Parcours IA - 2019/2020

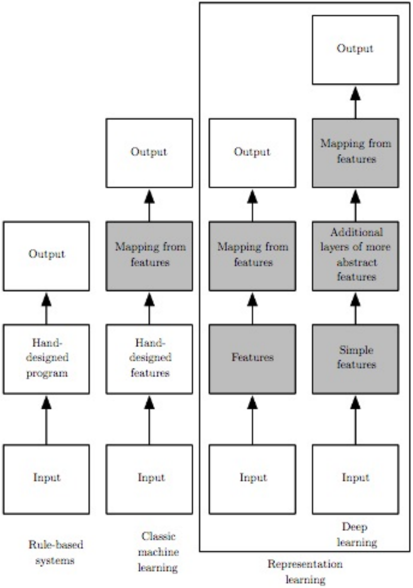
Fondements de l'Intelligence Artificielle et du *Machine Learning*

Nicolas Vayatis

Novembre 2019

Session 2 - Principaux algorithmes d'apprentissage

"Big picture" of Learning



Supervised machine learning
Setup

- Data : $(X_1, Y_1), \ldots, (X_1, Y_1)$ with X_i being a vector of variables (factors) for observation i, and Y_i being the label of X_i
- Hypothesis class : set of functions $h \in \mathcal{H}$
- Loss of a function h at a data point (X, Y) :

$$\ell(h(X), Y) \geq 0$$

- *Empirical risk* of a function h over the data :

$$\widehat{L}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(X_i), Y_i)$$

Overview of supervised learning methods

A. Shallow Learning
   - Local methods : k-NN, decision trees, local averaging
   - (Support Vector Machines)

B. Ensemble Learning
   - Bagging, Random Forests, Boosting

C. Deep Learning

Principle of shallow machine learning
Regularized optimization

- Objective : to find a function that fits the data and displays predictive power

- Until now : Learning amounts to the minimization of training error for some loss function over the hypothesis class of functions $h \in \mathcal{H}$ plus some penalty

$$C_n(h) = \underbrace{\hat{L}_n(h)}_{\text{Training error}} + \lambda \underbrace{\mathrm{pen}(h, n)}_{\text{Regularization}}$$

- Example : ridge regression where $h(x) = \theta^T x$:
  $\hat{L}_n(h) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \theta^T X_i)^2$ and $\mathrm{pen}(h, n) = \frac{1}{n}\|\theta\|_2^2$

- The penalty grows with the complexity of h (or the size of $\mathcal{H}$) and vanishes when $n \to \infty$

Machine Learning Methods
Optimization is central

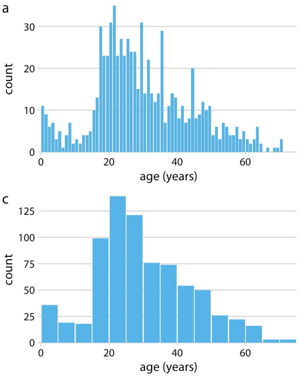
Some examples :

- (Sparse) Linear models $\longrightarrow$ gradient method (and extensions)
- Kernel ridge regression $\longrightarrow$ quadratic optimization (with KKT conditions)
- Deep learning $\longrightarrow$ nonconvex optimization (stochastic gradient descent) + implicit regularization (tricks)

Other forms of regularization

- General idea : Regularized function estimation without global optimization
- Two directions :
  - Local methods : nearest-neighbors and decision trees
  - Ensemble methods : bagging, boosting, random forests

Regularization without optimization
The case of histograms



Distribution of the age of the passengers of the Titanic with bins varying from 1 year to 15 years

Ingredients for that type of regularization

- Histograms use two general ideas of locality (bins) and averaging (piecewise constant function)
  - define local : which training data can be considered to be close to the point where a prediction has to be made ?
  - averaging (or voting if discrete outcome) : take the average of the values over each bin
- Regularization through hyperparameter selection : find the optimal bin size amounts to finding the right hypothesis class

From histograms to Machine Learning

- In the previous example, the objective was to estimate a density function from a sample drawn from this distribution (problem known in the literature as *nonparametric density estimation* or *kernel density estimation*)
- Density estimation can be seen as an *unsupervised learning problem*
- In the supervised setting, we establish the values of the function on every bin either by averaging (regression setup) or by voting (classification setup). The general terminology for averaging/voting is aggregating/combining.

