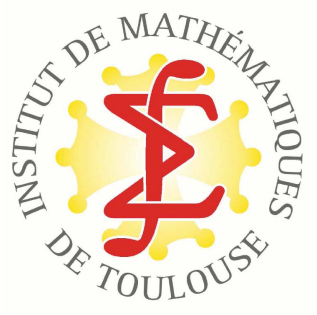


Institut de Mathématiques de Toulouse, INSA Toulouse

Supervised Learning- Part II

Data Science
December 2020

Béatrice Laurent-Olivier Roustant- Sébastien Gerchinovitz

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

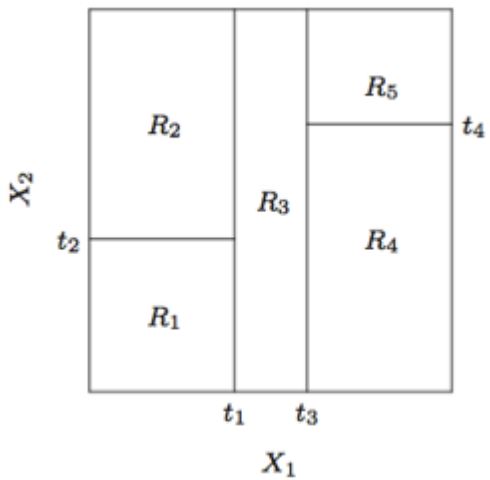
- Classification And Regression Trees (CART)
- Bagging, Random Forests
- Boosting
- Neural networks, Introduction to deep learning

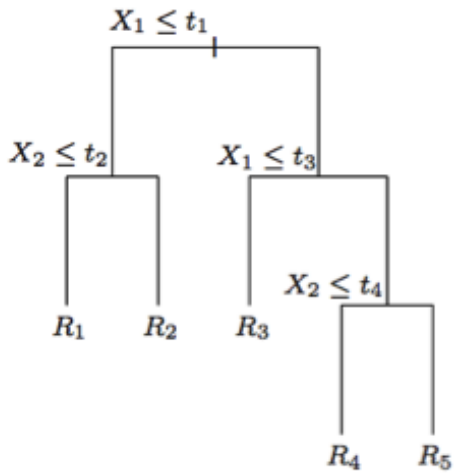
Classification And Regression Trees

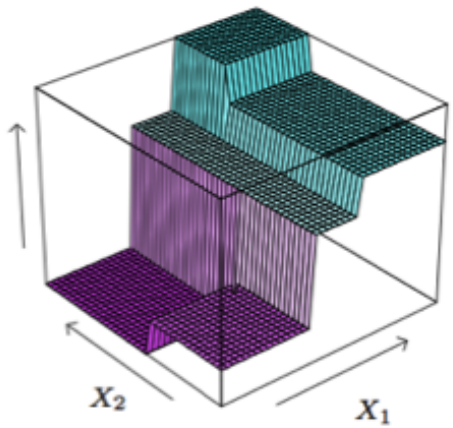
Introduction

- 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 regression tree

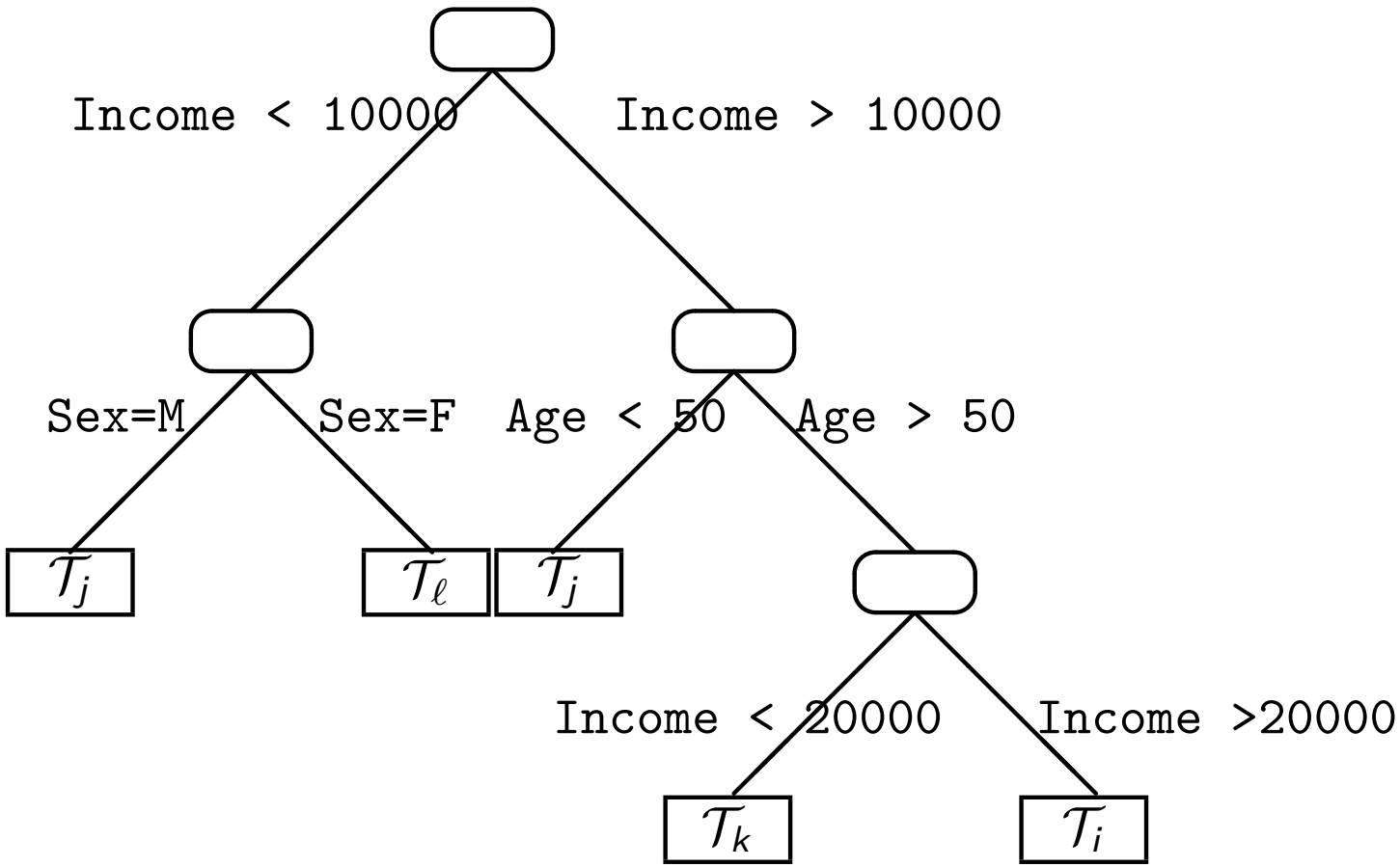






Source : Hastie, Tibshirani, Friedman (2019), "The elements of statistical learning"

Example of binary classification tree



Principles for constructing a tree

- Recursive binary split
  $\rightarrow$ Split a region in two, then split subregions in two, then ...
- Splits are defined by one variable
  $\rightarrow$ Very easy numerically : p optimizations in 1-dimension
- Clustering idea
  $\rightarrow$ Find a split that give the most homogeneous groups

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

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 a leaf 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

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 a majority vote.

Constructing regression trees

For a given region (node) κ with size $|\kappa|$, define the heterogeneity by :

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

Splitting procedure

For a variable x_j, and a split candidate t, define left and right subregions

$$\kappa_L(t,j) = \{x_j \leq t\}, \qquad \kappa_R(t,j) = \{x_j > t\}.$$

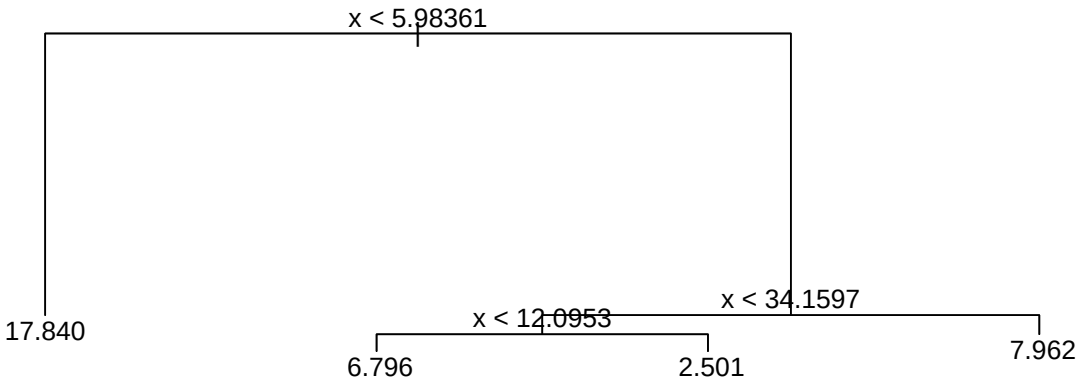
Find (j,t) in order to minimize the intra-class variance

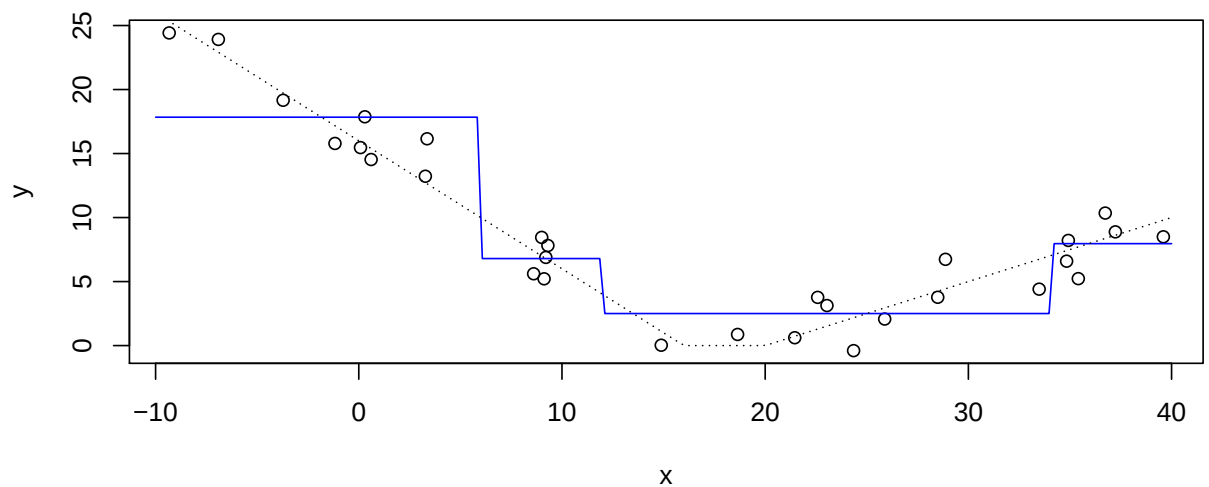
$$J(j,t) = D_{\kappa_L(t,j)} + D_{\kappa_R(t,j)},$$

or equiv. to maximize the decrease in heterogeneity (inter-class variance)

$$D_\kappa - J(j,t)$$

Illustration in 1 dimension





Constructing classification trees

This is the same procedure, with specific notions of heterogeneity

Heterogeneity measures in classification

p_κ^ℓ : proportion of the class $\mathcal{T}_\ell$ of Y in the node κ.

- Shannon Entropy

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

Maximal in $(\frac{1}{m}, \ldots, \frac{1}{m})$, minimal in $(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$
(by continuity, we assume that $0 \log(0) = 0$)

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

Illustration with two classes ($m = 2$)

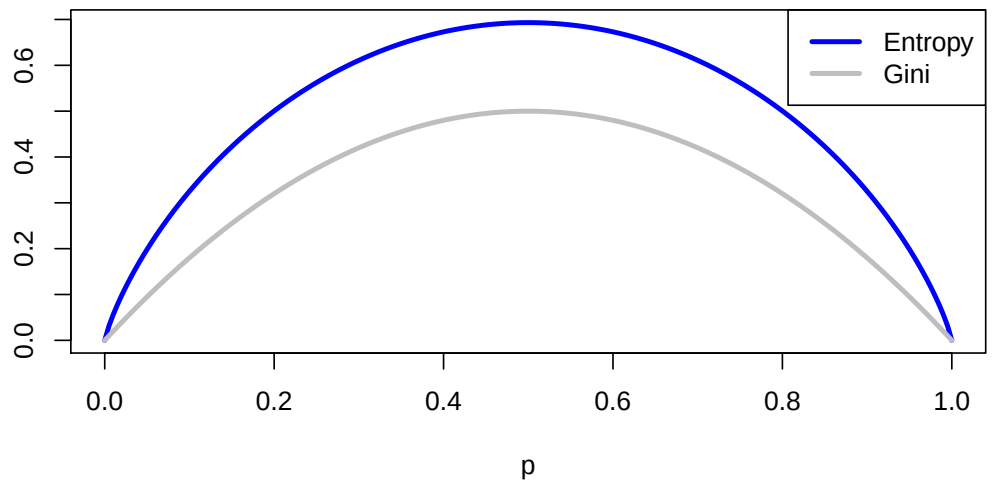
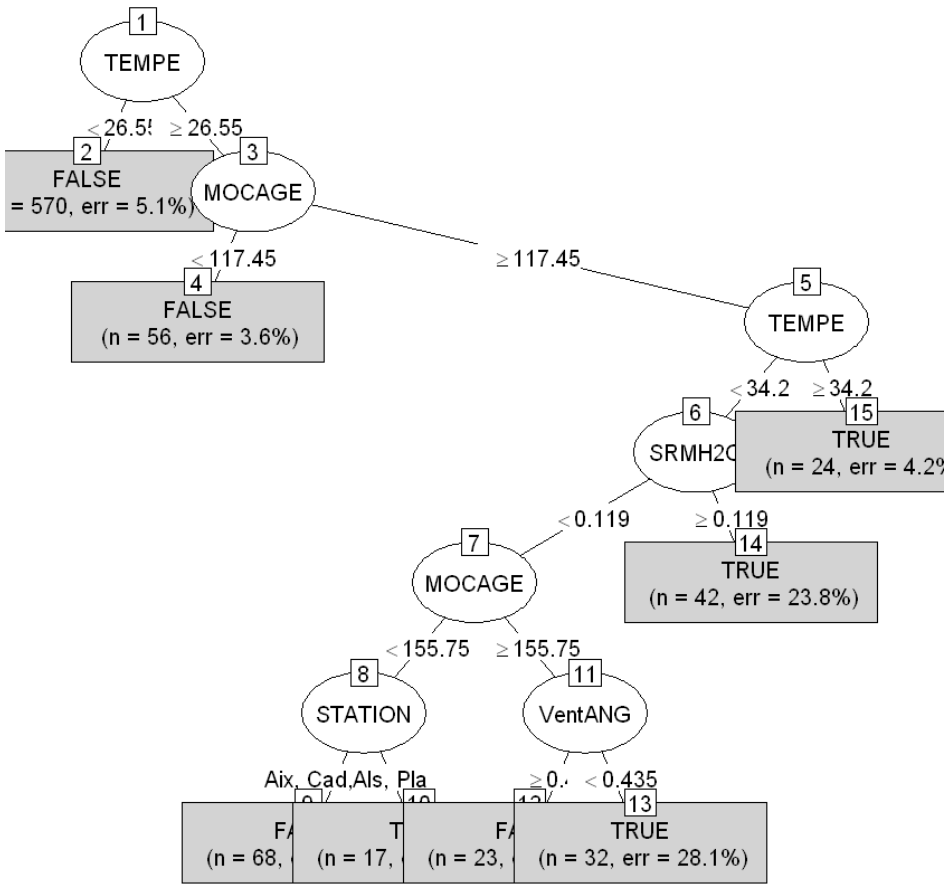


FIGURE – Heterogeneity criterions for classification. Both are minimal for $p = 0$ or $p = 1$, and maximal for $p = 1/2$.

