i=1}^n w_i^{(m)}} \in]0, 1[.$$
- Then, we have :
$$j_m = \underset{j=1,\ldots,k}{\operatorname{argmin}} \widehat{\mathcal{E}_m}(j),$$
and
$$\beta_m = \frac{1}{2} \log \left(\frac{1 - \widehat{\mathcal{E}_m}(j)}{\widehat{\mathcal{E}_m}(j)} \right).$$

AdaBoost

AdaBoost algorithm

- $w_i^{(1)} = 1/n$ for $i = 1, \ldots, n$.
- For $m = 1, \ldots, M$

$$
\begin{aligned}
j_m &= \underset{j=1,\ldots,p}{\operatorname{argmin}} \widehat{\mathcal{E}_m}(j) \\
\beta_m &= \frac{1}{2} \log\left(\frac{1 - \widehat{\mathcal{E}_m}(j)}{\widehat{\mathcal{E}_m}(j)}\right) \\
w_i^{(m+1)} &= w_i^{(m)} \exp(-y_i \beta_m f_{j_m}(\mathbf{x}_i)) \text{ for } i = 1, \ldots, n \\
&= \frac{1}{n} \exp(-y_i \hat{f}_m(\mathbf{x}_i))
\end{aligned}
$$

- $\hat{f}_M(x) = \sum_{m=1}^{M} \beta_m f_{j_m}(x)$.
- $\hat{f} = \text{sign}(\hat{f}_M)$.

Boosting : utilisation

- Tree as basic model
- Recommandation : q between 4 and 8
- Empirically , the prediction error can still decrease while the training error is equal to zero
- Warning : noisy data (error in the label), source of drift or overfitting
- Boosting reduces the variance like the bagging but also the bias

Gradient Boosting Models

GBM : Principle 1

- Gradient Boosting Models (Friedman, 2002-2009)
- In the case of a differentiable loss function
- Principle :
  - Construct a sequence of models in such a way that at each step, each model added to the linear combination, appears as a step towards a better solution
  - This step is done in the direction of the gradient of the loss function approached by a regression tree

GBM : Principle 2

- Previous Adaptative model (AdaBoost) :

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m f_{j_m}(\boldsymbol{x})$$

- Transformed in a gradient descent

$$f_m = f_{m-1} - \gamma_m \sum_{i=1}^{n} \nabla_{f_{m-1}(\boldsymbol{x}_i)} l(y_i, f_{m-1}(\boldsymbol{x}_i)).$$

- Search for a best step in the descent γ :

$$min_\gamma \sum_{i=1}^{n} \left[l\left(y_i, f_{m-1}(\boldsymbol{x}_i) - \gamma \frac{\partial l(y_i, f_{m-1}(\boldsymbol{x}_i))}{\partial f_{m-1}(\boldsymbol{x}_i)} \right) \right].$$

GBM in regression : algorithm

- Let $\mathbf{x}_0$ the point where we want to predict
- Initialize $\widehat{f}_0 = \arg\min_\gamma \sum_{i=1}^{n} l(y_i, \gamma)$
- **For** $m = 1$ **to** M
  - Compute $r_{mi} = -\left[\frac{\delta l(y_i, f(\mathbf{x}_i))}{\delta f(\mathbf{x}_i)}\right]_{f=f_{m-1}} \quad ; \quad i = 1, \ldots, n$
  - Adjust a regression tree δ_m to $(\mathbf{x}_i, r_m i)$
  - Calculate $\gamma_m = \text{arg}min_\gamma \sum_{i=1}^{n} l(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma \delta_m(\boldsymbol{x}_i))$
  - Update : $\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \gamma_m \delta_m(\boldsymbol{x})$
- Result : $\widehat{f}_M(\boldsymbol{x}_0)$

GBM : use with R

- Discrimination : as many probabilities as classes
- *Shrinkage* coefficient

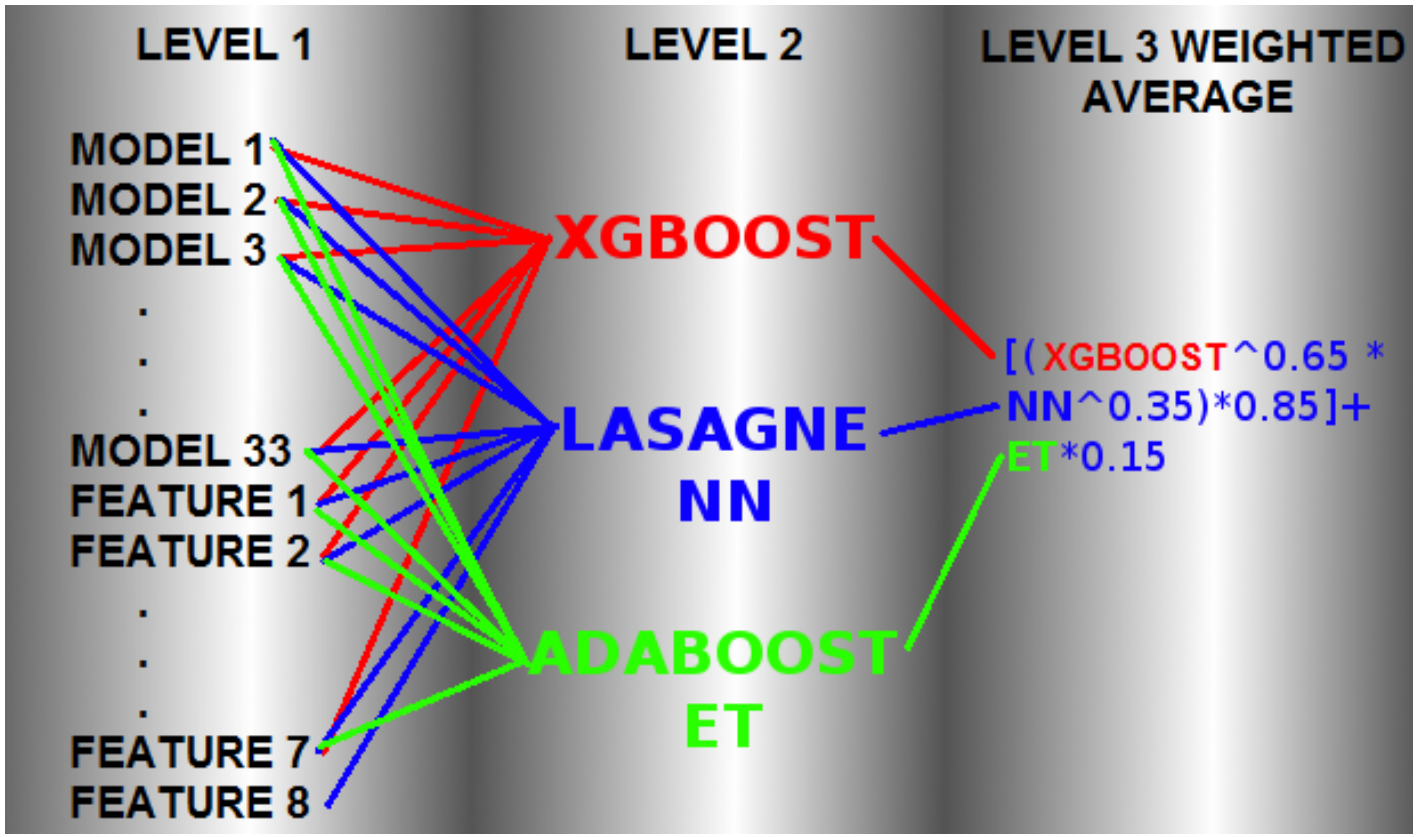
$$\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}\{\boldsymbol{x} \in R_{jm}\}$$

- Trade-off between shrinkage and number of iterations M.
- Maximum depth of the trees

Extreme gradient boosting

`XGBoost` : motivation

- Algorithm introduced by Chen et Guestrin (2016)
- Additional Penalization to control overfitting
- Problem : number of parameters to optimize
- Implementation tips for parallelisation
- Environments : R (`caret`), Python, Julia, GPU, *Amazon Web Service*, Spark...
- Winning solutions for *Kaggle* competitions



Kaggle : Identify people who have a high degree of Psychopathy based on Twitter usage

XGBoost : penalization

- Loss function L penalized version of l

$$L(f) = \sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y_i) + \sum_{m=1}^{M} \Omega(\delta_m)$$

$$\Omega(\delta) = \alpha|\delta| + \beta||\boldsymbol{w}||^2$$

- $|\delta|$ number of leaves in the tree δ
- $\boldsymbol{w}$ vector of values assigned to each leef
- Ω Mix of l_1 and l_2 penalization