A. Older Machine Learning approaches : Local methods

1. Nearest neighbors
2. Decision trees

Two popular types of local methods

- Nearest neighbors : local are the closest points
- Partition-based rules (also called *decision trees*) : local are the points within a cell from a partition of the input space only

Works for classification, regression and other problems... but here we will focus on classification

Problem considered
(Multiclass) Classification

- Given :
  - Consider a sample of classification data

$$(X_1, Y_1)...(X_n, Y_n)$$

  where $X_i \in \mathbb{R}^d$ vector of independent variables,
  $Y_i \in \{1, \dots, C\}$ the label
- Want :
  - to predict the label y at any position x

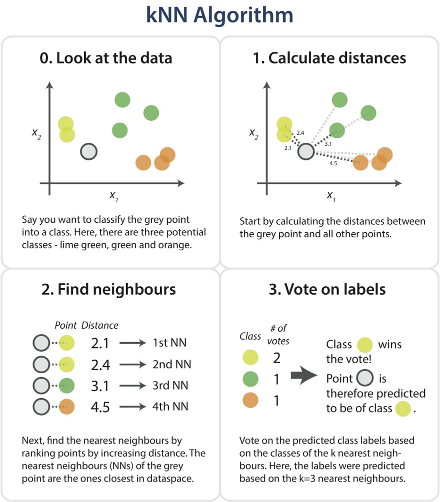
A. Local methods

1. k-Nearest neighbors (k-NN)

k-Nearest Neighbor (1/4)
Principle of the k-NN algorithm

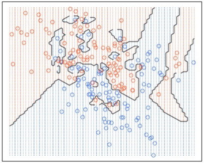
1. Compute distances
   - Compute pairwise distances $d(x, X_i)$ for all $i = 1, \ldots, n$
2. Sort training data
   - Sort the data points from the closest $X_{(1)}$ to the farthest $X_{(n)}$ (i.e. $d(x, X_{(1)}) \leq \ldots \leq d(x, X_{(n)})$)
3. Prediction $\hat{h}(x, k)$ = Majority vote of the k-NN
   - Consider the labels $Y_{(1)}, \ldots, Y_{(k)}$ of the k closest points to x and take the majority vote $\hat{h}(x, k) = \arg\max_c \{\sum_{l=1}^{k} \mathbb{I}\{Y_{(l)} = c\}\}$

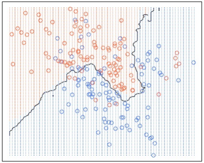
k-Nearest Neighbor (2/4)
Principle of the k-NN algorithm

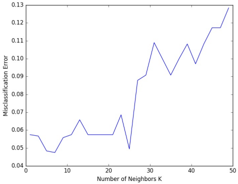


Nearest Neighbors (3/4)
Hyperparameters

- Choice of a distance d between points of $\mathbb{R}^d$
- Number k of Nearest Neighbors, estimated by cross-validation :







k-Nearest Neighbor (4/4)
Theory

- Recall : classification error $L(h) = \mathbb{P}(Y \neq h(X))$ and $L^* = \inf L$
- Consistency result :

$$\mathbb{E}L(\hat{h}(\cdot, k_n)) \to L^*$$

  under the condition : $k_n \to \infty$ and $k_n/n \to 0$ when $n \to \infty$
- No closed-form solution for optimal k_n (in practice, we use cross-validation)
- No theoretical clue on the choice of the distance (related to data representation and the physics of the problem)