Example for Ozone data



Ozone : Classification tree pruned by cross-validation

Stopping rule, pruning, optimal tree

- We need a tradeoff between maximal tree (overfits) and the constant tree (too rough)
- There exists a nice theory to find an optimal tree, minimizing prediction error penalized by complexity (number of leaves)
- When aggregating trees (random forest), simpler procedures are often prefered (see why after), e.g. fixing the number of leaves

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 :

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

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

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$^{(\star)}$ aggregating : Breiman, 1996
- Random forests : Breiman, 2001
- Allows to aggregate any modelisation method
- Efficient methods : Fernandez-Delgado et al. (2014), *Kaggle*

$^{(\star)}$ *bootstrap = sampling with replacement*

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

Bagging - Principle

Bootstrap AGGregatING

- Variance reduction : by aggregating independent predictions
  - Aggregation : average (regression), majority vote (classification)
- Bootstrap trick : get new data from themselves by resampling !
  - Caution : new data remain (slightly) dependent on the initial ones

Bagging - Introductive example

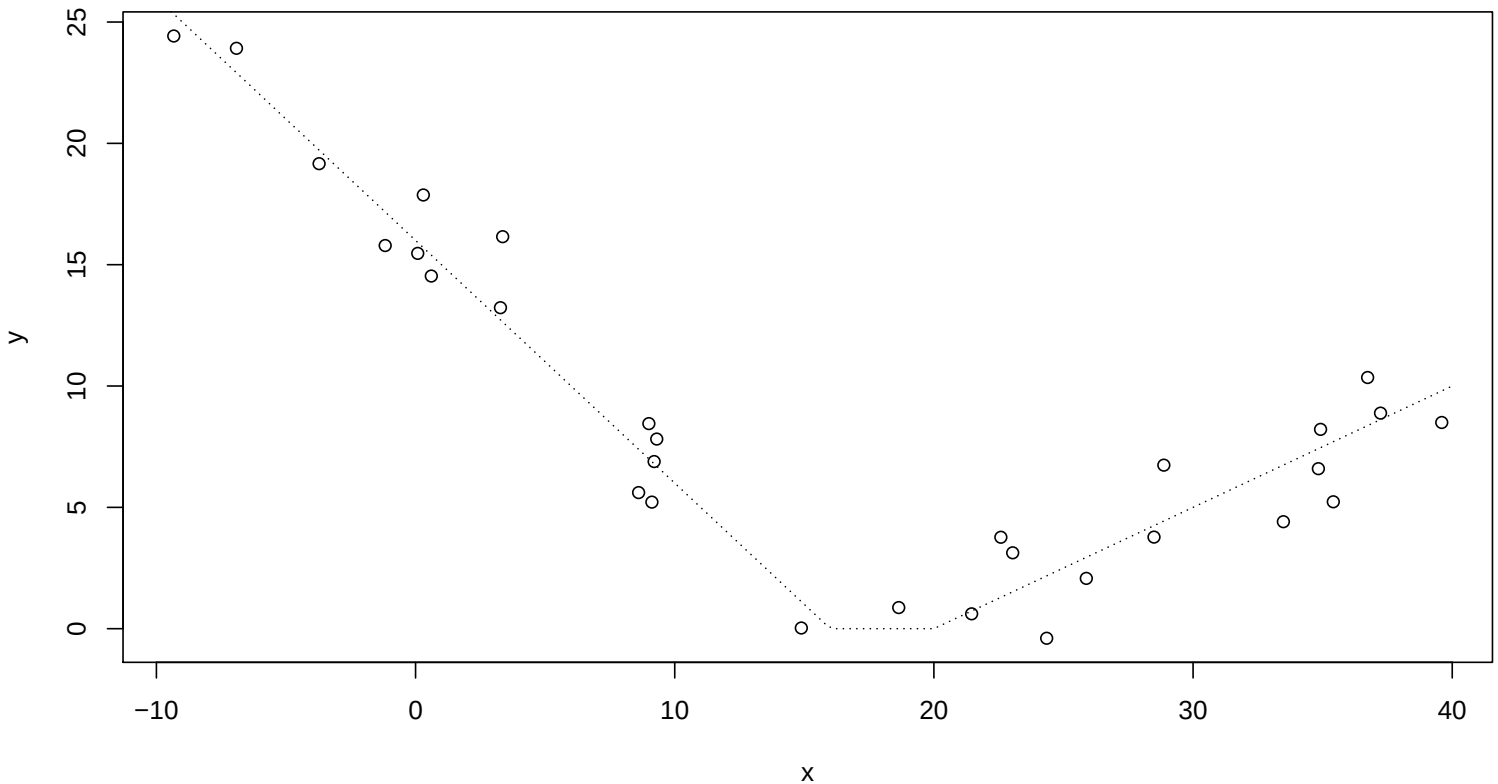


FIGURE – Original data

Bagging - Introductive example

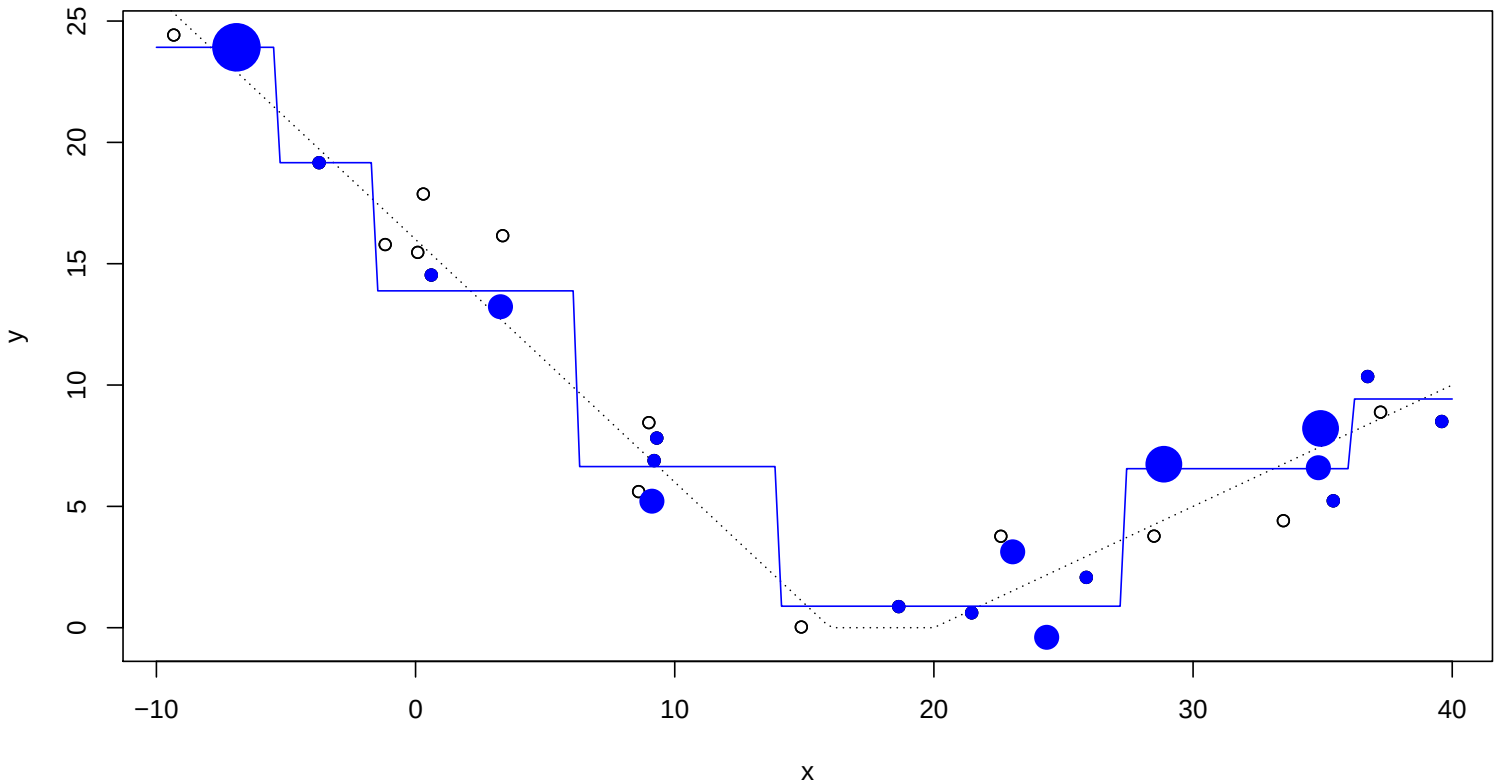


FIGURE – Bootstrap sample n^o1 (in blue), and corresp. prediction with tree. The point size is proportional to the number of replicates.

Bagging - Introductive example

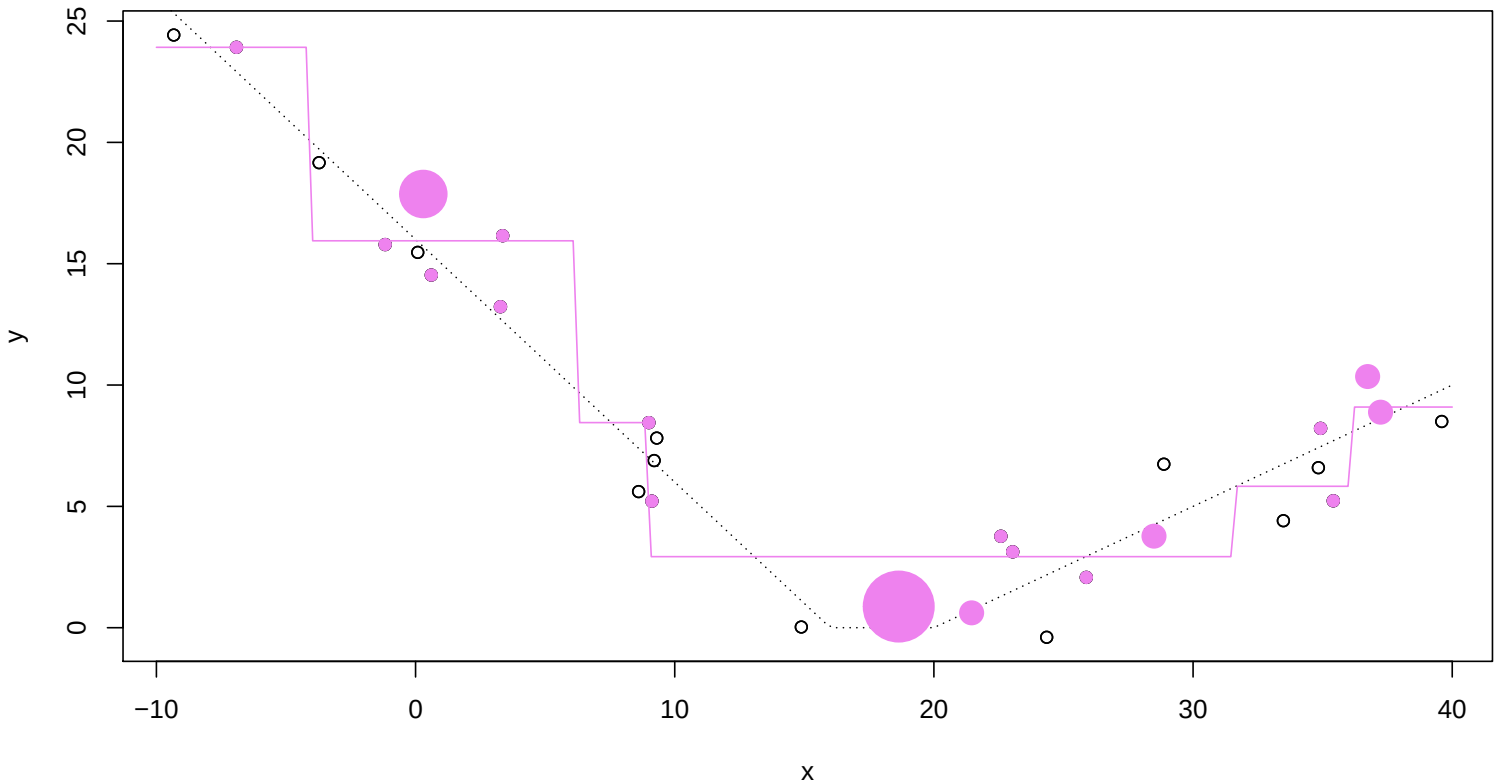


FIGURE – Bootstrap sample $n^o 2$ (in violet), and corresp. prediction with tree. The point size is proportional to the number of replicates.