XGBoost : tips

- Gradient Approximation by Taylor expansion : summations and parallelization
- Complexity of subdivisions in the trees : quantiles of the distributions
- Tolerant algorithm for Missing data : gradient calculated on present values
- Index of importance of the variables

XGBoost : additional parameters

`alpha`	Lasso (l_1) penalization of the complexity of the regression tree to estimate the gradient
`lambda`	*Ridge* (l_2) penalization
`gamma`	Minimal reduction of the loss to accept a division
`tree_method`	greedy algorithm for searching for divisions or simplification (quantiles or grouping of classes)
`sketch_eps`	control of the number of classes
`scale_pos_weight`	to take into account for unbalanced classes
Other	Optimisation parameters of the performances

XGBoost : Optimisation strategy

- Principle :
- Default values and optimize the parameters by decreasing order of the supposed influence
- A strategy among others :
  1. Number of trees
  2. Maximal depth *vs.* minimal number observations per leef
  3. Minimal reduction of the loss
  4. Sampling rate *vs.* number of variables used
  5. Number of trees *vs.* shrinkage
- Need : computing power (GPU)

Outline

- Classification And Regression Trees (CART)
- Bagging, Random Forests
- Boosting
- Neural networks, Introduction to deep learning

Introduction

- Deep learning is a set of learning methods attempting to model data with complex architectures combining different non-linear transformations.
- The elementary bricks of deep learning are the neural networks, that are combined to form the deep neural networks.

There exist several types of architectures for neural networks :

- The multilayer perceptrons, that are the oldest and simplest ones
- The Convolutional Neural Networks (CNN), particularly adapted for image processing
- The recurrent neural networks, used for sequential data such as text or times series.

Introduction

- Deep learning architectures are based on deep cascade of layers.
- They need clever stochastic optimization algorithms, and initialization, and also a clever choice of the structure.
- They lead to very impressive results, although very few theoretical fondations are available till now.
- These techniques have enabled significant progress in the fields of sound and image processing, including facial recognition, speech recognition, computer vision, automated language processing, text classification (for example spam recognition).

Neural networks

- An artificial neural network is non linear with respect to its parameters θ that associates to an entry x an output $y = f(x, \theta)$.
- The neural networks can be use for regression or classification.
- The parameters θ are estimated from a learning sample.
- The function to minimize is not convex, leading to local minimizers.
- The success of the method came from a universal approximation theorem due to Cybenko (1989) and Hornik (1991).
- Le Cun (1986) proposed an efficient way to compute the gradient of a neural network, called backpropagation of the gradient, that allows to obtain a local minimizer of the quadratic criterion easily.

Artificial Neuron

Artificial neuron

- a function f_j of the input $x = (x_1, \ldots, x_d)$
- weighted by a vector of connection weights $w_j = (w_{j,1}, \ldots, w_{j,d})$,
- completed by a neuron bias b_j,
- and associated to an activation function ϕ :

$$y_j = f_j(x) = \phi(\langle w_j, x \rangle + b_j).$$

Activation functions

Activation functions

- The identity function $\phi(x) = x$
- The sigmoid function (or logistic) $\phi(x) = \frac{1}{1+e^{-x}}$
- The hyperbolic tangent function ("tanh")

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- The hard threshold function $\phi_\beta(x) = \mathbb{1}_{x \geq \beta}$
- The Rectified Linear Unit (ReLU) activation function $\phi(x) = \max(0, x)$

Activation functions

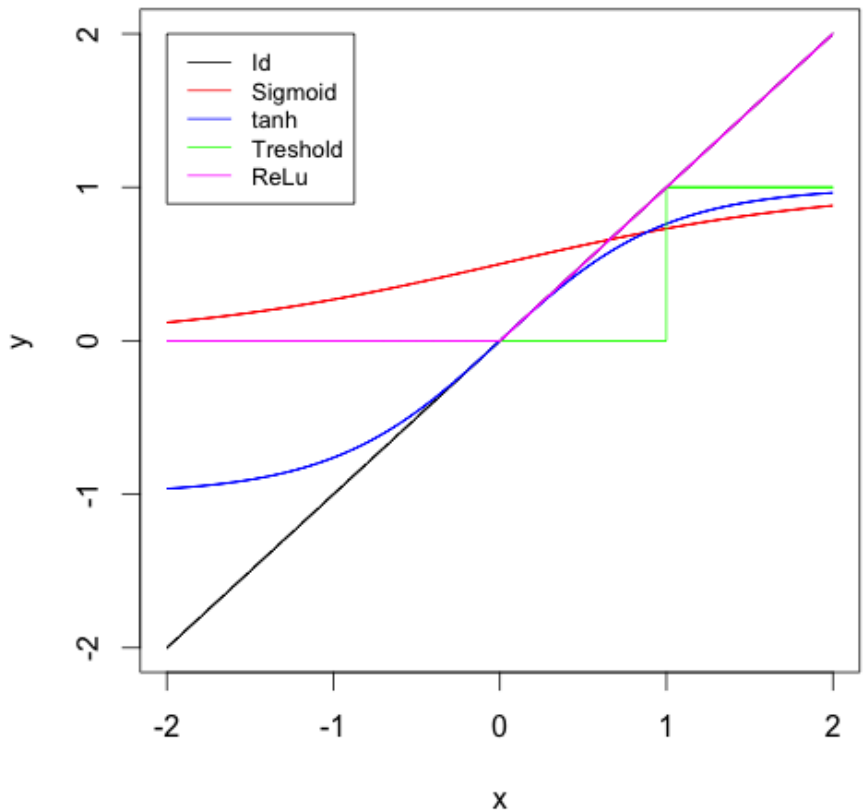
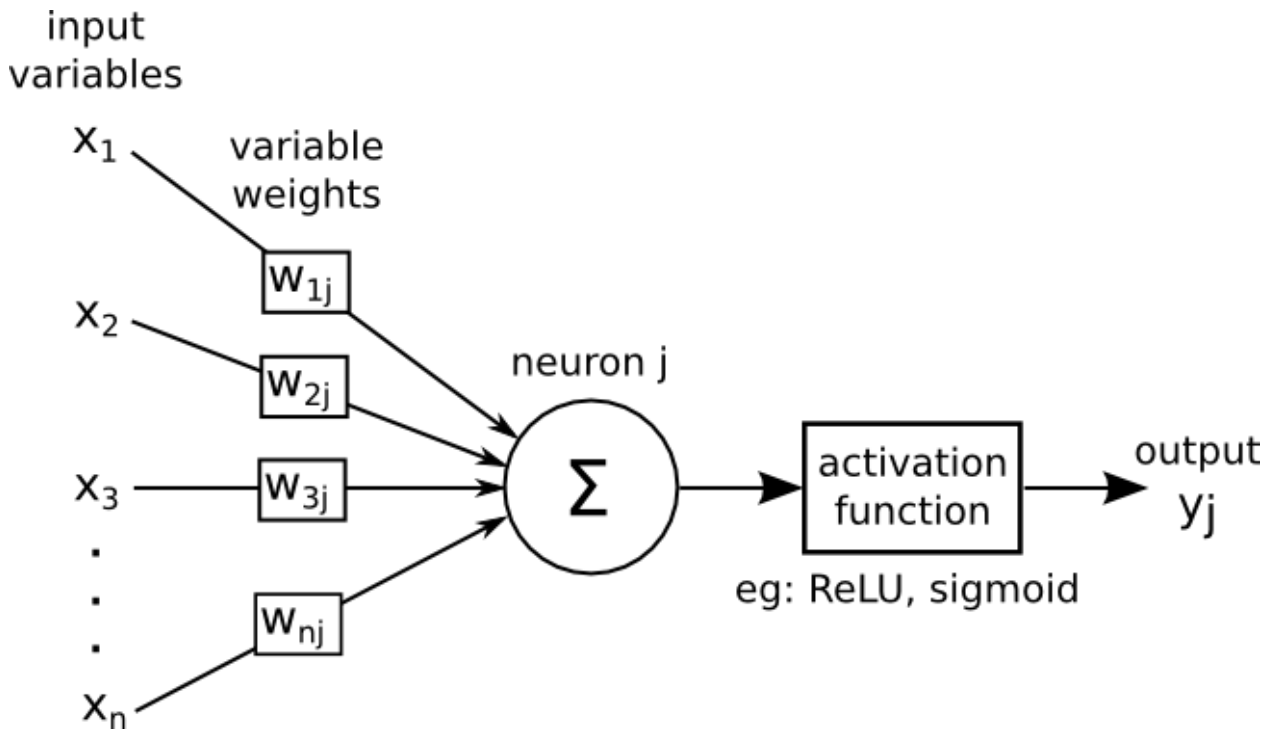


FIGURE: Activation functions

Artificial Neuron

Schematic representation of an artificial neuron where $\Sigma = \langle w_j, x \rangle + b_j$.