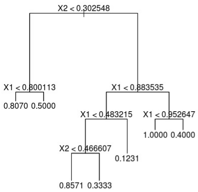
A. Local methods

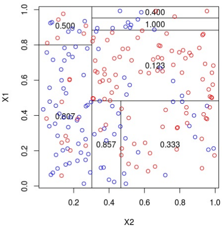
2. Partition-based (decision trees)

Partition-based classifier (1/4)
Computing the prediction for fixed partition

Denote the partition by $c = \bigcup_j \gamma_j$ with cells γ_j

1. Find the cell $\gamma(x)$ where x falls
2. Consider the training data in the cell $\gamma(x)$
3. Prediction $\hat{h}(x, c)$ = Majority vote over the training data in cell $\gamma(x)$

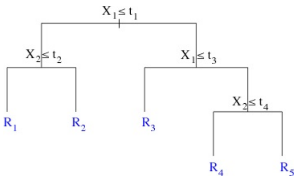




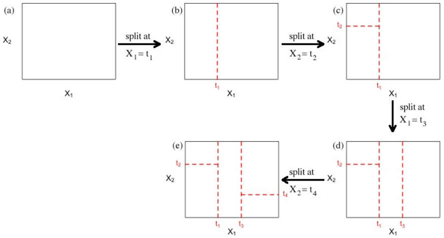
Partition-based classifier (2/4)
Building data-driven partitions

- Start with all the training data and find a (simple) classifier which minimizes some cost function

- Repeat the process with the subset of training data on each side of the frontier of the classifier $\longrightarrow$ this is called *recursive partitioning*



tree representation

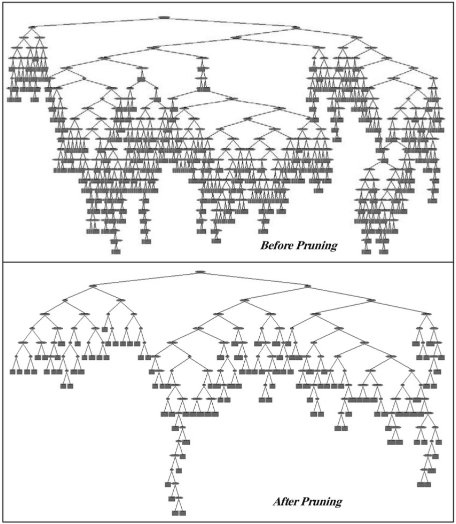


recursive partitioning of the X-domain

Partition-based classifier (3/4)
Hyperparameters

- Cost function optimized locally (at the cell level for the data within the cell)
- Number of minimal points in a cell
- Maximal depth of the tree or total number of cells estimated by pruning the tree - pruning amounts to explore the class of all subpartitions (subtrees) and optimize a penalized criterion of the form
$$\arg\min_{c} \hat{L}_n(h_c) + \lambda |c|$$
where $c \subset \hat{c}$ is the collection of subpartitions obtained from the learned partition by pruning from bottom to top

Pruning example



Partition-based classifier (4/4)
Theory

- Case of regular partitions with cells which are hypercubes of $\mathbb{R}^d$ with edges of length δ_n :

  $$\mathbb{E}L(\hat{h}(\cdot, \delta_n)) \to L^*$$

  under the condition : $n\delta_n^d \to \infty$ and $\delta_n \to 0$ when $n \to \infty$ (need enough data points in every cell and cell diameter go to zero as sample size grows)

- Case of data-driven partitions : VC and Rademacher theory applies

Take-home message on local methods

Major limitations :

- The k-Nearest Neighbor method requires to store all the training data in order to predict the label of new entries.
- Decision trees are extremely unstable.
- Both display prediction performance below state-of-the-art methods

Virtue of decision trees :

- Can handle missing/categorical data, scale change
- Can be expressed in terms of logical rule $\longrightarrow$ explainable machine learning

What can be saved from decision trees ?