Bagging - Introductive example

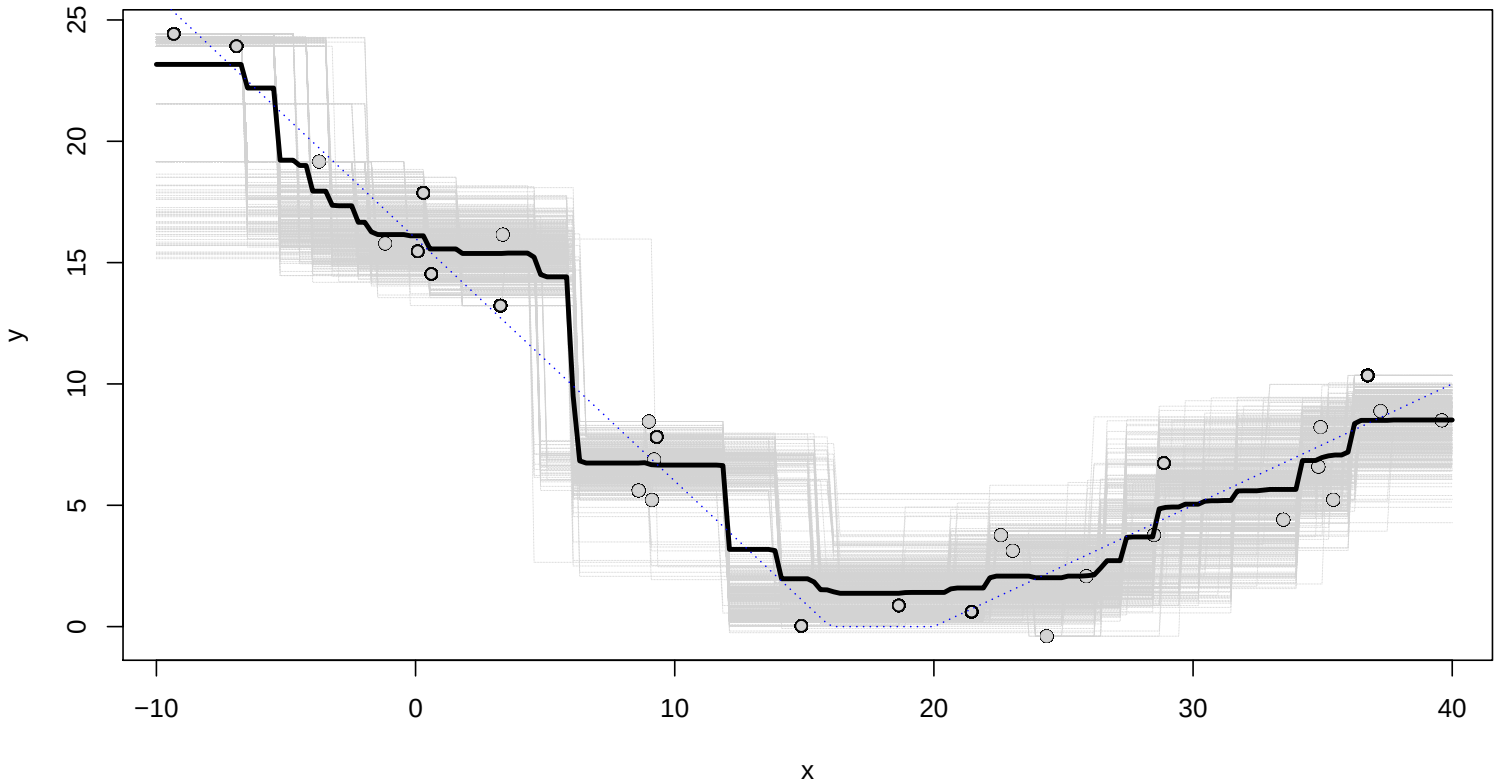


FIGURE – 500 bootstrap samples (grey), corresp. predictions with tree, and their average (bold line).

Bagging - Pause

Physical experiment !

Experiment yourself the bootstrap procedure by resampling "by hand"

Question : Choose a number between 1 and N (number of participants). What is the probability that your number does not appear in the boostrap sample ?

Bagging - Out-Of-Bag data

Out-Of-Bag (OOB) data

For each bootstrap sample :

- Let $U_1^\star, \ldots, U_N^\star$ be random variables representing the boostrapped indices. The probability that a given data z_i is not chosen is :

$$\mathbb{P}\left(z_{U_1^\star} \neq z_i, \ldots, z_{U_N^\star} \neq z_i\right) = \left(1 - \frac{1}{N}\right)^N \underset{N \to +\infty}{\longrightarrow} e^{-1} \approx 0.367$$

- The non-chosen data are called Out-Of-Bag (OOB). They can be used as a test set inside the bootstrap loop

The OOB error is obtained by averaging prediction errors over OOB data

Bagging - Out-Of-Bag data

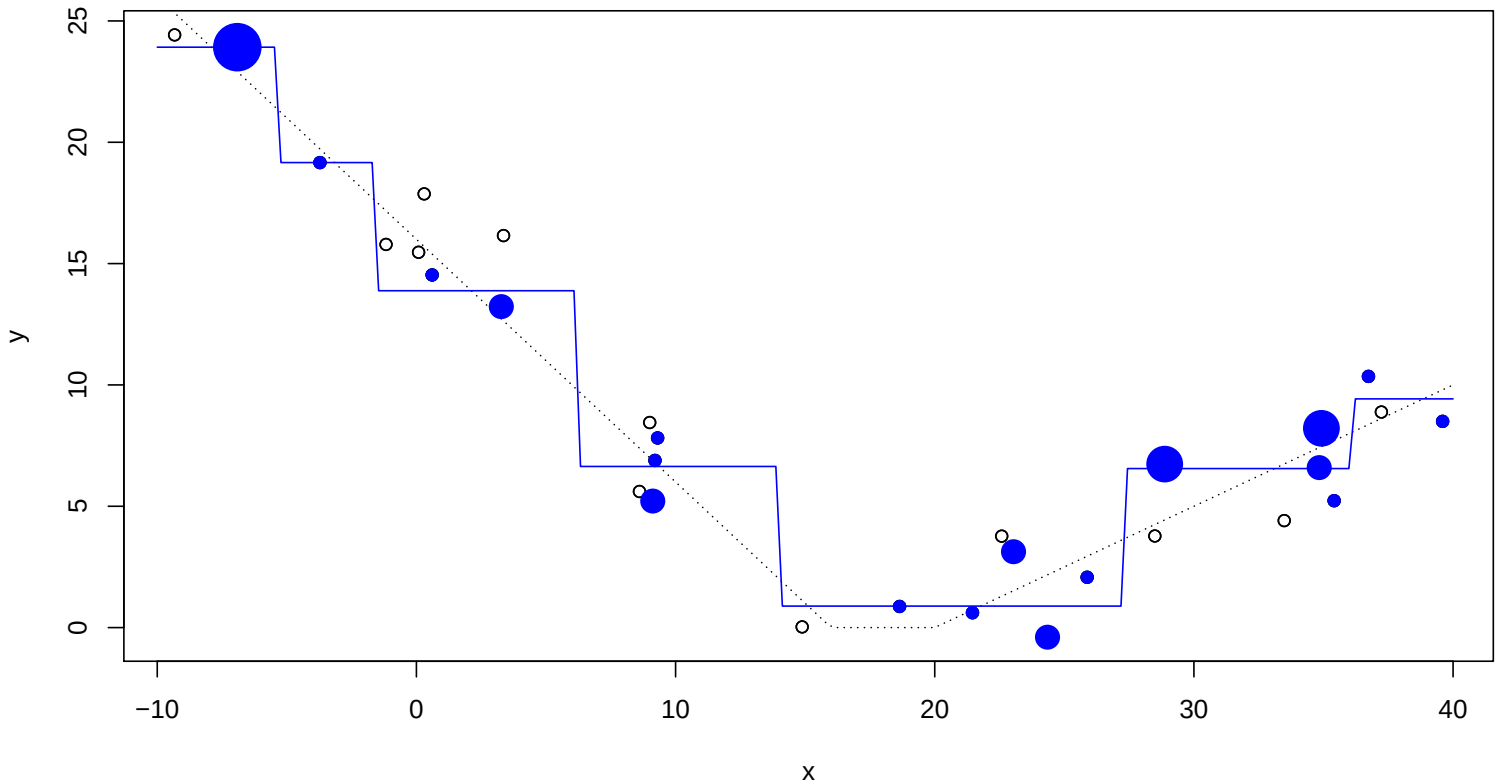


FIGURE – Residuals for the OOB bootstrap sample $n^o 1$ (red bars).

Bagging - Out-Of-Bag data

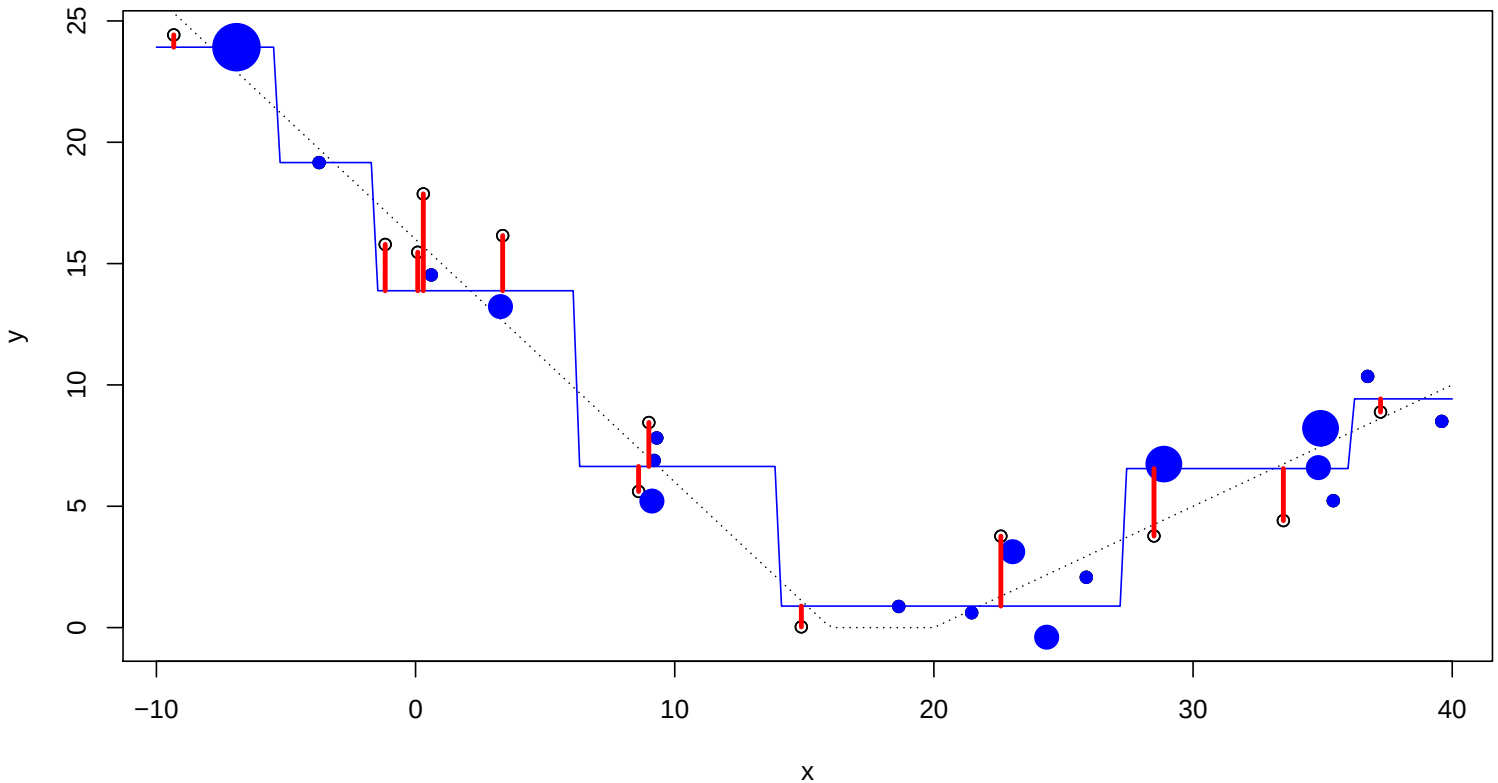


FIGURE – Residuals for the OOB bootstrap sample $n^o 1$ (red bars).

Bagging - Out-Of-Bag data

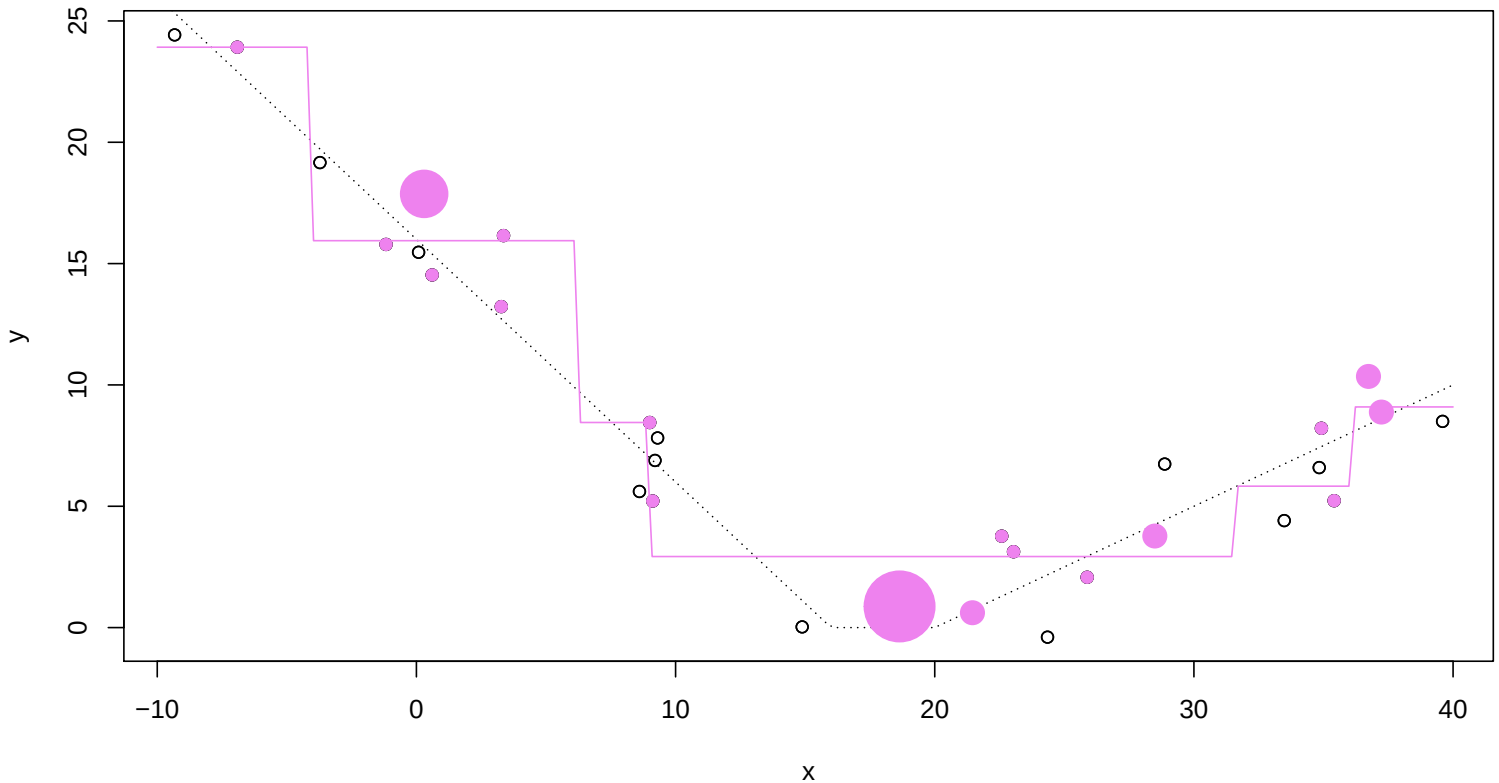


FIGURE – Residuals for the OOB bootstrap sample $n^o 2$ (red bars).