- Historically, the sigmoid was the mostly used activation function since it is differentiable and allows to keep values in the interval $[0, 1]$.
- Nevertheless, it is problematic since its gradient is very close to 0 when $|x|$ is not close to 0.

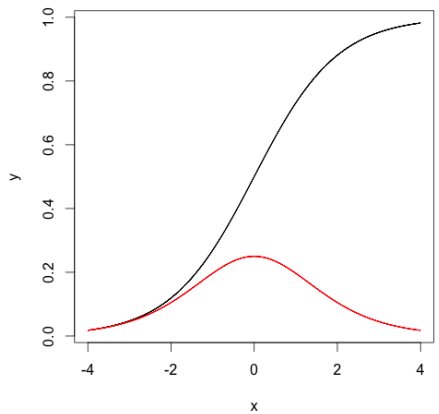


FIGURE: Sigmoid function (in black) and its derivatives (in red)

Activation functions

- With neural networks with a high number of layers, this causes troubles for the backpropagation algorithm to estimate the parameters.
- This is why the sigmoid function was supplanted by the rectified linear function (ReLu).
- This function is not differentiable in 0 but the probability to have an entry equal to 0 is generally null.
- The ReLU function and its derivative are equal to 0 for negative values, hence it is advised
  - to add a small positive bias to ensure that each unit is active.
  - to consider a variations of the ReLU function such as

$$\phi(x) = \max(x, 0) + \alpha \min(x, 0)$$

where α is either a fixed parameter set to a small positive value, or a parameter to estimate.

Multilayer perceptron

- A multilayer perceptron (or neural network) is a structure composed by several hidden layers of neurons where the output of a neuron of a layer becomes the input of a neuron of the next layer.
- On last layer, called output layer, we may apply a different activation function as for the hidden layers depending on the type of problems we have at hand : regression or classification.

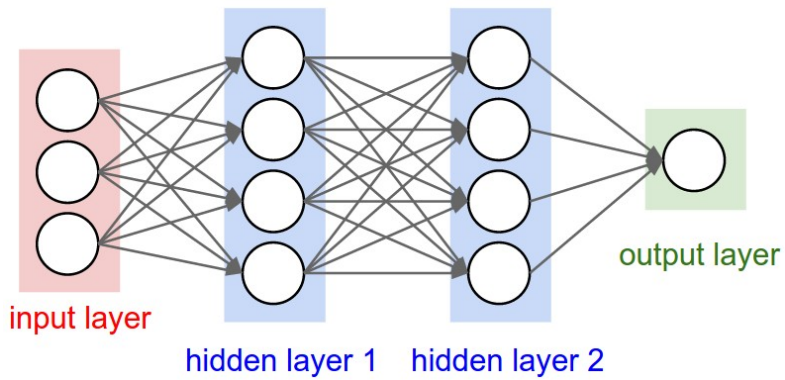


FIGURE: A basic neural network.

Multilayers perceptrons

- The parameters of the architecture are the number of hidden layers and of neurons in each layer.
- The activation functions are also to choose by the user. On the output layer, we apply no activation function in the case of regression.
- For binary classification, the output gives a prediction of $\mathbb{P}(Y = 1/X)$ since this value is in $[0, 1]$, the sigmoid activation function is generally considered.
- For multi-class classification, the output layer contains one neuron per class i, giving a prediction of $\mathbb{P}(Y = i/X)$. The sum of all these values has to be equal to 1.
- The multidimensional function **softmax** is generally used

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

Mathematical formulation

Multilayer perceptron with L hidden layers :

- We set $h^{(0)}(x) = x$.
  For $k = 1, \ldots, L$ (hidden layers),

$$\begin{aligned} a^{(k)}(x) &= b^{(k)} + W^{(k)} h^{(k-1)}(x) \\ h^{(k)}(x) &= \phi(a^{(k)}(x)) \end{aligned}$$

  For $k = L + 1$ (output layer),

$$\begin{aligned} a^{(L+1)}(x) &= b^{(L+1)} + W^{(L+1)} h^{(L)}(x) \\ h^{(L+1)}(x) &= \psi(a^{(L+1)}(x)) := f(x, \theta). \end{aligned}$$

  where ϕ is the activation function and ψ is the output layer activation function
- At each step, $W^{(k)}$ is a matrix with number of rows the number of neurons in the layer k and number of columns the number of neurons in the layer $k - 1$.

Estimation of the parameters

- Once the architecture of the network has been chosen, the parameters (the weights w_j and biases b_j) have to be estimated from a learning sample.
- As usual, the estimation is obtained by minimizing a loss function with a gradient descent algorithm.
- We first have to choose the loss function.
- It is classical to estimate the parameters by minimizing the loss function which is the opposite of the log likelihood.
- Denoting θ the vector of parameters to estimate, we consider the expected loss function

$$L(\theta) = -\mathbb{E}_{(X,Y)\sim P}(\log(p_\theta(Y/X)).$$

Loss functions

- If the model is Gaussian, namely if $P_\theta(Y/X = x) \sim \mathcal{N}(f(x,\theta), I)$, maximizing the likelihood is equivalent to minimize the quadratic loss

$$L(\theta) = \mathbb{E}_{(X,Y)\sim P}(\|Y - f(X,\theta)\|^2).$$

- For binary classification, with $Y \in \{0,1\}$, maximizing the log likelihood corresponds to the minimization of the cross-entropy. Setting $f(X,\theta)) = P_\theta(Y = 1/X)$,

$$L(\theta) = -\mathbb{E}_{(X,Y)\sim P}\left[Y \log(f(X,\theta)) + (1-Y)\log(1 - f(X,\theta))\right].$$

- This loss function is well adapted with the sigmoid activation function since the use of the logarithm avoids to have too small values for the gradient.

Loss functions

- For a multi-class classification problem, we consider a generalization of the previous loss function to k classes

$$L(\theta) = -\mathbb{E}_{(X,Y)\sim P}\left[\sum_{j=1}^{k} \mathbb{1}_{Y=j} \log p_\theta(Y = j/X)\right].$$

- Ideally we would like to minimize the classification error, but it is not smooth, this is why we consider the cross-entropy (or eventually a convex surrogate).

Penalized empirical risk

- The expected loss can be written as

$$L(\theta) = \mathbb{E}_{(X,Y)\sim P}\left[\ell(f(X,\theta), Y)\right]$$

  and it is associated to a loss function ℓ.

- In order to estimate the parameters θ, we use a training sample $(X_i, Y_i)_{1\leq i\leq n}$ and we minimize the empirical loss

$$\tilde{L}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(X_i,\theta), Y_i)$$

  eventually we add a regularization term.

- This leads to minimize the penalized empirical risk

$$L_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(X_i,\theta), Y_i) + \lambda\Omega(\theta).$$

Penalized empirical risk

- We can consider $\mathbb{L}^2$ regularization.

$$\begin{aligned}\Omega(\theta) &= \sum_k \sum_i \sum_j (W_{i,j}^{(k)})^2 \\ &= \sum_k \|W^{(k)}\|_F^2\end{aligned}$$

  where $\|W\|_F$ denotes the Frobenius norm of the matrix W.
- Note that only the weights are penalized, the biases are not penalized.
- It is easy to compute the gradient of $\Omega(\theta)$:

$$\bigtriangledown_{W^{(k)}} \Omega(\theta) = 2W^{(k)}.$$

- One can also consider $\mathbb{L}^1$ regularization, leading to parcimonious solutions :

$$\Omega(\theta) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|.$$

Penalized empirical risk

- In order to minimize the criterion $L_n(\theta)$, a **stochastic gradient descent algorithm** is used.
- In order to compute the gradient, a clever method, called **Backpropagation algorithm** is considered.
- It has been introduced by Rumelhart et al. (1988), it is still crucial for deep learning.