B. Shallow and efficient Machine Learning algorithms : Ensemble methods

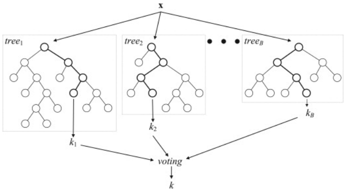
1. Bagging and Random Forests
2. Boosting

- Technology : the champions in data science competitions combine several methods to boost performance (e.g. BelKor team, winner of the Netflix challenge)
- Decision theory : Social choice theory
- Probability : Ergodic theorem
- Nonparametric statistics : aggregation estimators

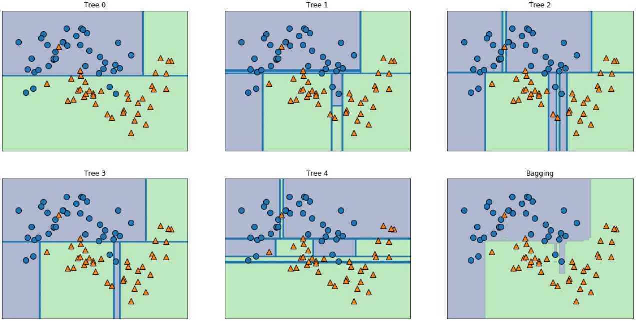
- Consider we already have a machine learning algorithm with reasonable performance that we want to improve, e.g. decision tree, k-NN, SVM, ...
- The idea of the ensemble is to generate different functions from the same training data and the same hypothesis space
- In the illustration coming next and most of the discussion, the basic hypothesis space is the one with decision trees obtained with orthogonal splits (such splits are called decision stumps).

Ensembles of decision trees
General principle

- Generate a collection of *weak* predictors (ensemble) obtained with a basic Machine Learning algorithm (e.g. decision tree)
- For every point x, compute their individual predictions
- Take an average or a majority vote of the individual predictions to determine the prediction of the ensemble



Ensembles of decision trees
Resulting classifier



Ensembles of decision trees
Three popular methods

- Bagging (Breiman, 1996)
- Random forests (Amit-Geman, 1997 ; Breiman, 2000)
- Boosting (Freund-Schapire, 1996)

B. Ensemble methods

1. Bagging and Random Forests

Bagging and Random Forests
What is their hypothesis space ?

- Denote by $\mathcal{H}$ the base hypothesis space (for the not so brilliant algorithm we already have, e.g. decision trees)
- Denote by D_n the training data and assume that we can sample functions $\hat{h}_1, \ldots, \hat{h}_t$ (the ensemble) from $\mathcal{H}$ conditionally to D_n
- With an ensemble of T functions, the output of bagging/random forests is the average of those "random" (generated based on the data) functions :

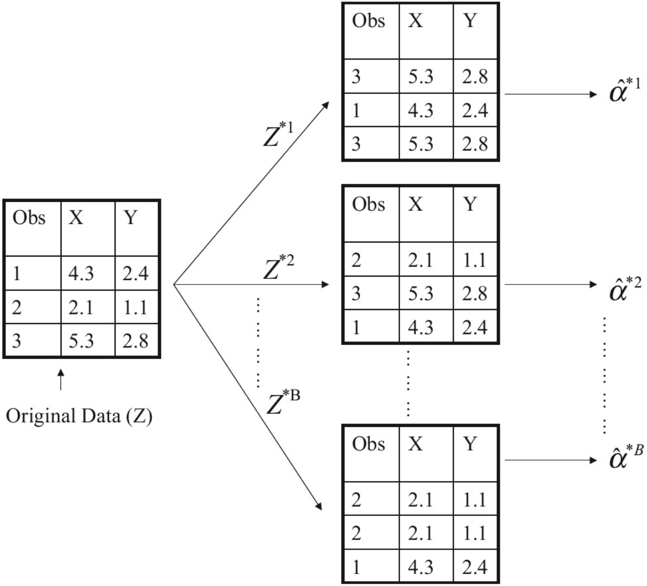
$$\hat{f}_T = \frac{1}{T} \sum_{t=1}^{T} \hat{h}_t$$

- The hypothesis space for those methods is the linear span of the base hypothesis space $\mathcal{H}$. This can be a huge space !