Bagging - Out-Of-Bag data

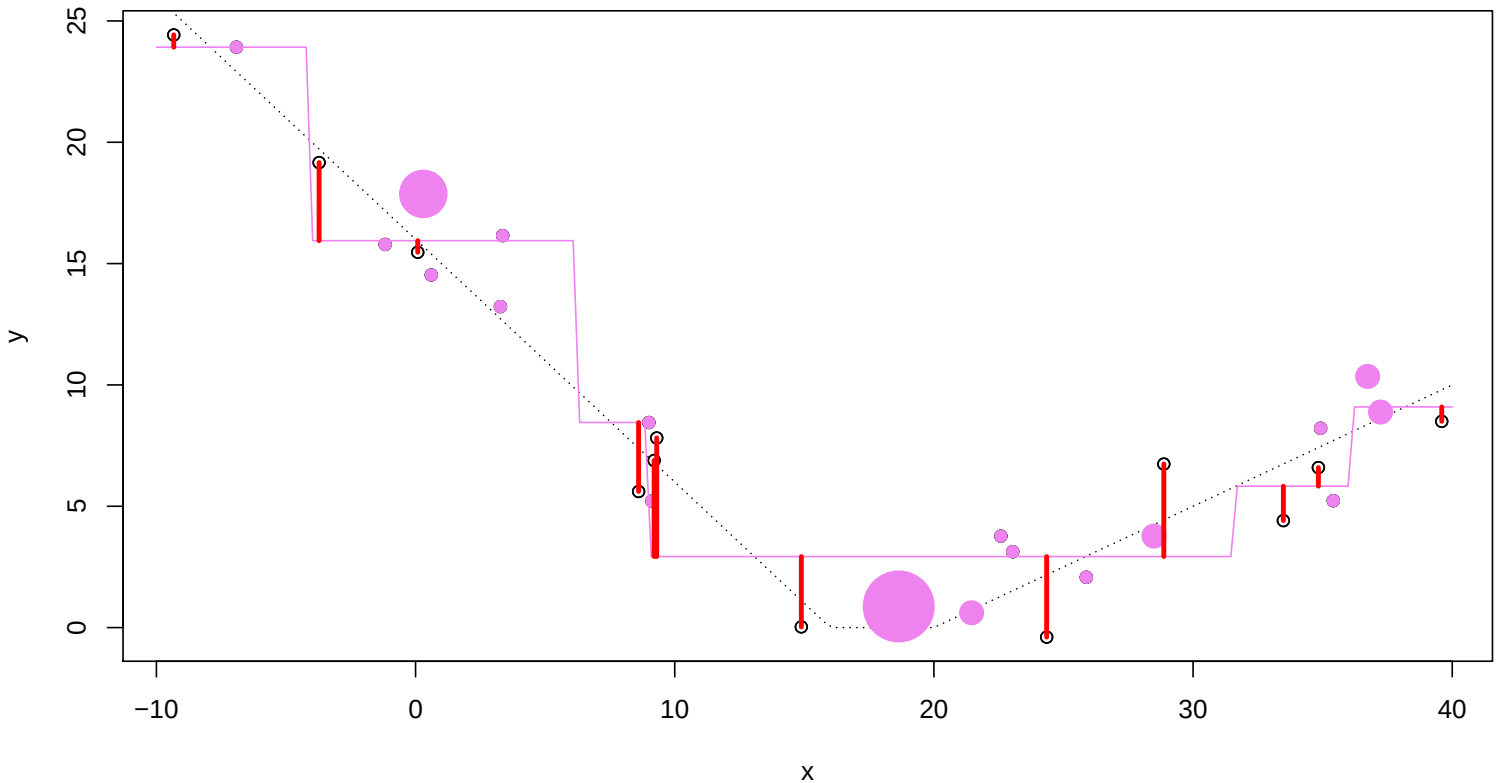


FIGURE – Residuals for the OOB bootstrap sample $n^o 2$ (red bars).

Bagging - Theory

Framework and notations

- 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 avoid 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 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

Interpretation - Variable importance

How can we quantifiy the importance of a variable X^j in random forest ?

Decrease in heterogeneity

Average the decrease of heterogeneity when X^j is chosen as a split.

- Mean Decrease Accuracy
- Mean Decrease Gini

Permutation of variables

Compute the OOB error for the subsample of OOB data.
Compare with the OOB error when permuting at random the values of the variable X^j in the learning sample (but keeping the output Y unchanged).

Example on ozone data

```
> library(randomForest)
> rf.reg <- randomForest(O3obs ., data = datappr, xtest = datestr[, -2],
    ytest = datestr[, "O3obs"], ntree = 500, do.trace = 50, importance = TRUE)
```

Tree	Out -of - bag MSE	%Var(y)	Test set MSE	%Var(y)
50	697.9	40.77	568.5	36.75
100	689.5	40.28	555.9	35.93
150	683.8	39.95	563.2	36.41
200	685.4	40.04	561	36.27
250	678.2	39.62	564.2	36.47
300	675.1	39.44	569.2	36.79
350	676.8	39.54	572.8	37.02
400	674.3	39.39	571.4	36.93
450	673.9	39.37	571.5	36.94
500	674.3	39.39	569.6	36.82

```
> round(importance(rf.reg), 2)

           %IncMSE    IncNodePurity
JOUR         1.98         11011.79
MOCAGE      41.46        388657.27
TEMPE       51.73        409018.57
STATION     21.73         75350.42
VentMOD     12.95         91387.20
VentANG     18.81        124908.37
SRMH2O      16.76        114463.05
LNO2         7.73         84152.34
LNO         10.04         74387.32
```

Details (from R help file of function `importance`)

"The first measure [%IncMSE] is computed from permuting OOB data : For each tree, the prediction error on the out-of-bag portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences.

The second measure [IncNodePurity] is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares."

rf.reg

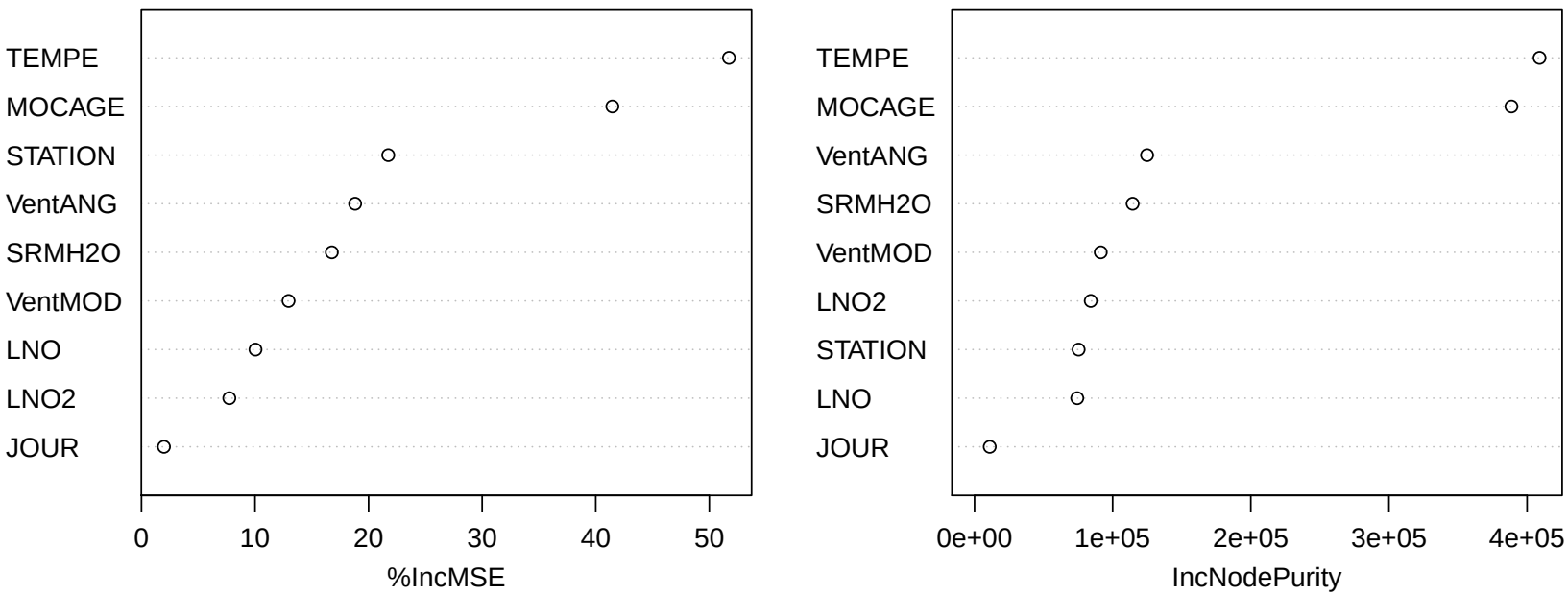


FIGURE – Variable importance plot, returned by the R function `importance`

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 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

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

- 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
$$\beta_m = \frac{1}{2} \log \left(\frac{1 - \widehat{\mathcal{E}_m}(j_m)}{\widehat{\mathcal{E}_m}(j_m)} \right).$$

AdaBoost

AdaBoost algorithm

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

$$j_m = \underset{j=1,\ldots,p}{\operatorname{argmin}} \widehat{\mathcal{E}_m}(j), \beta_m = \frac{1}{2} \log \left(\frac{1 - \widehat{\mathcal{E}_m}(j_m)}{\widehat{\mathcal{E}_m}(j_m)} \right).$$

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \exp(-y_i \beta_m f_{j_m}(\mathbf{x}_i)) \text{ for } i = 1, \ldots, n \\ &= \frac{1}{n} \exp(-y_i \hat{f}_{m-1}(\mathbf{x}_i)) \exp(-y_i \beta_m f_{j_m}(\mathbf{x}_i)) \\ &= \frac{1}{n} \exp(-y_i \hat{f}_m(\mathbf{x}_i)) \end{aligned}$$

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

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 approximated 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

$$f_m(\boldsymbol{x}_i) = f_{m-1}(\boldsymbol{x}_i) - \gamma_m \frac{\partial l(y_i, f_{m-1}(\boldsymbol{x}_i))}{\partial f_{m-1}(\boldsymbol{x}_i)}.$$

- Search for a best step in the descent γ :

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

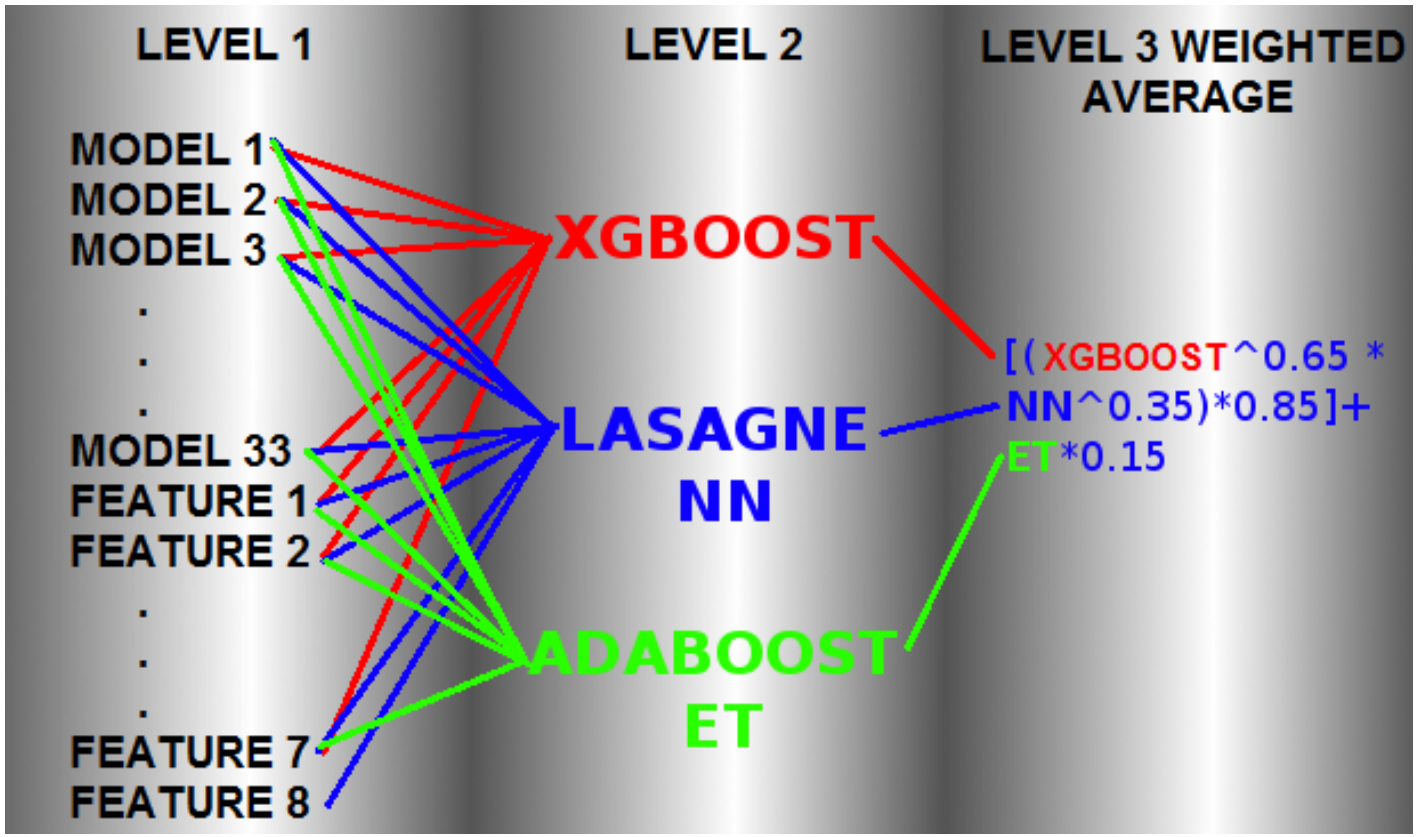
GBM in regression : algorithm

- 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 $r_{m,i} = -\left[\frac{\partial l(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f=f_{m-1}}$; $i = 1, \ldots, n$
  - Adjust a regression tree δ_m to $(\mathbf{x}_i, r_{m,i})$
  - Calculate $\gamma_m = \arg min_\gamma \sum_{i=1}^n l(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma\delta_m(\boldsymbol{x}_i))$
  - Update : $\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \gamma_m\delta_m(\boldsymbol{x})$
- Result : $\widehat{f}_M(\boldsymbol{x}_0)$

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 leaf
- Ω Mix of l_1 and l_2 penalization

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).