The stochastic gradient descent algorithm

- Initialization of $\theta = (W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)})$.
- **For** $j = 1, \ldots, N$ iterations :
  - At step j :

$$\theta = \theta - \varepsilon \frac{1}{m} \sum_{i \in B} \left[\bigtriangledown_\theta \ell(f(X_i, \theta), Y_i) + \lambda \bigtriangledown_\theta \Omega(\theta) \right].$$

The stochastic gradient descent algorithm

- In the previous algorithm, we do not compute the gradient for the loss function at each step of the algorithm but only on a subset B of cardinality m (called a **batch**).
- This is what is classically done for big data sets (and for deep learning) or for sequential data.
- B is taken at random without replacement.
- An iteration over all the training examples is called an **epoch**.
- The numbers of epochs to consider is a parameter of the deep learning algorithms.
- The total number of iterations equals the number of epochs times the sample size n divided by m, the size of a batch.
- This procedure is called **batch learning**, sometimes, one also takes batches of size 1, reduced to a single training example (X_i, Y_i).

Backpropagation algorithm for regression

- We consider the regression case and explain how to compute the gradient of the empirical quadratic loss by the Backpropagation algorithm.
- To simplify, we do not consider here the penalization term, that can easily be added.
- Assuming that the output of the multilayer perceptron is of size K, and using the previous notations, the empirical quadratic loss is proportional to

$$\sum_{i=1}^{n} R_i(\theta)$$

with

$$R_i(\theta) = \sum_{k=1}^{K} (Y_{i,k} - f_k(X_i, \theta))^2.$$

Backpropagation algorithm for regression

- In a regression model, the output activation function ψ is generally the identity function, to be more general, we assume that

$$\psi(a_1, \ldots, a_K) = (g_1(a_1), \ldots, g_K(a_K))$$

  where $g_1, \ldots, g_K$ are functions from $\mathbb{R}$ to $\mathbb{R}$.
- Computation of the partial derivatives of R_i with respect to the weights of the output layer.
  - Recalling that

$$a^{(L+1)}(x) = b^{(L+1)} + W^{(L+1)} h^{(L)}(x),$$

    we get

$$\frac{\partial R_i}{\partial W_{k,m}^{(L+1)}} = -2(Y_{i,k} - f_k(X_i, \theta)) g_k'(a_k^{(L+1)}(X_i)) h_m^{(L)}(X_i).$$

Backpropagation algorithm for regression

- Computation of the partial derivatives of R_i with respect to the weights of the previous layer.
  - Differentiating now with respect to the weights of the previous layer

$$\frac{\partial R_i}{\partial W_{m,l}^{(L)}} = -2 \sum_{k=1}^{K} (Y_{i,k} - f_k(X_i, \theta)) g_k'(a_k^{(L+1)}(X_i)) \frac{\partial a_k^{(L+1)}(X_i)}{\partial W_{m,l}^{(L)}}.$$

with

$$\begin{aligned} a_k^{(L+1)}(x) &= \sum_j W_{k,j}^{(L+1)} h_j^{(L)}(x), \\ h_j^{(L)}(x) &= \phi\left(b_j^{(L)} + \langle W_j^{(L)}, h^{(L-1)}(x) \rangle\right). \end{aligned}$$

  - This leads to

$$\frac{\partial a_k^{(L+1)}(x)}{\partial W_{m,l}^{(L)}} = W_{k,m}^{(L+1)} \phi'\left(b_m^{(L)} + \langle W_m^{(L)}, h^{(L-1)}(x) \rangle\right) h_l^{(L-1)}(x).$$

Backpropagation algorithm for regression

- Let us introduce the notations

$$\begin{aligned} \delta_{k,i} &= -2(Y_{i,k} - f_k(X_i, \theta))g_k'(a_k^{(L+1)}(X_i)) \\ s_{m,i} &= \phi'\left(a_m^{(L)}(X_i)\right) \sum_{k=1}^{K} W_{k,m}^{(L+1)} \delta_{k,i}. \end{aligned}$$

- Then we have

$$\frac{\partial R_i}{\partial W_{k,m}^{(L+1)}} = \delta_{k,i} h_m^{(L)}(X_i) \quad (1)$$

$$\frac{\partial R_i}{\partial W_{m,l}^{(L)}} = s_{m,i} h_l^{(L-1)}(X_i), \quad (2)$$

known as the **backpropagation equations**.

Backpropagation algorithm for regression

- The values of the gradient are used to update the parameters in the gradient descent algorithm.
- At step $r+1$, we have :

$$W_{k,m}^{(L+1,r+1)} = W_{k,m}^{(L+1,r)} - \varepsilon_r \sum_{i \in B} \frac{\partial R_i}{\partial W_{k,m}^{(L+1,r)}}$$

$$W_{m,l}^{(L,r+1)} = W_{m,l}^{(L,r)} - \varepsilon_r \sum_{i \in B} \frac{\partial R_i}{\partial W_{m,l}^{(L,r)}}$$

where B is a batch and $\varepsilon_r > 0$ is the learning rate that satisfies $\varepsilon_r \to 0$, $\sum_r \varepsilon_r = \infty$, $\sum_r \varepsilon_r^2 < \infty$, for example $\varepsilon_r = 1/r$.

Backpropagation algorithm for regression

- We use the Backpropagation equations to compute the gradient by a two pass algorithm.
- In the *forward pass*, we fix the value of the current weights $\theta^{(r)} = (W^{(1,r)}, b^{(1,r)}, \ldots, W^{(L+1,r)}, b^{(L+1,r)})$, and we compute the predicted values $f(X_i, \theta^{(r)})$ and all the intermediate values $(a^{(k)}(X_i), h^{(k)}(X_i) = \phi(a^{(k)}(X_i)))_{1 \leq k \leq L+1}$ that are stored.
- Using these values, we compute during the *backward pass* the quantities $\delta_{k,i}$ and $s_{m,i}$ and the partial derivatives given in Equations 1 and 2.
- We have computed the partial derivatives of R_i only with respect to the weights of the output layer and the previous ones, but we can go on to compute the partial derivatives of R_i with respect to the weights of the previous hidden layers.
- In the back propagation algorithm, each hidden layer gives and receives informations from the neurons it is connected with.
- Hence, the algorithm is adapted for parallel computations.

Initialization

- The input data have to be normalized to have approximately the same range.
- The biases can be initialized to 0. The weights cannot be initialized to 0 since for the tanh activation function, the derivative at 0 is 0, this is a saddle point.
- They also cannot be initialized with the same values, otherwise, all the neurons of a hidden layer would have the same behaviour.
- We generally initialize the weights at random : the values $W_{i,j}^{(k)}$ are i.i.d. Uniform on $[-c, c]$ with possibly $c = \frac{\sqrt{6}}{N_k + N_{k-1}}$ where N_k is the size of the hidden layer k. We also sometimes initialize the weights with a normal distribution $\mathcal{N}(0, 0.01)$ (see Gloriot and Bengio, 2010).

Optimization algorithms

- Many algorithms can be used to minimize the loss function with hyperparameters to calibrate.
- The Stochastic Gradient Descent (SGD) algorithm :

$$\theta_i^{new} = \theta_i^{old} - \varepsilon \frac{\partial L}{\partial \theta_i}(\theta_i^{old}),$$

where ε is the *learning rate* , and its calibration is very important for the convergence of the algorithm.

  - If it is too small, the convergence is very slow and the optimization can be blocked on a local minimum.
  - If the learning rate is too large, the network will oscillate around an optimum without stabilizing and converging.
  - A classical way to proceed is to adapt the learning rate during the training : recommended to begin with a "large " value of ϵ, and reduce this value during the successive iterations : The observation of the evolution of the loss function can give indications on the way to proceed.

Optimization algorithms

- Another stopping rule, called *early stopping* is also used : we stop learning when the loss function for a validation sample stops to decrease.
- *Batch learning* is used for computational reasons, backpropagation algorithms need to store intermediate values : for big data sets, such as millions of images, this is not feasible, all the more that the deep networks have millions of parameters to calibrate.
- The batch size m is also a parameter to calibrate. Small batches generally lead to better generalization properties.
- The particular case of batches of size 1 is called *On-line Gradient Descent*. The disadvantage of this procedure is the very long computation time.