How to generate the ensemble ?
Bootstrap and aggregation

- Bagging and random forests rely on bootstrap samples of the training data
- They differ by some different specifications of the recursive partitioning procedure to build each tree (no pruning involved)

What is bootstrap in general ?



Bagging Theory

- Consistency result for some idealized version of bagging
- Most important ! Bagging can render inconsistent rules consistent !
  - Biau, Devroye and Lugosi (2008) have considered bagging applied to 1-NN, given that 1-NN is inconsistent in general classification scenarios (except zero-noise or pure random labels)
  - Bagging applied to 1-NN classifier is consistent under some reasonable conditions on the sampling process

B. Ensemble methods

2. Boosting

Historical perspective on Boosting

- Original paper : Freund, Y. and Schapire, R. E. (ICML, 1996).
- Interpretation of the optimization problem solved as stochastic gradient descent : Friedman, J. H. (CSDA, 2002).
- Wald Memorial lecture (IMS, 2000) : Leo Breiman declares that *"understanding Boosting is the most important problem in Machine Learning"*
- Proof of boosting consistency : Lugosi, G. and Vayatis, N. (Special issue with discussion of the Annals of Statistics, 2004).
- Xgboost, a scalable implementation : Chen, T. and Guestrin, C. (ACM SIGKDD, 2016).

Boosting (1/7)
Principle

- **Input**
  - Data sample $D_n = \{(X_i, Y_i) \ : \ i = 1, \ldots, n\}$ with classification data $\{-1, +1\}$
  - Base hypothesis class $\mathcal{H}$ of *weak* classifiers such as decision trees (assumed to be symmetric, i.e. $h \in \mathcal{H}$ iff $-h \in \mathcal{H}$)
- **Iterations** $t = 1, \ldots, T$**.**
  - Compute weights $w_t > 0$ and weak classifiers $\widehat{h}_t \in \mathcal{H}$
- **Output.**
  - The Boosting classifier takes the sign of the following linear combination of weak classifiers : $\widehat{f}_n(x) = \sum_{t=1}^{T} w_t \widehat{h}_t(x)$

Boosting (2/7)
Notations

- Boosting distributions on the data : sequence of discrete probability distributions over $\{1, \dots, n\}$ denoted by Π_t, $t \geq 1$
- Weighted training error : for any weak classifier $h \in \mathcal{H}$ and for $t \geq 1$

$$\widehat{\varepsilon}_t(h) = \sum_{i=1}^{n} \Pi_t(i)\mathbb{I}\{h(X_i) \neq Y_i\}$$

1. **Initialization.** Π_1 is the uniform distribution on $\{1, \ldots, n\}$
2. **Boosting iterations.** For $t = 1, \ldots, T$, find the weak classifier such that :

$$\widehat{h}_t = \arg\min_{h \in \mathcal{H}} \widehat{\varepsilon}_t(h)$$

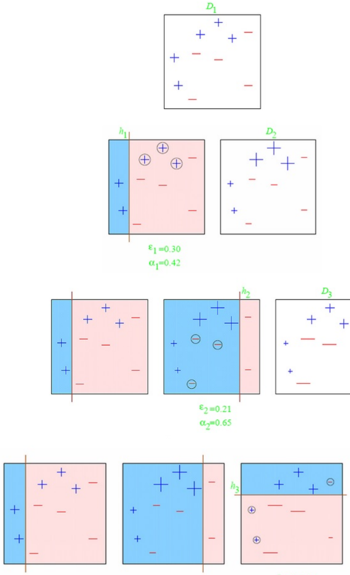
then set $e_t = \widehat{\varepsilon}_t(\widehat{h}_t)$ and take the weight to be

$$w_t = \frac{1}{2} \log\left(\frac{1 - e_t}{e_t}\right)$$

3. **Boosting distribution update.** For any $i = 1, \ldots, n$,

$$\Pi_{t+1}(i) \propto \Pi_t(i) \exp\left(-w_t Y_i \cdot \widehat{h}_t(X_i)\right)$$