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

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

Several activation functions can be considered.

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

The following Figure represents the activation function described above.

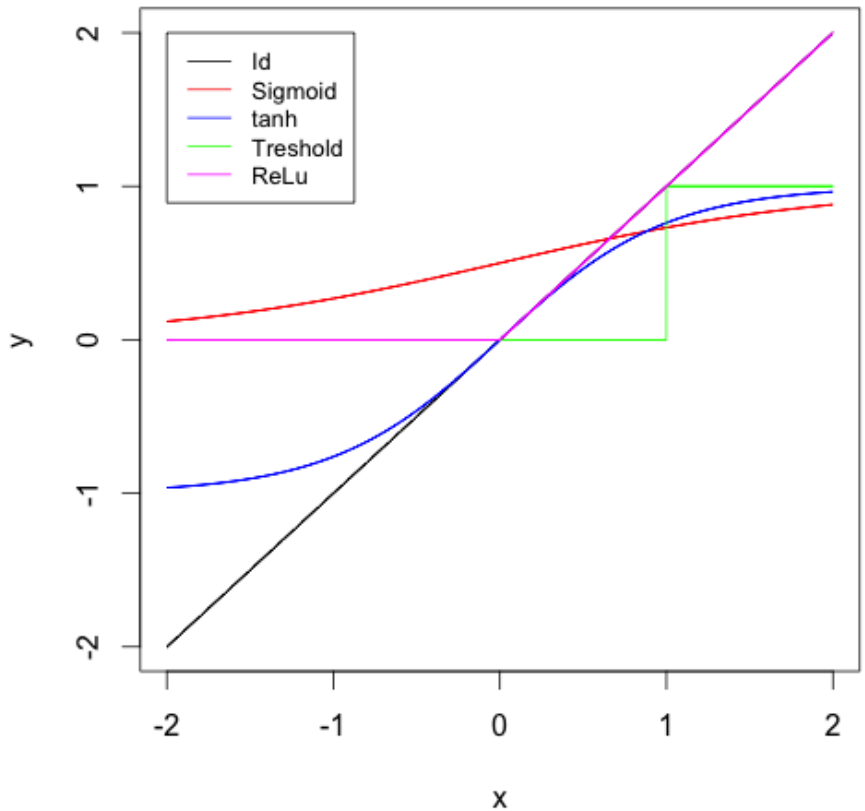
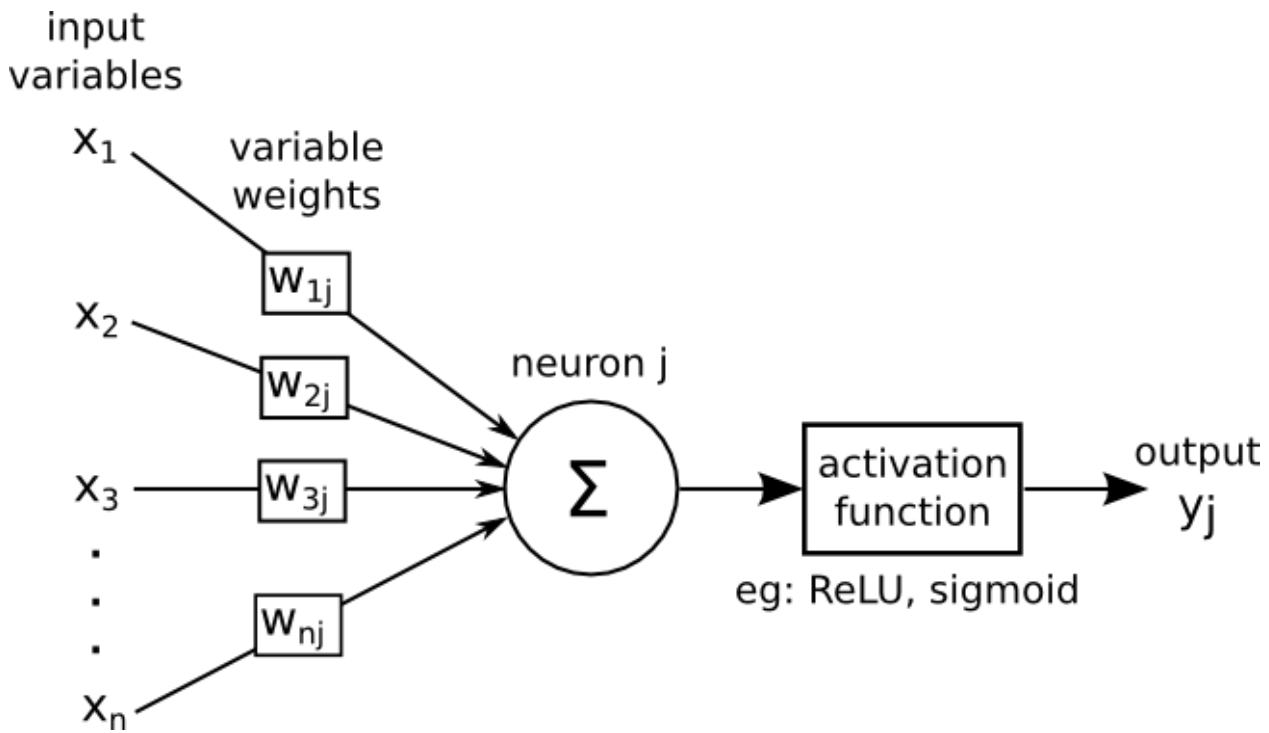


FIGURE – Activation functions

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.

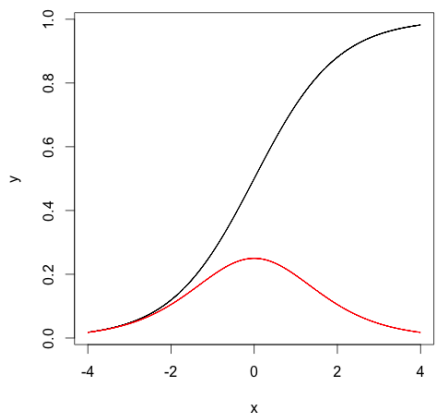


FIGURE – Sigmoid function (in black) and its derivatives (in red)

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 piecewise linear and has many of the properties that make linear model easy to optimize with gradient-based methods.

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

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.

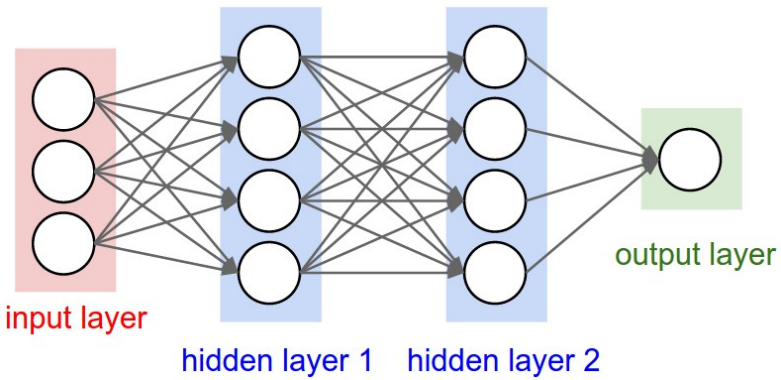


FIGURE – A basic neural network.

Multilayer perceptron

- The output of a neuron can also be the input of a neuron of the same layer or of neuron of previous layers : this is the case for recurrent neural networks.

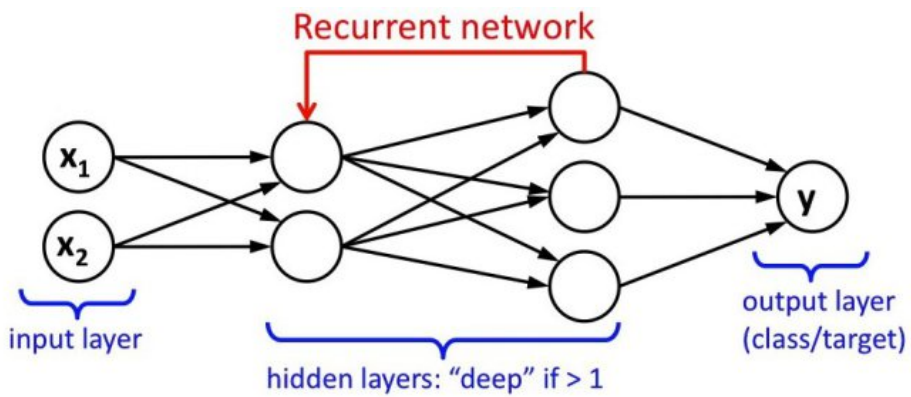


FIGURE – A recurrent neural network

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 (the identity 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 $k - 1$.

Universal approximation theorem

Theorem

Let ϕ be a bounded, continuous and non decreasing (activation) function. Let K_d be some compact set in $\mathbb{R}^d$ and $\mathcal{C}(K_d)$ the set of continuous functions on K_d. Let $f \in \mathcal{C}(K_d)$. Then for all $\varepsilon > 0$, there exists $N \in \mathbb{N}$, real numbers v_i, b_i and $\mathbb{R}^d$-vectors w_i such that, if we define

$$F(x) = \sum_{i=1}^{N} v_i \phi(\langle w_i, x \rangle + b_i)$$

then we have

$$\forall x \in K_d, |F(x) - f(x)| \leq \varepsilon.$$

Universal approximation theorem

- This theorem due to Hornik (1991) shows that any bounded and regular function $\mathbb{R}^d \to \mathbb{R}$ can be approximated at any given precision by a neural network with one hidden layer containing a finite number of neurons, having the same activation function, and one linear output neuron.
- This theorem is interesting from a theoretical point of view. From a practical point of view, this is not really useful since the number of neurons in the hidden layer may be very large. The strength of deep learning lies in the deep (number of hidden layers) of the networks.

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

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 $(X_i, Y_i)_{1 \leq i \leq n}$.
- As usual, the estimation is obtained by minimizing a loss function, generally with a gradient descent algorithm.
- We first have to choose the loss function between the output of the model $f(x, \theta)$ for an entry x and the target y.

Loss functions

- If the regression model, we generally consider the loss function associated to the $\mathbb{L}_2$ norm :

$$\ell(f(x,\theta),y) = \|y - f(x,\theta)\|^2.$$

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

$$\begin{aligned} \ell(f(x,\theta),y) &= -\left[y \log(f(x,\theta)) + (1-y)\log(1-f(x,\theta))\right] \\ &= -\log(P_\theta(Y = y/X = x)) \end{aligned}$$

Loss functions

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

$$\ell(f(x,\theta),y) = -\left[\sum_{j=1}^{k} \mathbb{1}_{y=j} \log P_\theta(Y = j/X = x)\right].$$

- Ideally we would like to minimize the classification error, but it is not smooth, this is why we consider the cross-entropy.

Penalized empirical risk

- Denoting θ the vector of parameters to estimate, we consider the expected loss function

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

  associated to the 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 $\Omega(\theta)$:

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

- One can also consider $\mathbb{L}^1$ regularization (LASSO penalty), 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 often used.
- In order to compute the gradient, a clever method, called **Backpropagation algorithm** is considered.

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

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

$$\sum_{i=1}^{n} R_i(\theta) = \sum_{i=1}^{n} \| Y_i - f(X_i, \theta) \|^2$$

with

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

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 $k - 1$.

Backpropagation algorithm for regression

- Let us introduce the notations

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

- 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

- 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.

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

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 :

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

where B is a subset of $\{1, \ldots, n\}$ with cardinality m.

The stochastic gradient descent algorithm

- In the previous algorithm, we do not compute the gradient for the empirical 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).

To apply the SGD algorithm, we need to compute the gradients $\bigtriangledown_\theta \ell(f(X_i, \theta), Y_i)$. For this, we use the Backpropagation algorithms.

Stochastic Gradient Descent algorithm

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

$$\tilde{\nabla}_\theta = \frac{1}{m} \sum_{i \in B} \nabla_\theta \ell(f(X_i, \theta), Y_i) + \lambda \nabla_\theta \Omega(\theta),$$

    where B is a batch with cardinality m.
  - Update the parameters

$$\theta^{(r+1)} = \theta^{(r)} - \varepsilon_r \tilde{\nabla}_\theta,$$

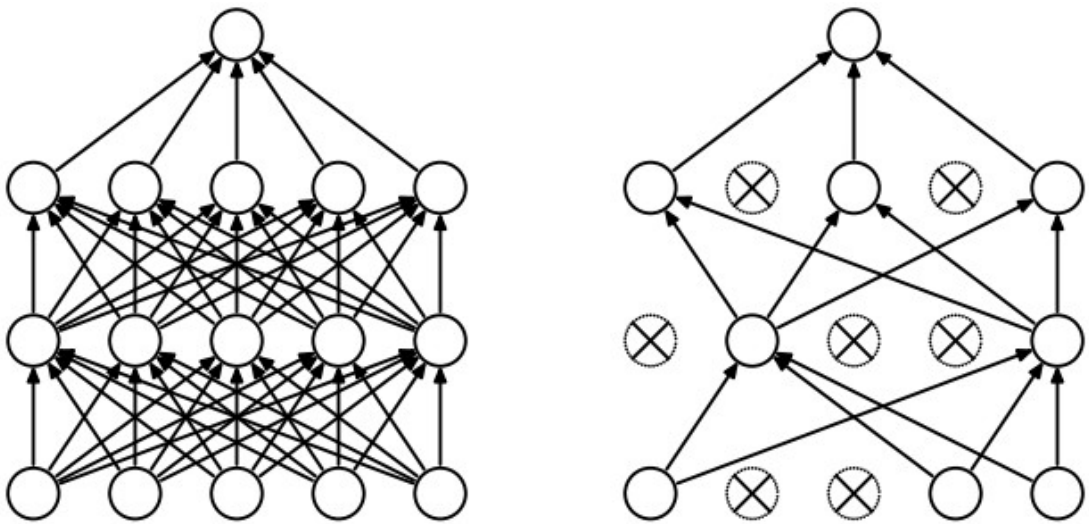
    where $\varepsilon_r > 0$ is the learning rate. that satisfies $\varepsilon_r \to 0$, $\sum_r \varepsilon_r = \infty$, $\sum_r \varepsilon_r^2 < \infty$, for example $\varepsilon_r = 1/r$.