SGD algorithm

Summary of Stochastic Gradient Descent algorithm

- Fix the parameters ε : learning rate, m : batch size, nb : number of epochs.
- For $l = 1$ to nb epochs
- For $l = 1$ to n/m,
  - Take a random batch of size m without replacement in the learning sample : $(X_i, Y_i)_{i \in B_l}$
  - Compute the gradients with the backpropagation algorithm

$$\tilde{\nabla}_\theta = \frac{1}{m} \sum_{i \in B_l} \nabla_\theta \ell(f(X_i, \theta), Y_i).$$

  - Update the parameters

$$\theta^{new} = \theta^{old} - \varepsilon \tilde{\nabla}_\theta.$$

SGD algorithm

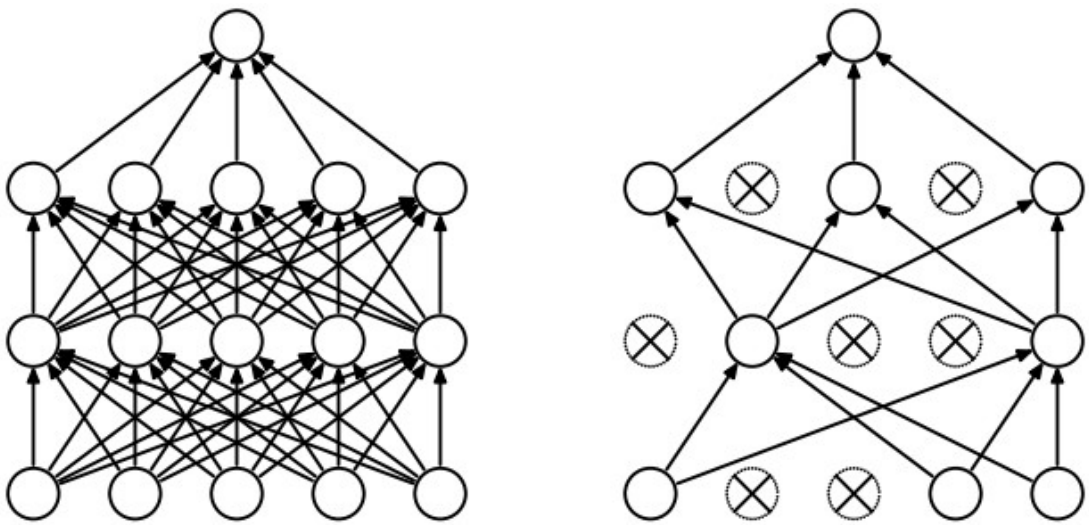
- Since the choice of the learning rate is delicate, variations of the algorithm that are less sensitive to this parameter have been proposed.
- **Nesterov accelerated gradient** : Nesterov (1983) and Sutskever et al. (2013)
- **RMSProp** algorithm, due to Hinton (2012) .

Regularization

- We have already mentioned $\mathbb{L}^2$ or $\mathbb{L}^1$ penalization ; we have also mentioned early stopping.
- For deep learning, the mostly used method is the **dropout** introduced by Hinton et al. (2012).
- With a certain probability p, and independently of the others, each unit of the network is set to 0.
- It is classical to set p to 0.5 for units in the hidden layers, and to 0.2 for the entry layer.
- The computational cost is weak since we just have to set to 0 some weights with probability p.

Dropout

- This method improves significantly the generalization properties of deep neural networks and is now the most popular regularization method in this context.
- The disadvantage is that training is much slower (it needs to increase the number of epochs).



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

FIGURE: Dropout

Convolutional neural networks

- For some types of data, especially for images, multilayer perceptrons are not well adapted.
- They are defined for vectors. By transforming the images into vectors, we loose spatial informations, such as forms.
- The **convolutional neural networks (CNN)** introduced by LeCun (1998) have revolutionized image processing.
- CNN act directly on matrices, or even on tensors for images with three RGB color chanels.

Convolutional neural networks

- CNN are now widely used for image classification, image segmentation, object recognition, face recognition ..

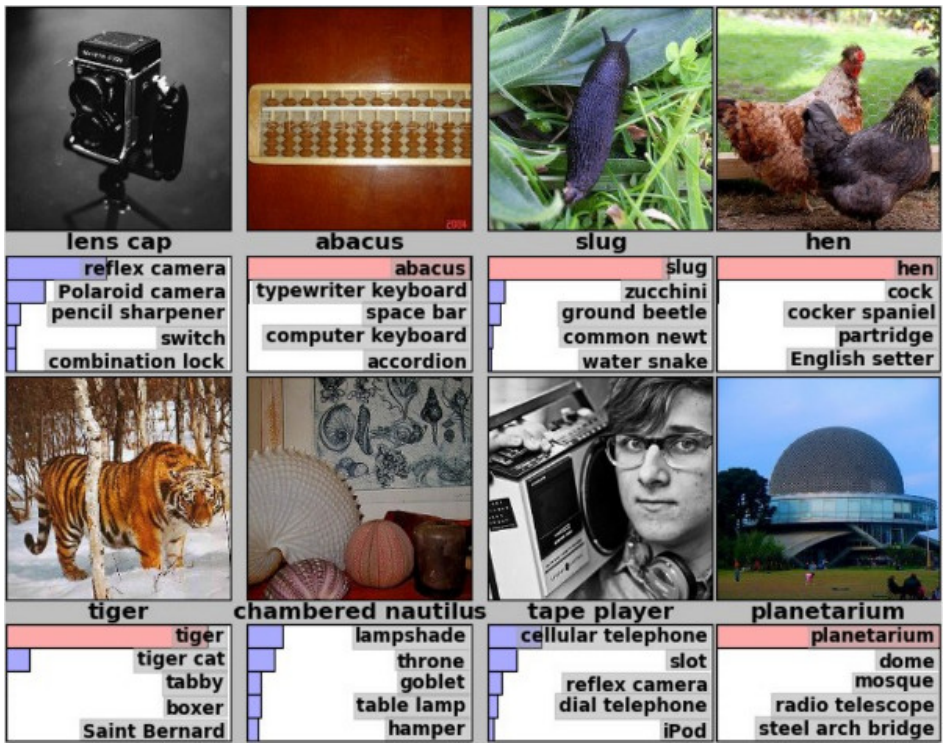


FIGURE: Image annotation

Convolutional neural networks

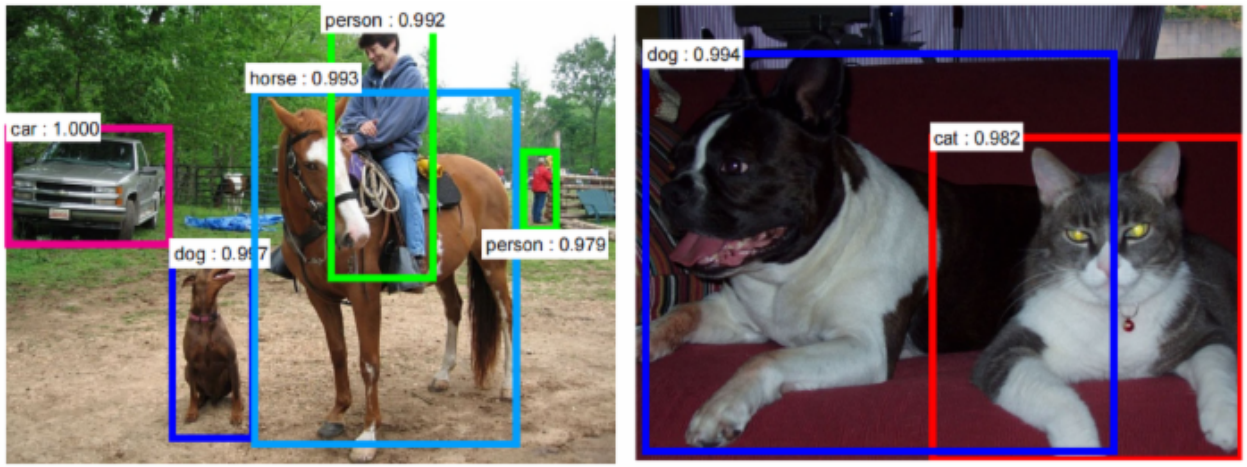


FIGURE: Image Segmentation.

Layers in a CNN

- A Convolutional Neural Network is composed by several kinds of layers, that are described in this section :
  - convolutional layers
  - pooling layers
  - fully connected layers

Convolution layer

- For 2-dimensional signals such as images, we consider the 2D-convolutions : $(K * I)(i,j) = \sum_{m,n} K(m,n) I(i+n, j+m)$.
- K is a convolution kernel applied to a 2D signal (or image) I.