Boosting (4/7)
Example



Boosting (5/7)

- Boosting can be interpreted as a functional gradient descent on the following functional :

$$\hat{A}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \exp\left(-Y_i f(X_i)\right)$$

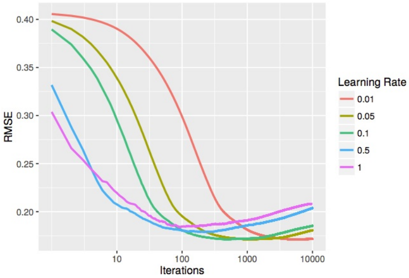
  where f is taken in a hypothesis space which is the linear span of 'simple' set $\mathcal{H}$ of classifiers.

- Exercise : why ?

  Refer to : J. Friedman, "Greedy Function Approximation : A Gradient Boosting Machine", The Annals of Statistics, Vol. 29, No. 5, 2001.

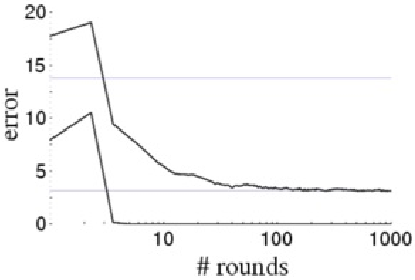
Boosting (6/7)
Hyperparameters for Gradient Boosting

- The number T of iterations : the bigger, the higher the chance of overfitting.
- The stepsize η is fixed : decreasing learning rate tends to improve generalization performance.



Boosting (7/7)
A mystery not fully explained yet...

The test error continues to drop along the iterations even though the training error is zero $\longrightarrow$ Regularization effect thanks to averaging ? ?



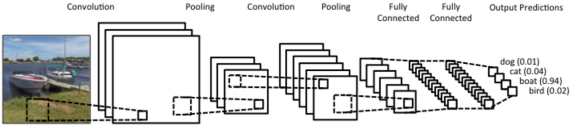
Packages

- Python : scikit-learn
- R :
  - `rpart` : recursive partitioning
  - `caret` : classification and regression training (SVM, random forest...)
  - `xgboost` : extreme gradient boosting

C. Deep Learning :

Introducing the main concepts

Deep Feedforward Network



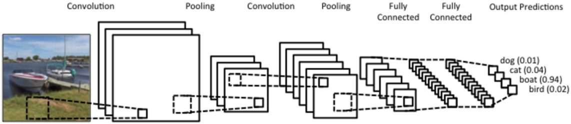
- Hypothesis space : functions of the form

$$h(x, \theta) = \sigma_m \circ A_m \circ \sigma_{m-1} \circ \ldots \circ A_2 \circ \sigma_1 \circ A_1 x$$

where $\theta = (A_1, \ldots, A_m)$ sequence of parameters to be estimated through learning

- We denote by $\sigma = (\sigma_1, \ldots, \sigma_m)$ the so-called activation functions which are hyperparameters related to the choice of a network architecture (which includes the number and size of the layers - see below).