Regularization

- We have already mentioned $\mathbb{L}^2$ or $\mathbb{L}^1$ penalization.
- 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

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

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 ..

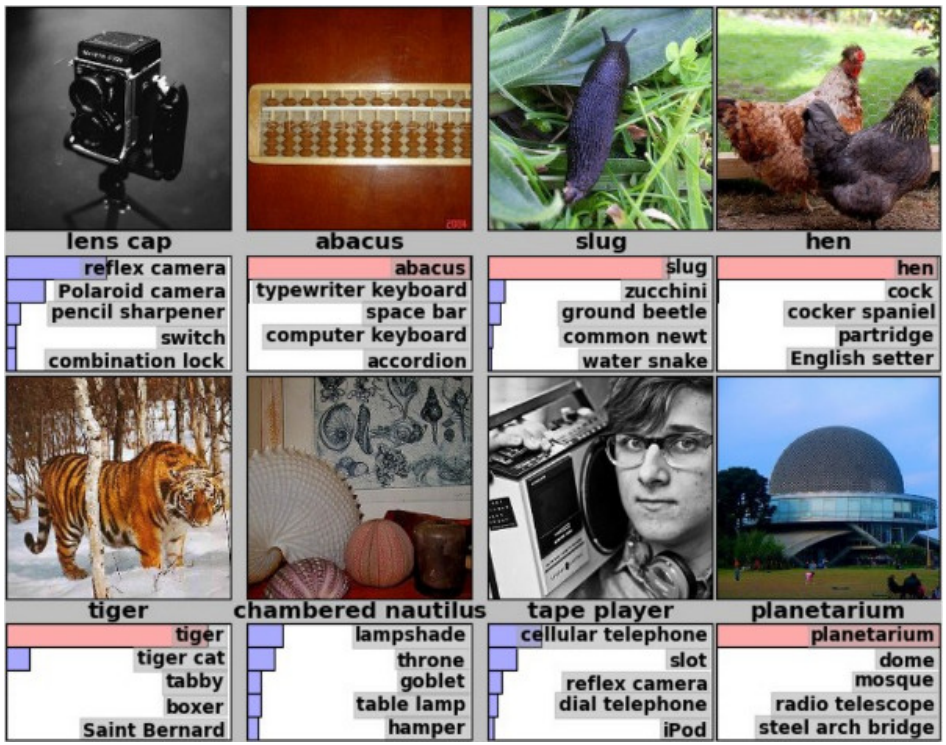


FIGURE – Image annotation

Convolutional neural networks

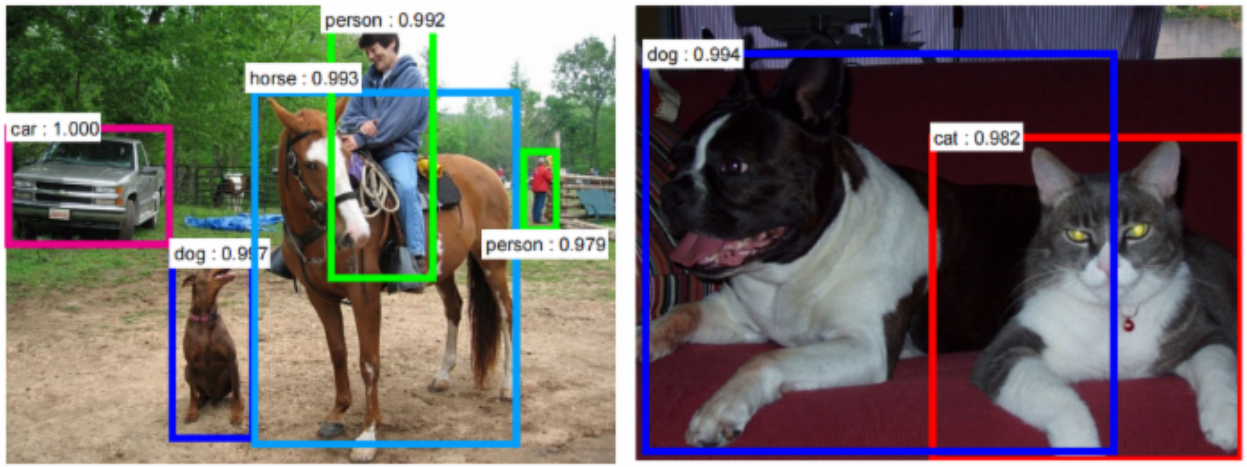
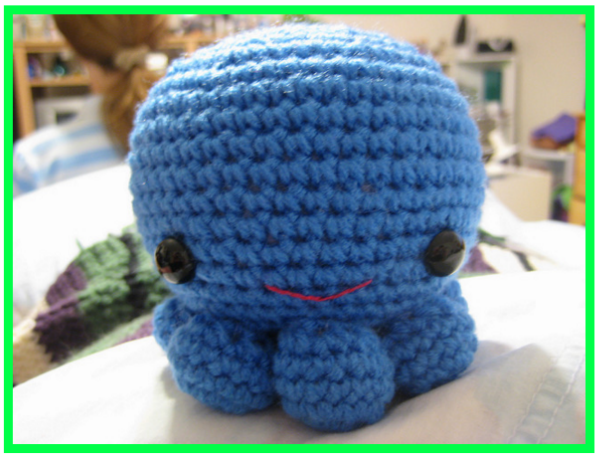


FIGURE – Image Segmentation.

Image Classification

Your specialized huggable recommendation

Go for it! Hug it out. With a score of **0.695779**, we're somewhat sure.



Your specialized huggable recommendation

Don't hug that. Please. With a score of **0.999875**, we're pretty sure.

Your specialized huggable recommendation

Don't hug that. Please. With a score of **0.778686**, we're pretty sure.

Your specialized huggable recommendation

Go for it! Hug it out. With a score of **0.958104**, we're pretty sure.

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.

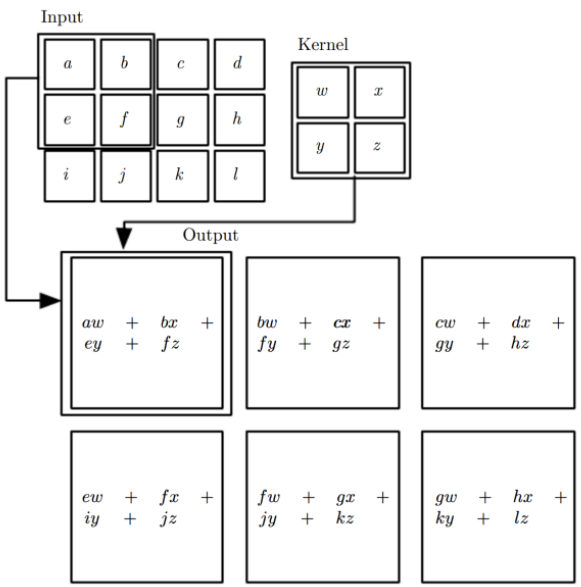


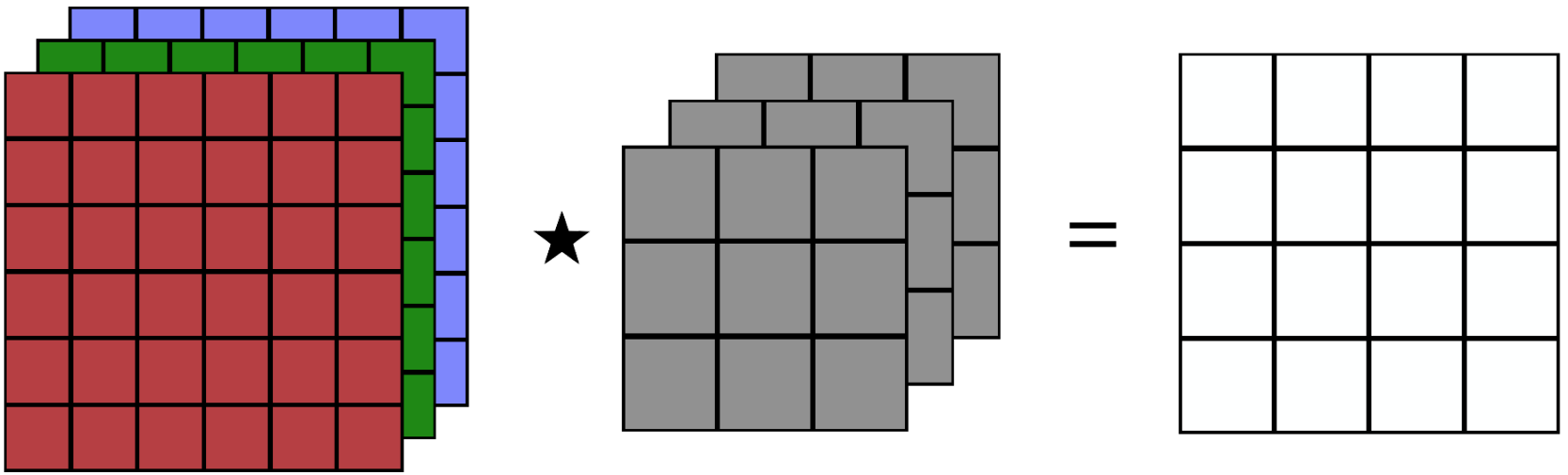
FIGURE – 2D convolution

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.

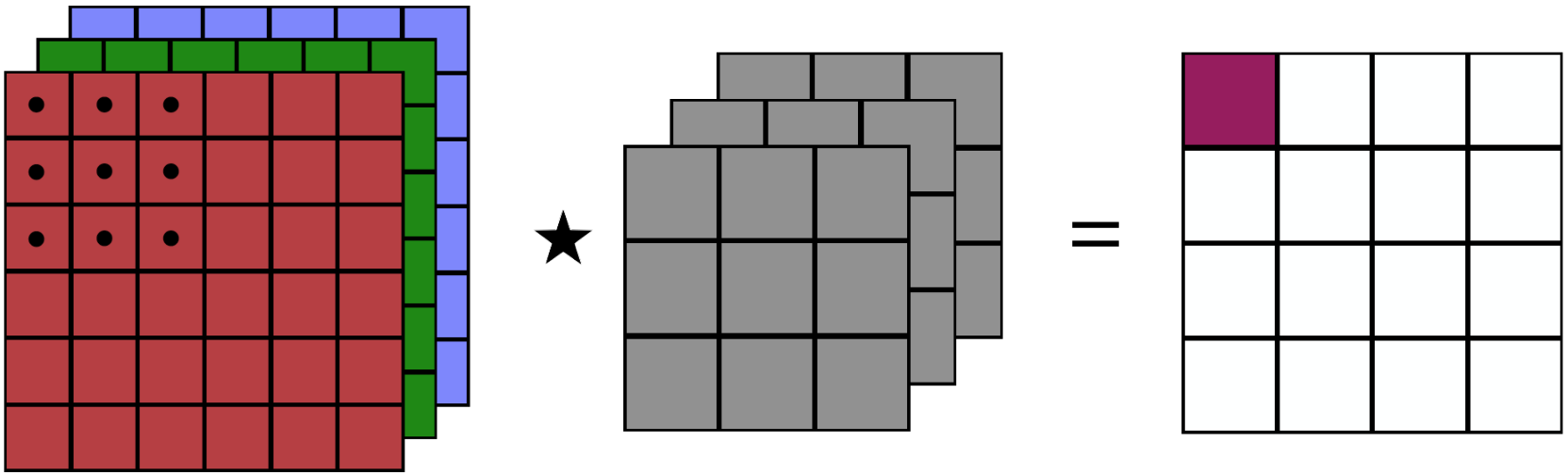
Convolution - channels

Colored image in 3 dimensions : (height, width, channels (RGB))



Convolution - channels

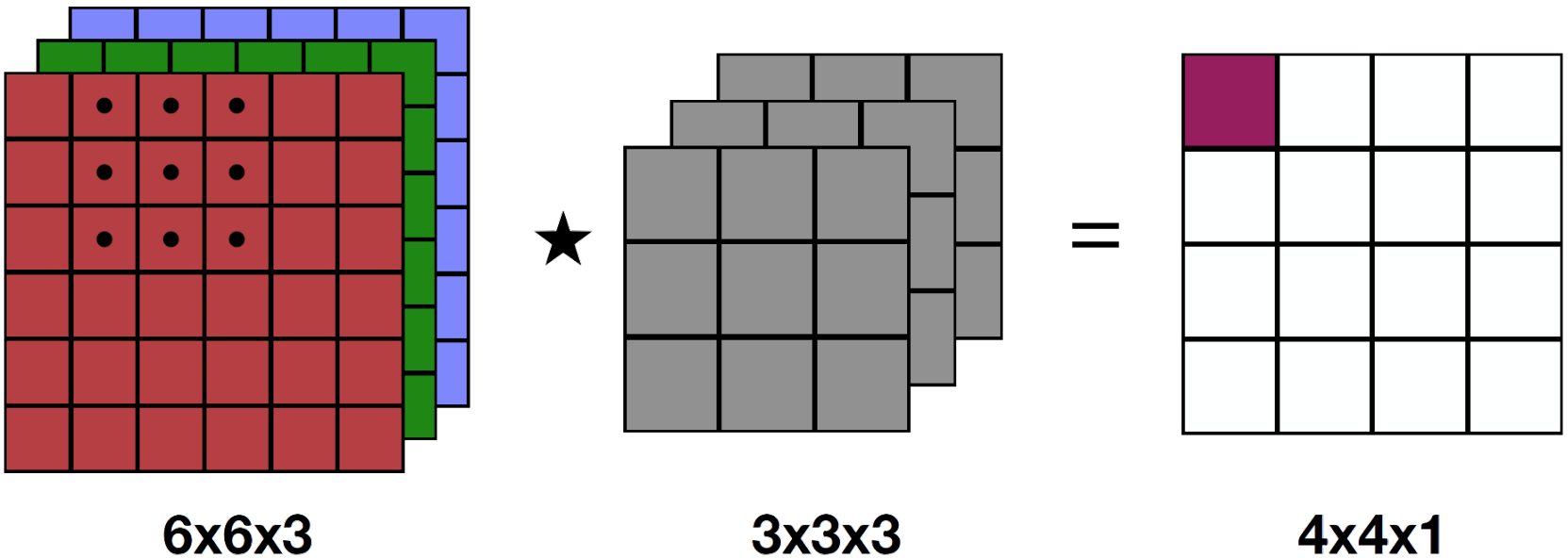
Colored image in 3 dimensions : (height, width, channels (RGB))



$$(F \star img)(x, y) = \sum_c \sum_m \sum_n F^c(n, m) \cdot Img^c(x + m, y + n)$$