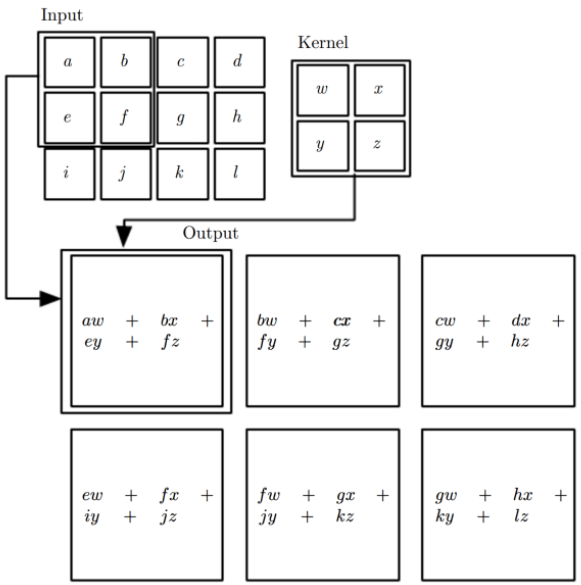


FIGURE: 2D convolution

Convolution layer

- The principle of 2D convolution is to drag a convolution kernel on the image.
- At each position, we get the convolution between the kernel and the part of the image that is currently treated.
- Then, the kernel moves by a number s of pixels, s is called the *stride*.
- When the stride is small, we get redondant information.
- Sometimes, we also add a *zero padding*, which is a margin of size p containing zero values around the image in order to control the size of the output.

Convolution layer

- Assume that we apply C_0 kernels, each of size $k \times k$ on an image.
- If the size of the input image is $W_i \times H_i \times C_i$ (W_i denotes the width, H_i the height, and C_i the number of channels, typically $C_i = 3$), the volume of the output is $W_0 \times H_0 \times C_0$, where C_0 corresponds to the number of kernels that we consider, and

$$W_0 = \frac{W_i - k + 2p}{s} + 1$$

$$H_0 = \frac{H_i - k + 2p}{s} + 1.$$

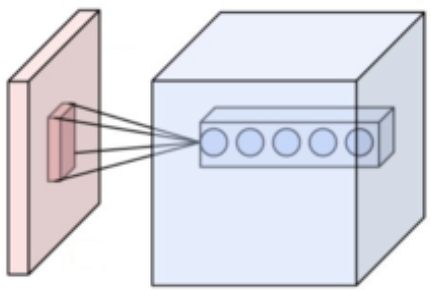
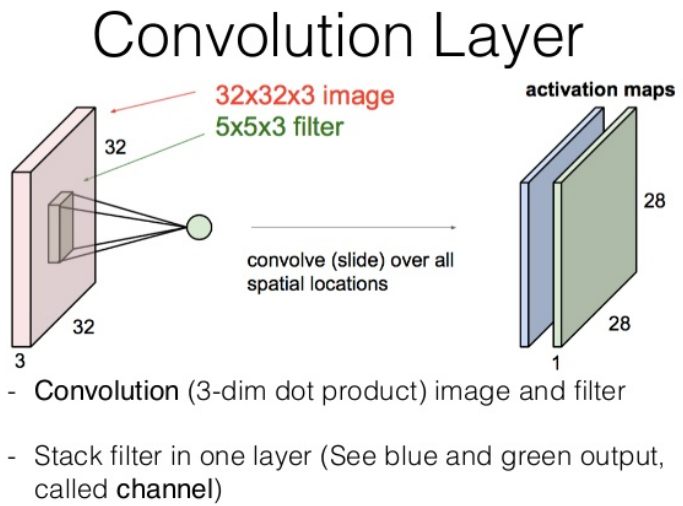


FIGURE: 2D convolution - Units corresponding to the same position but at various depths ; each unit applies a different kernel on the same patch of the

Convolution layer



- If the image has 3 channels and if K_l $(l = 1, \ldots, C_0)$ denote $5 \times 5 \times 3$ kernels, the convolution with the image I with the kernel K_l corresponds to the formula :

$$K_l * I(i,j) = \sum_{c=0}^{2} \sum_{n=0}^{4} \sum_{m=0}^{4} K_l(n,m,c) I(i+n-2, i+m-2, c).$$

Convolution layer

- For images with C^i channels, the shape of the kernel is (k, k, C^i, C^0) where C^0 is the number of output channels.
- The convolution operations are combined with an activation function ϕ (ReLu in general) : if we consider a kernel K of size $k \times k$, if $\boldsymbol{x}$ is a $k \times k$ patch of the image, the activation is obtained by sliding the $k \times k$ window and computing $z(\boldsymbol{x}) = \phi(K * x + b)$, where b is a bias.
- CNN learn the filters (or kernels) that are the most useful for the task that we have to do (such as classification). Several convolution layers are considered : the output of a convolution becomes the input of the next one.

Pooling layer

- CNN also have *pooling* layers, which allow to reduce the dimension, also referred as *subsampling*, by taking the mean or the maximum on patches of the image (mean-pooling or max-pooling).
- Like the convolutional layers, pooling layers acts on small patches of the image.
- If we consider 2×2 patches, over which we take the maximum value to define the output layer, with a stride 2, we divide by 4 the size of the image.

Pooling layer

- Another advantage of the pooling is that it makes the network less sensitive to small translations of the input images.

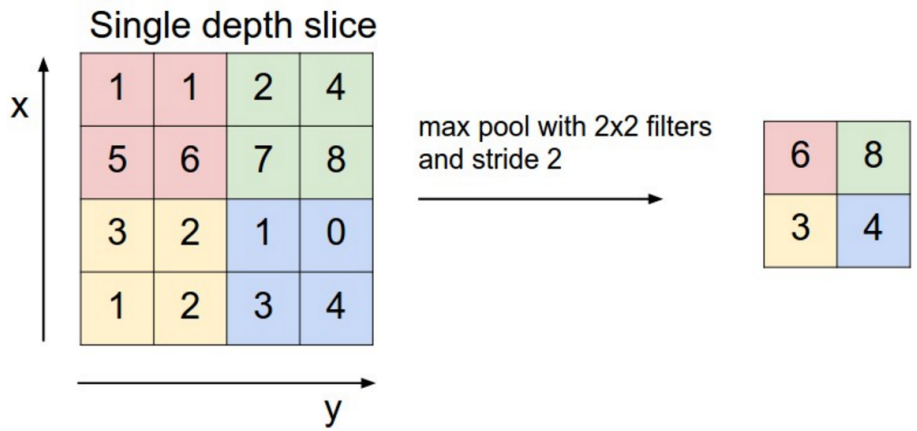


FIGURE: Maxpooling and effect on the dimension

Fully connected layers

- After several convolution and pooling layers, the CNN generally ends with several *fully connected* layers.
- The tensor that we have at the output of these layers is transformed into a vector and then we add several perceptron layers.

Architectures

- We have described the different types of layers composing a CNN.
- We now present how this layers are combined to form the architecture of the network.
- Choosing an architecture is very complex and this is more engineering that an exact science.
- It is therefore important to study the architectures that have proved to be effective and to draw inspiration from these famous examples.
- In the most classical CNN, we chain several times a convolution layer followed by a pooling layer and we add at the end fully connected layers.

Architectures

The **LeNet** network, proposed by the inventor of the CNN, Yann LeCun (1998) is of this type. This network was devoted to digit recognition. It is composed only on few layers and few filters, due to the computer limitations at that time.

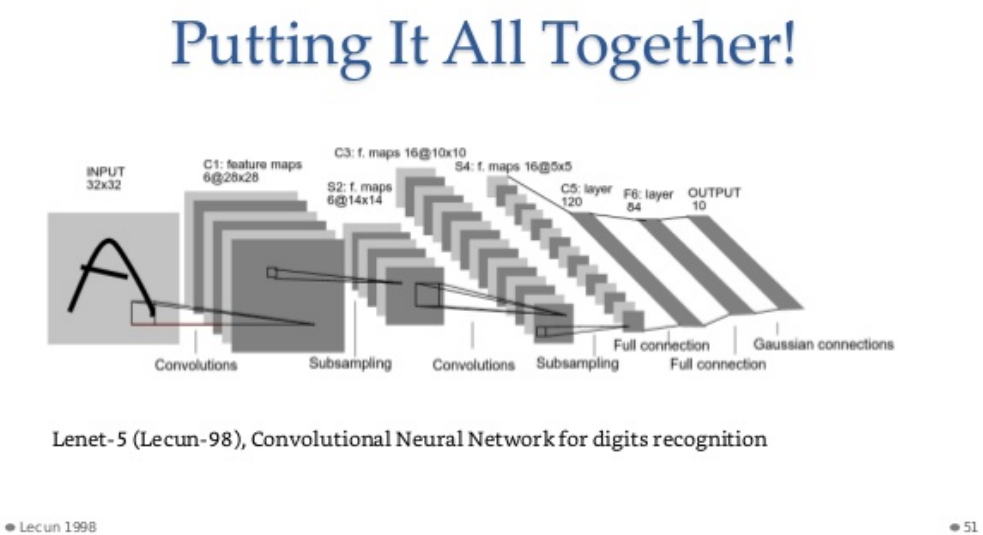


FIGURE: Architecture of the network Le Net. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998)

Architectures

- With the appearance of GPU (Graphical Processor Unit) cards, much more complex architectures for CNN have been proposed, like the network **AlexNet** (Krizhevsky (2012)).
- **AlexNet** won the ImageNet competition devoted to the classification of one million of color images (224×224) onto 1000 classes.
- AlexNet is composed of 5 convolution layers, 3 max-pooling 2×2 layers and fully connected layers.

Architectures

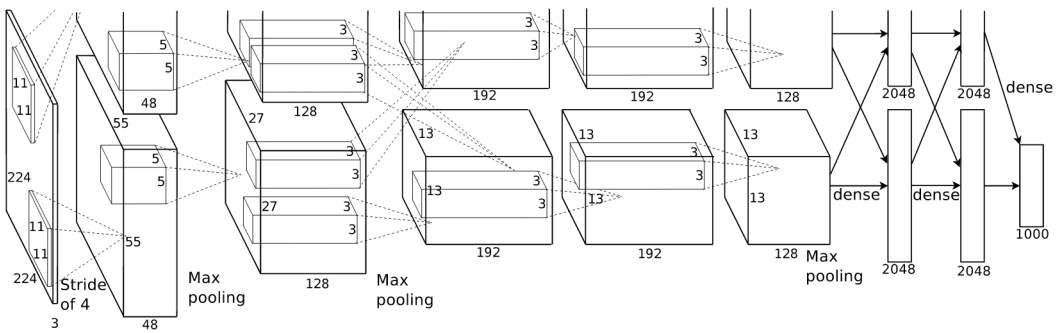


FIGURE: Architecture of the network AlexNet. Krizhevsky, A. et al (2012)

Architectures

Alexnet architecture

Input	227 * 227 * 3			
Conv 1	55*55*96	96	11 *11	filters at stride 4, pad 0
Max Pool 1	27*27*96		3 *3	filters at stride 2
Conv 2	27*27*256	256	5*5	filters at stride 1, pad 2
Max Pool 2	13*13*256		3 *3	filters at stride 2
Conv 3	13*13*384	384	3*3	filters at stride 1, pad 1
Conv 4	13*13*384	384	3*3	filters at stride 1, pad 1
Conv 5	13*13*256	256	3*3	filters at stride 1, pad 1
Max Pool 3	6*6*256		3 *3	filters at stride 2
FC1	4096	4096	neurons	
FC2	4096	4096	neurons	
FC3	1000	1000	neurons	(softmax logits)

Architectures

We present another example.

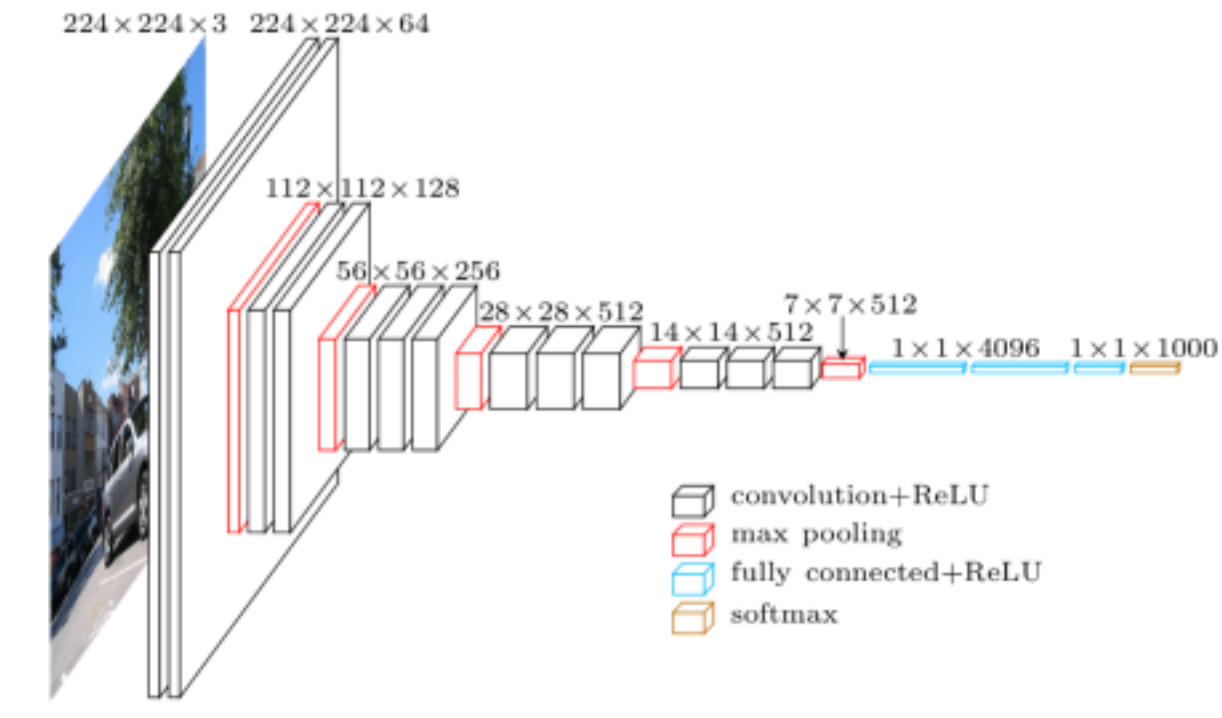


FIGURE: Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition (2014).

Architectures

The network that won the competition in 2014 is the network **GoogLeNet**(Szegedy et al. (2016)), which is a new kind of CNN, not only composed on successive convolution and pooling layers, but also on new modules called *Inception*, which are some kind of network in the network.

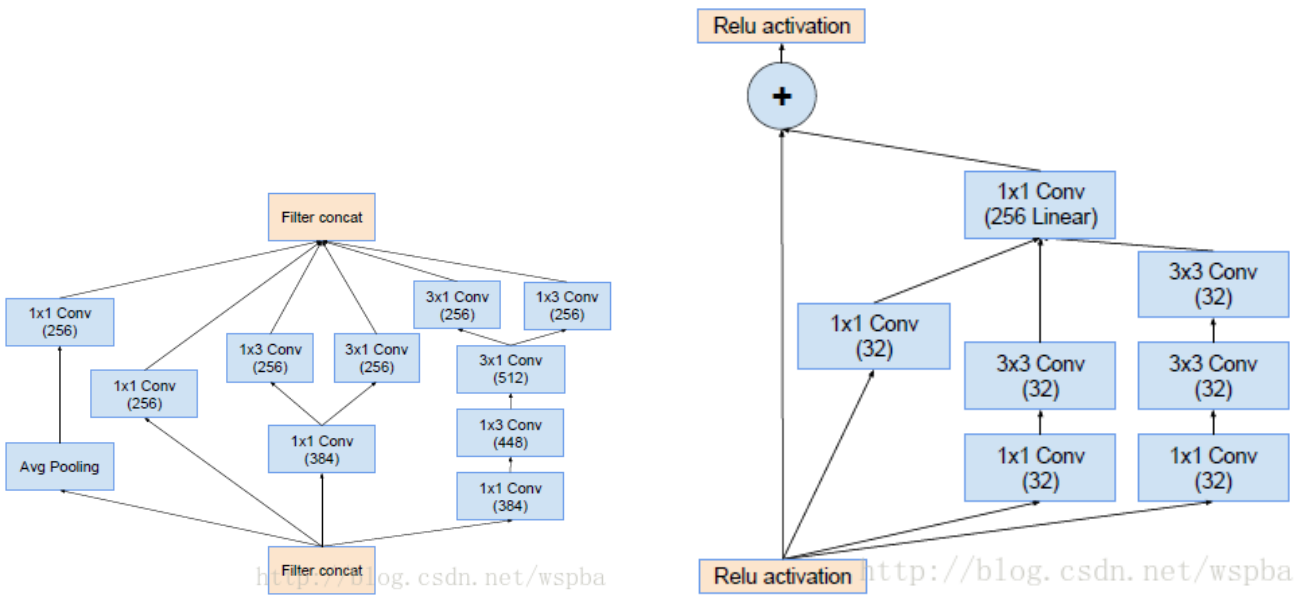


FIGURE: Inception modules, Szegedy et al. (2016)

Architectures

- The most recent innovations concern the **ResNet** networks (see Szegedy 2016).
- The originality of the ResNets is to add a connection linking the input of a layer (or a set of layers) with its output.
- In order to reduce the number of parameters, the ResNets do not have fully connected layers.
- GoogleNet and ResNet are much deeper than the previous CNN, but contain much less parameters.
- They are nevertheless much costly in memory than more classical CNN such as VGG or AlexNet.