Convolution - channels

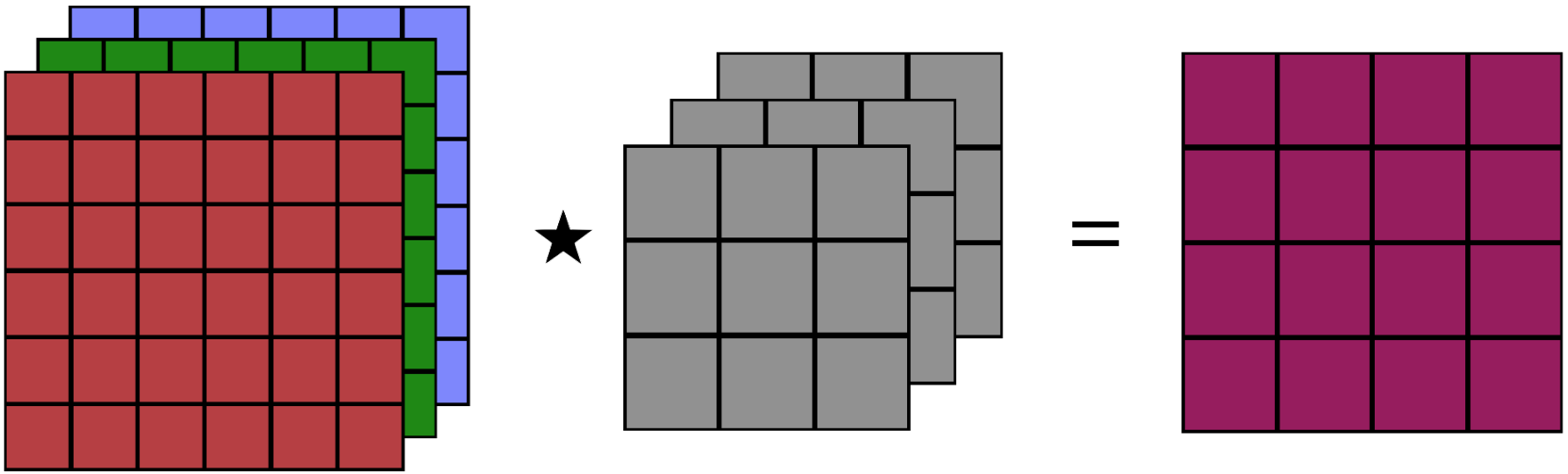
Colored image in 3 dimensions : (height, width, channels (RGB))



$$(F \star img)(x, y) = \sum_c \sum_m \sum_n F^c(n, m) \cdot Img^c(x + m, y + n)$$

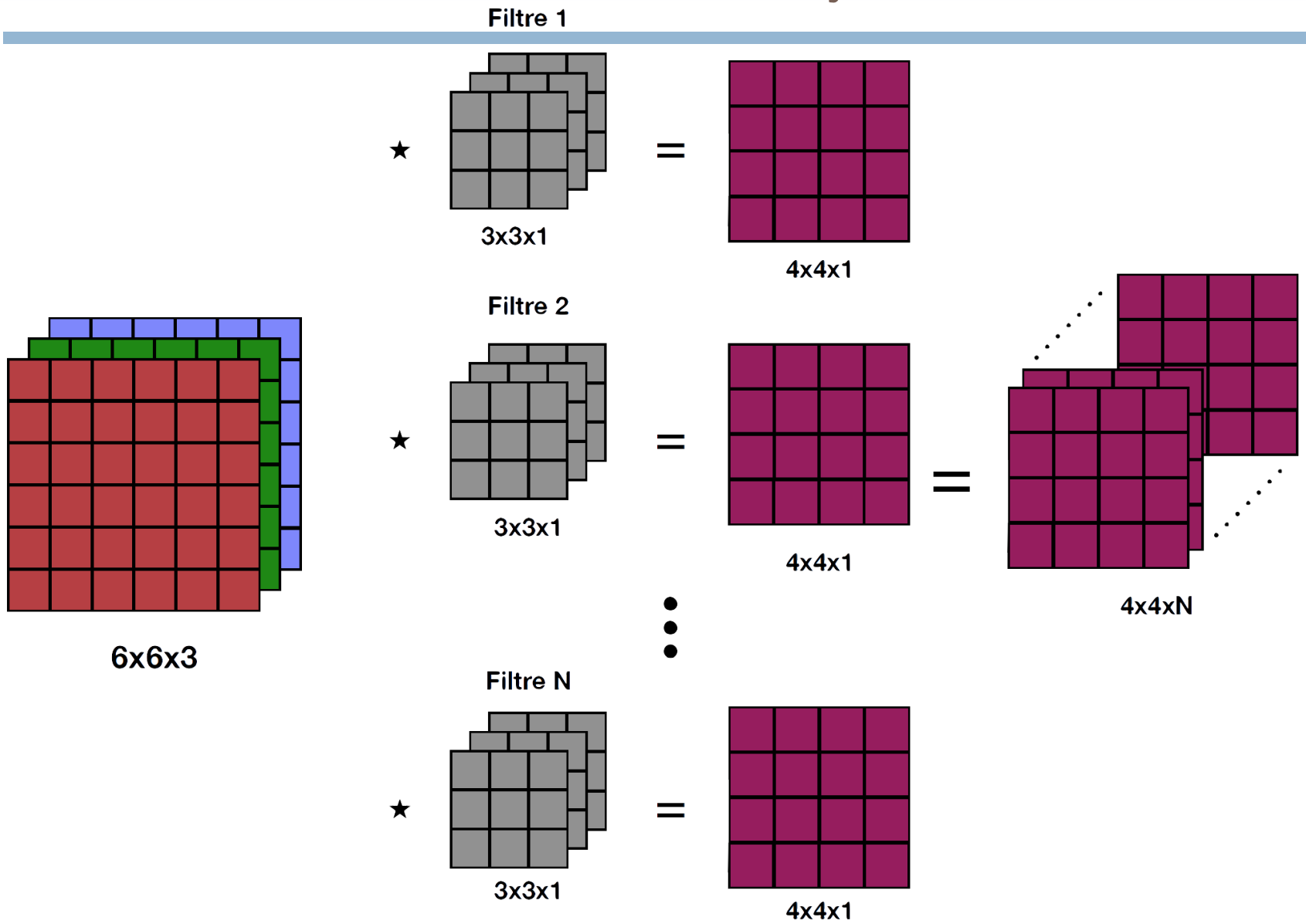
Convolution - channels

Colored image in 3 dimensions : (height, width, channels (RGB))

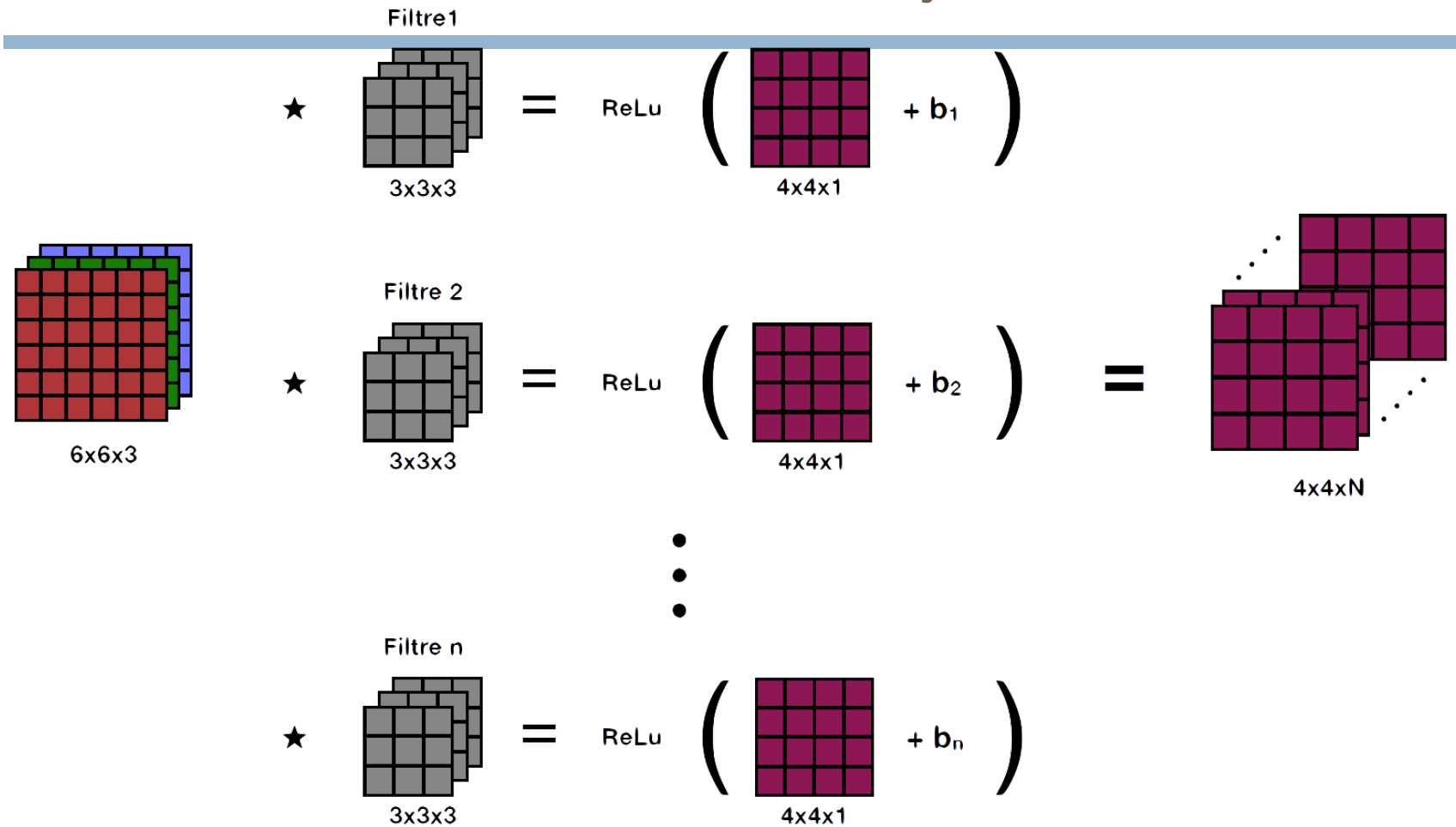


$$(F \star img)(x, y) = \sum_{c} \sum_{m} \sum_{n} F^{c}(n, m) \cdot Img^{c}(x + m, y + n)$$

Convolution - Layer



Convolution - Layer



- Input Dimension : (h, w, c_i)
- Filter Dimension : (f, f), Nb filter : N
- Output Dimension : $(h - f + 1, w - f + 1, N)$

Dimensions formula - Convolutional Layer

Input Dimensions : $(h \times w \times c)$

- h : height of the image,
- w : width of the image,
- c : channel of the image.

Convolution parameter : (f, p, s)

- f : filter size
- p : padding size
- s : stride size
- n : number of output channels

Output Dimensions :

$$\left(\left[\frac{h + 2p - f}{s} + 1 \right] \quad \times \quad \left[\frac{w + 2p - f}{s} + 1 \right] \quad \times \quad n \right)$$

Number of parameters : $N_p = \big((h \cdot w \cdot c) + 1\big) \cdot n$

Convolution layer

- 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 refered 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.

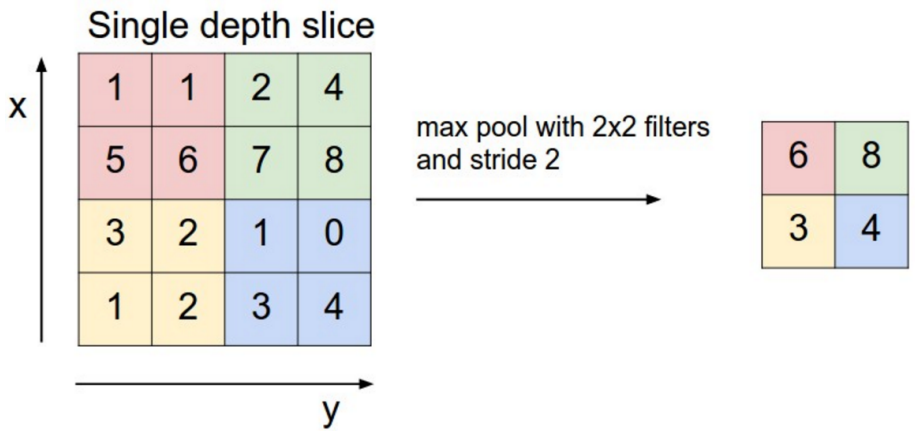


FIGURE – Maxpooling and effect on the dimension

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 experimental than 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.

LeNet-5 and Mnist Classification

Objectif :

- Manuscrit number from 0 to 9 hand-written
- Dimension (32,32,1)
- Image for training : $N_{train} = 60.000$
- Image for testing : $N_{test} = 10.000$

LeNet-5 and Mnist Classification

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

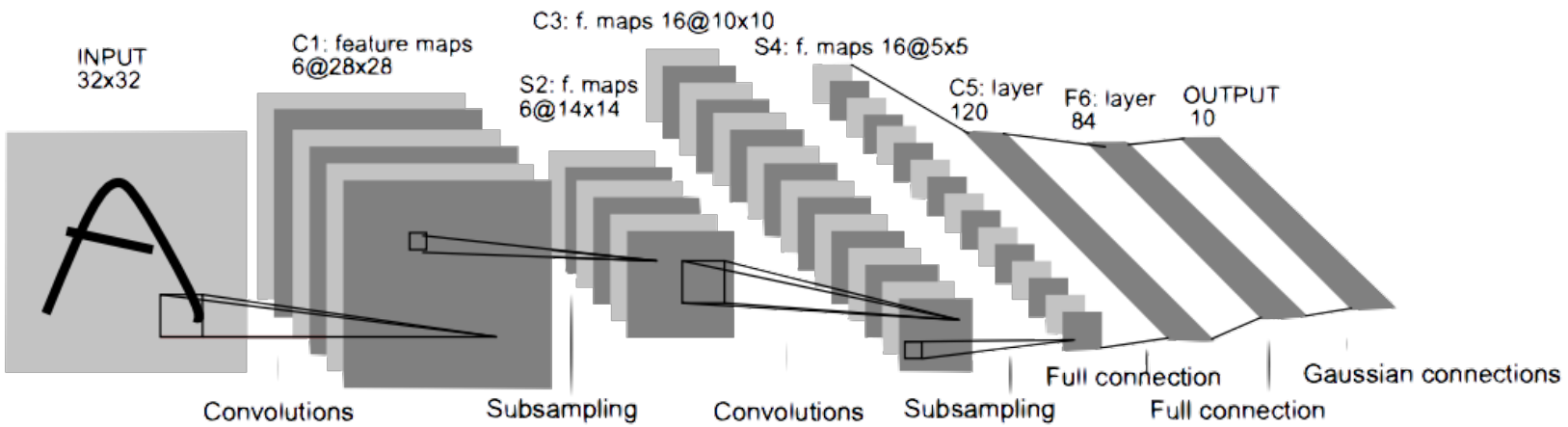


FIGURE – Architecture of the network Le Net. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998)

Size and height decrease while channel increase.