Architectures

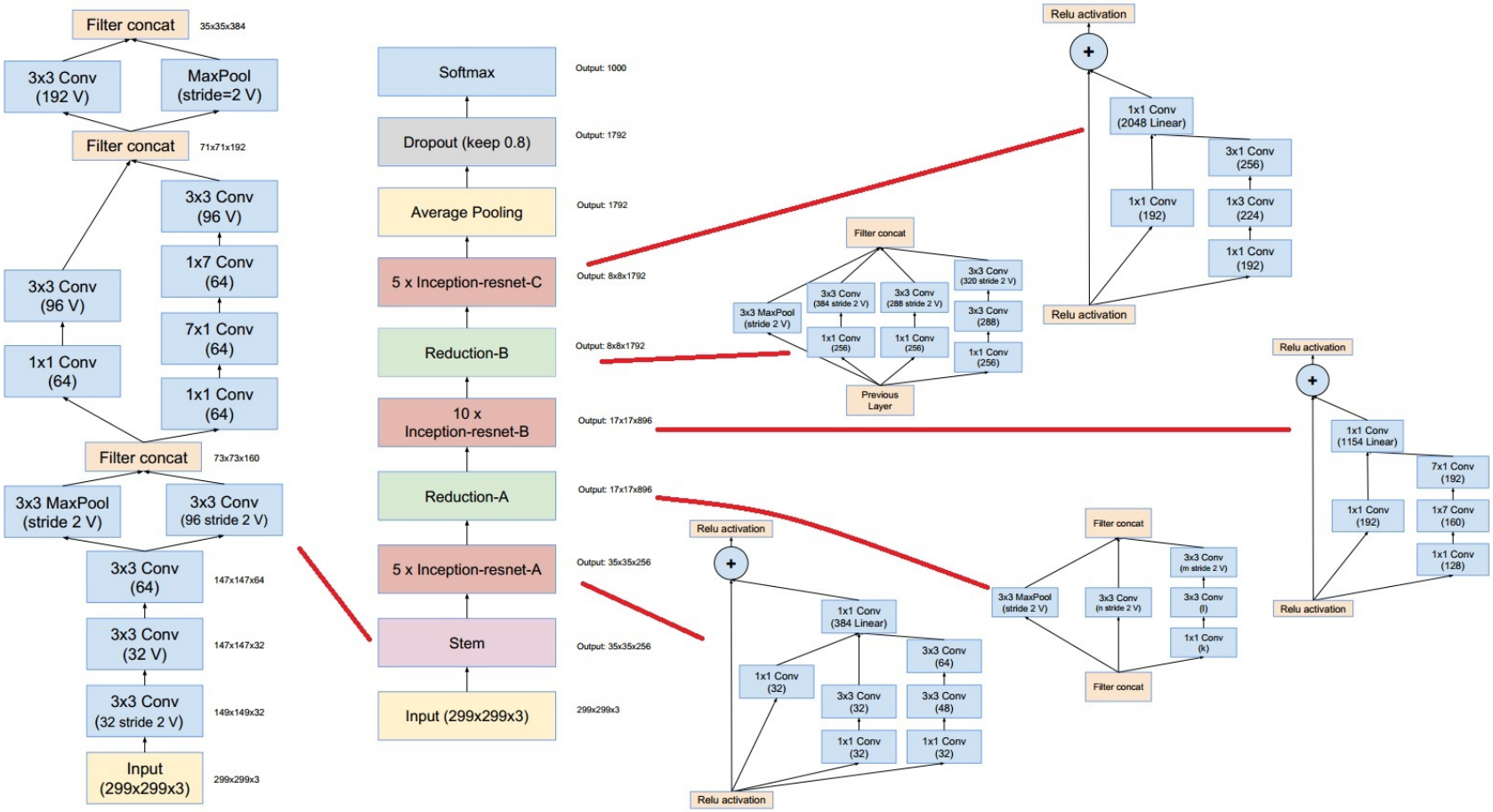


FIGURE: Inception-v4, Inception-resnet (Szegedy, C. et al. , 2016)

Architectures

The Figure shows a comparison of the depth and of the performances of the different networks, on the ImageNet challenge.

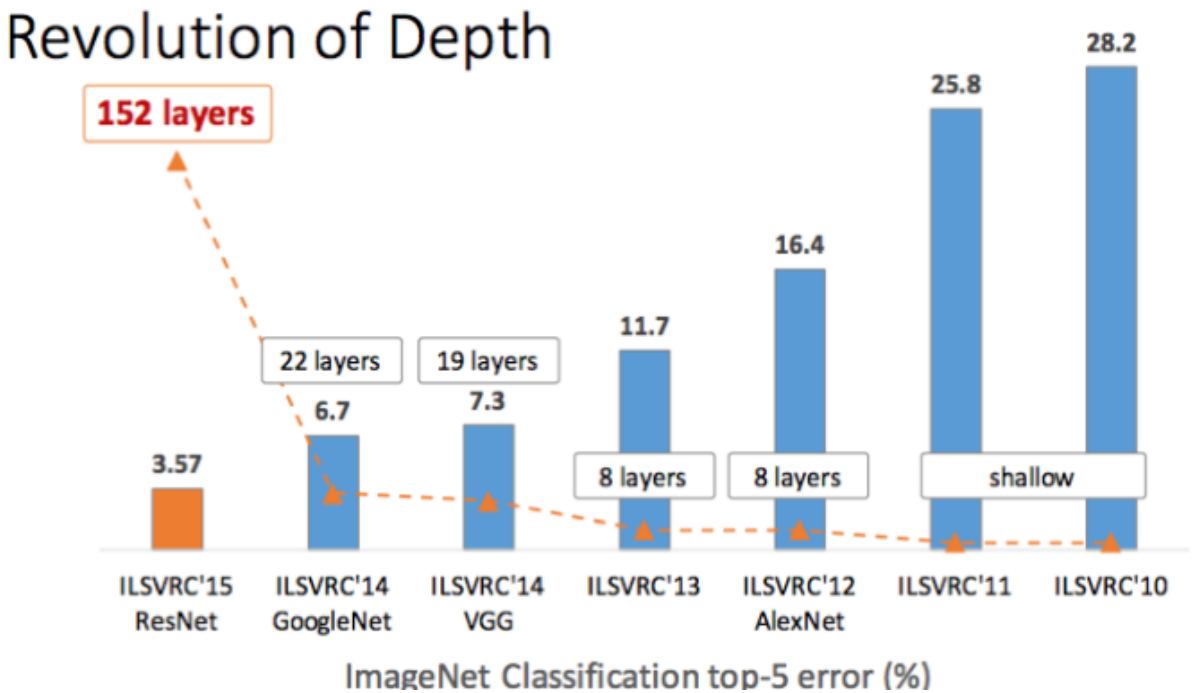


FIGURE: Evolution of the deepth of the CNN and the test error

References

- L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen (1984). *Classification and regression trees*. Chapman et Hall. CRC Press, Boca Raton.
- Giraud C. (2015) *Introduction to High-Dimensional Statistics* Vol. 139 of Monographs on Statistics and Applied Probability. CRC Press, Boca Raton, FL.
- Hastie, T. and Tibshirani, R. and Friedman, J, (2009), *The elements of statistical learning : data mining, inference, and prediction*, Springer.
- R.E. Shapire and Y. Freund *Boosting : Fundation and Algorithms* MIT Press, 2012

References

The main references for the course on Neural networks are :

- Ian Goodfellow, Yoshua Bengio and Aaron Courville *Deep learning*
- Bishop (1995) *Neural networks for pattern recognition*, Oxford University Press.
- Hugo Larochelle (Sherbrooke) :
  http ://www.dmi.usherb.ca/ larocheh/
- Charles Ollion et Olivier Grisel *Deep learning course*
  https ://github.com/m2dsupsdlclass/lectures-labs