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

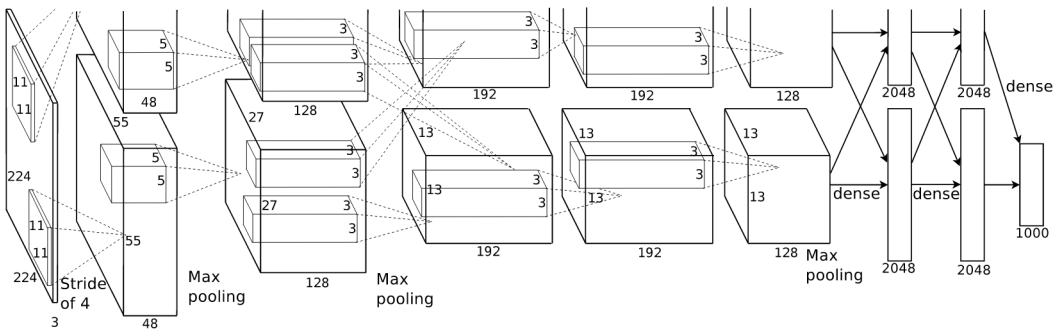


FIGURE – Architecture of the network AlexNet. Krizhevsky, A. et al (2012)

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)

Alex Net (Krizhevsky, A. et al (2012))

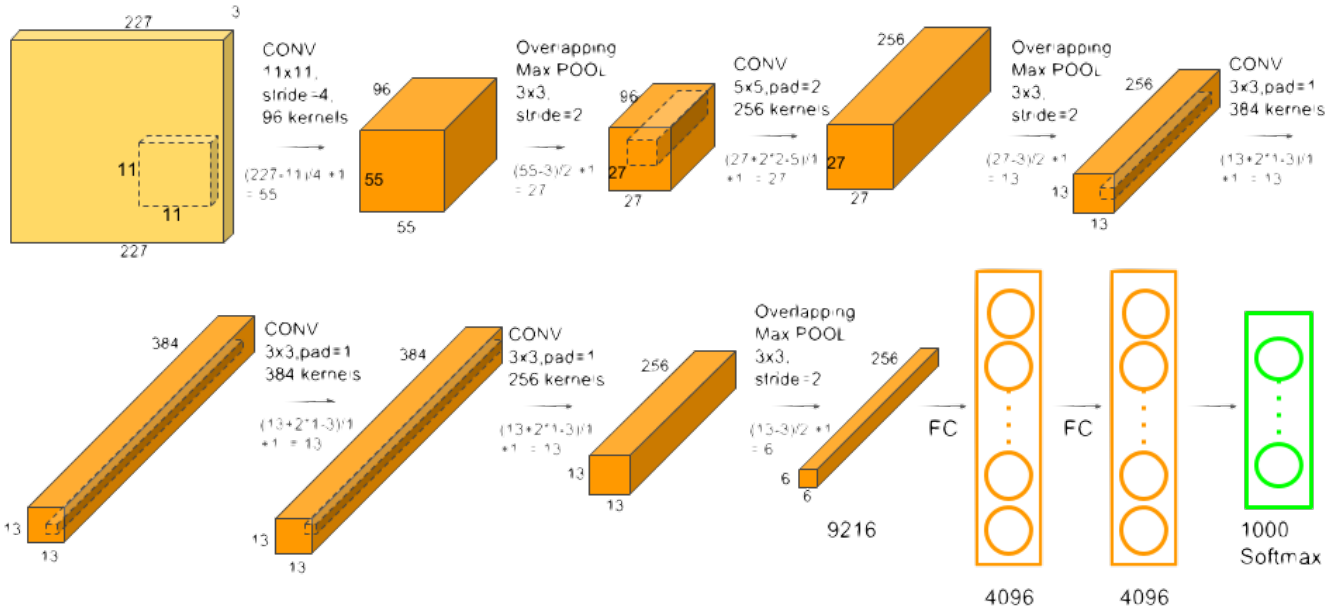
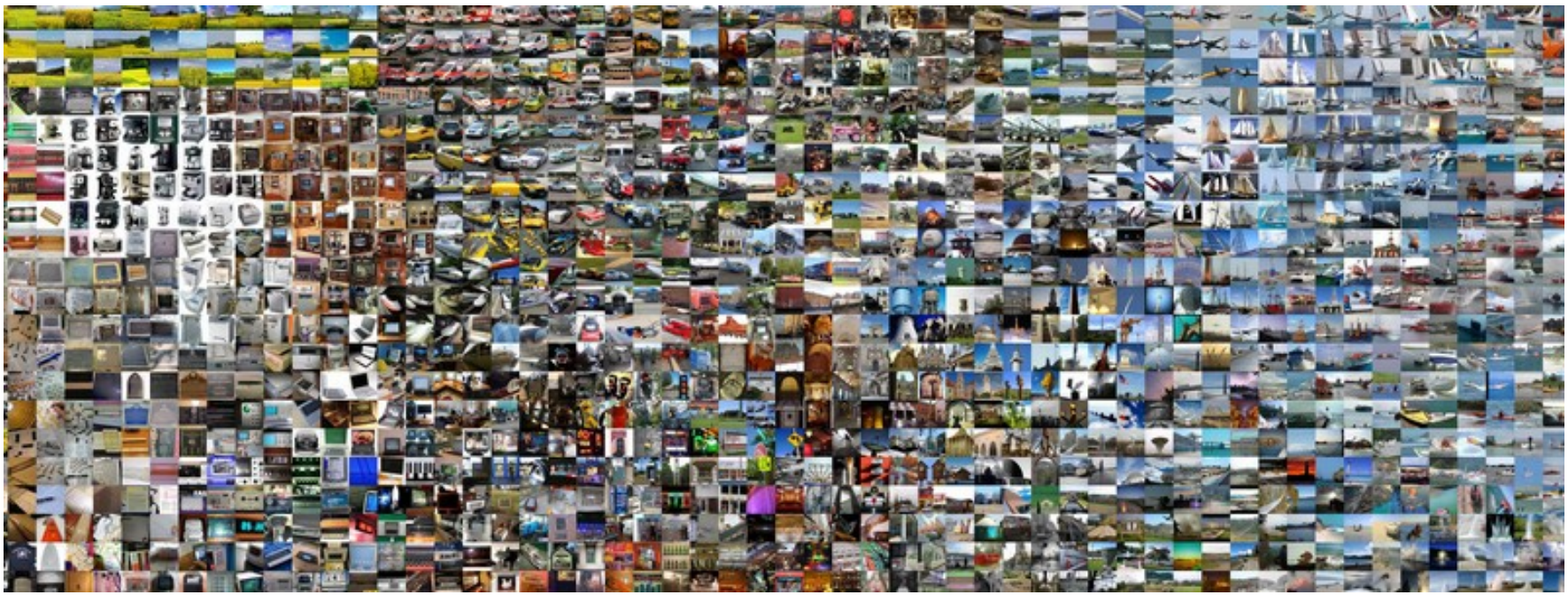


Image source : https://www.learnopencv.com/understanding-alexnet/

- $\simeq 60M$ parameters
- Similar to LeNet-5 but Much bigger
- Only ReLu is different.
- Multiple GPUs
- Learned on huge amount of data : **ImageNet** .

Image net

http://image-net.org/



- 1000 classes
  - 1,2 M training images,
  - 100k test images.

VGG16

Use always same layer :

- Convolution : 3×3 filter, stride $= 1$, padding$=$ SAME
- Max Pooling : 2×2 filter, stride $= 2$

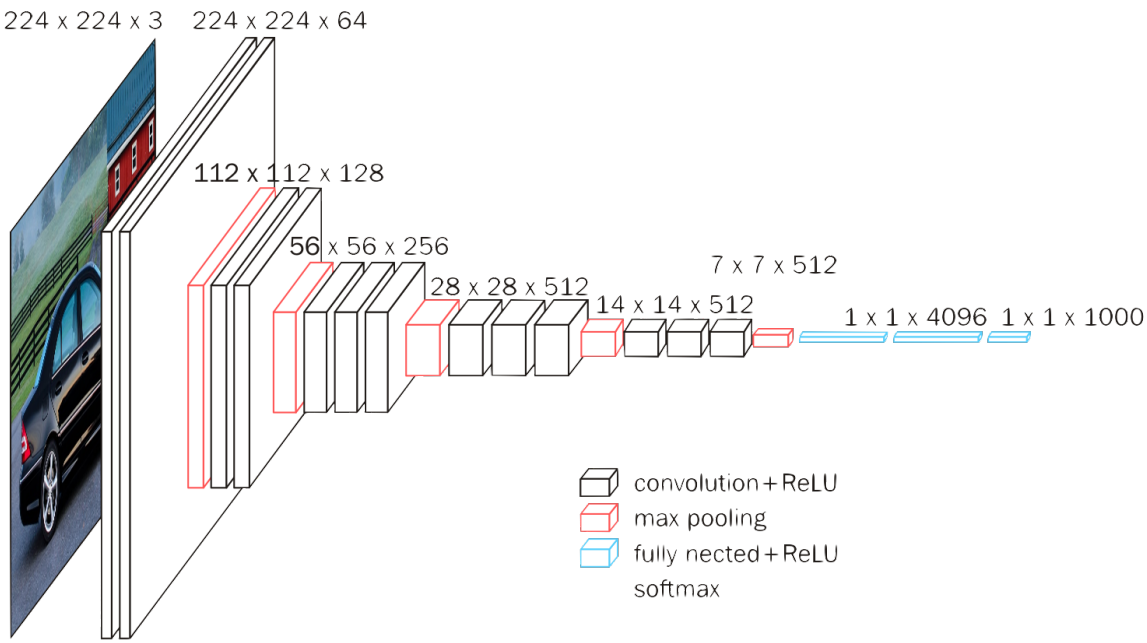


FIGURE – Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition (2014). 138M parameters.

Image source : https://blog.datawow.io/cnn-models-ef356bc11032, realized with Tensorflow.

Architectures

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

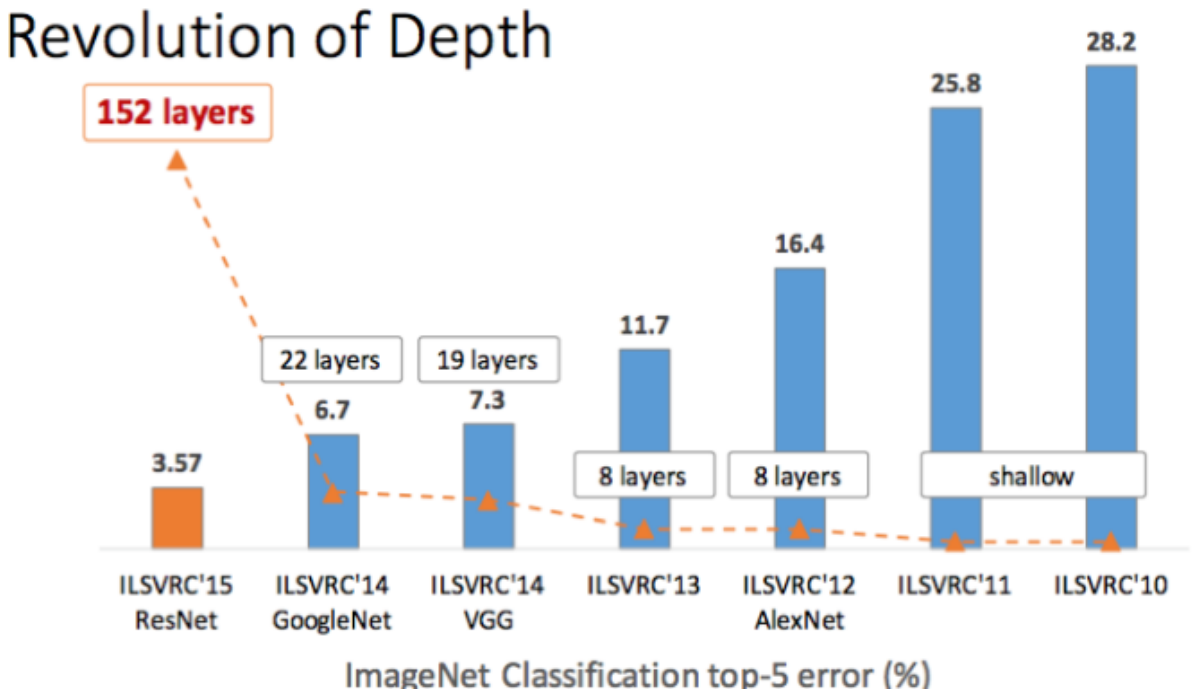


FIGURE – Evolution of the deepth of the CNN and the test error

In practice

Pre-trained model

- Some models using previous architectures have already be trained on massive amount of data (ImageNet).
- Theses models are available on easily usable on deep learning Framework such as Keras (Python library).
- Two ways to used them :
  - **Transfer Learning**
  - **Fine Tuning**

Data Augmentation

Outline

- Neural Networks
- Multilayer perceptrons
- Estimation of the parameters
- Backpropagation algorithms
- Optimization algorithms
- Convolutional Neural Networks
- Conclusion

Conclusion

- We have presented in this course the feedforward neural networks and explained how the parameters of these models can be estimated.
- The choice of the architecture of the network is also a crucial point. Several models can be compared by using a validation methods on a test sample to select the "best" model.
- The perceptrons are defined for vectors. They are not well adapted for some types of data such as images. By transforming an image into a vector, we loose spatial information, such as forms.

Conclusion

- 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 channels.
- They involve complex architectures, which are in constant evolution.
- Various libraries and frameworks have been developed : we will use Keras (Tensorflow) for simplicity reasons.
- GPU is required for training the deep networks.
- Few theoretical foundations available.

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