Deep Feedforward Network



- Optimization objective (far from convex ! where is the regularizer ?) :

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(h(X_i, \theta), Y_i)$$

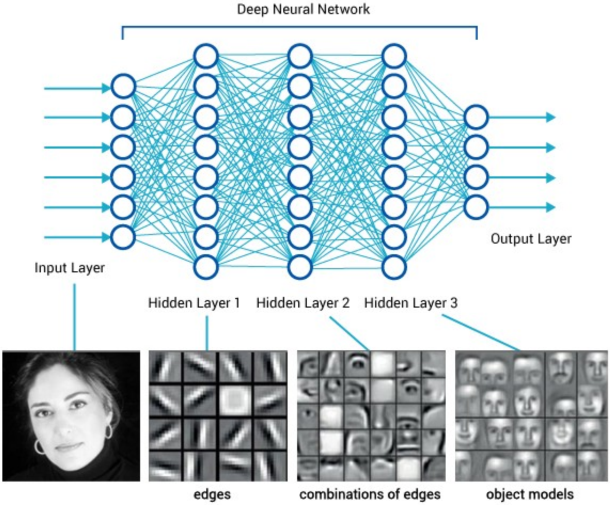
- Optimization method based on stochastic gradient descent (iterates over data points)

$$\theta_{i+1} = \theta_i - \eta \frac{\partial \ell(h(X_i, \theta), Y_i)}{\partial \theta}(\theta_i)$$

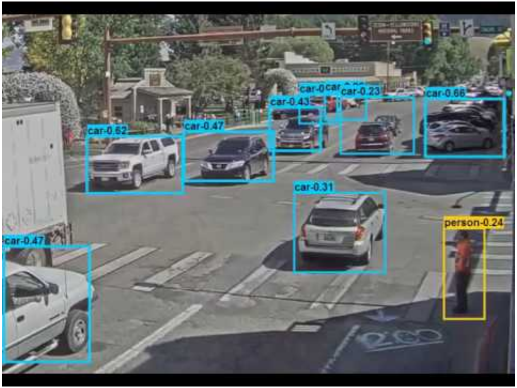
Deep Learning :

Why is it popular ?

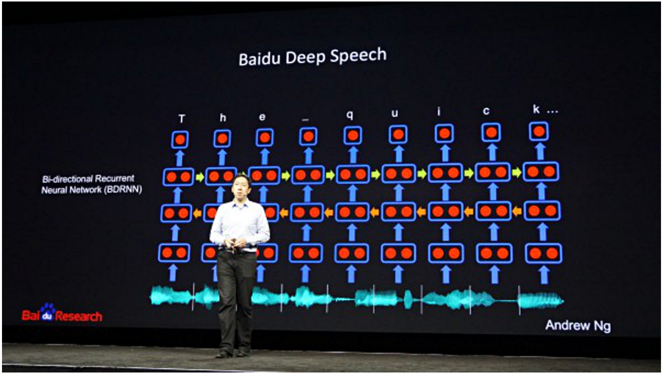
Deep means 'many layers' (compositions) between input and output...



Success of deep learning (1/3)
Computer Vision

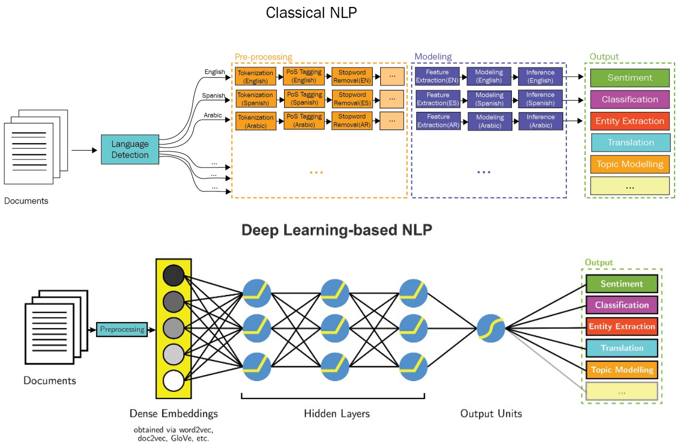


Success of deep learning (2/3)
Speech recognition



System	Clean (94)	Noisy (82)	Combined (176)
Apple Dictation	14.24	43.76	26.73
Bing Speech	11.73	36.12	22.05
Google API	6.64	30.47	16.72
wit.ai	7.94	35.06	19.41
Deep Speech	**6.56**	**19.06**	**11.85**

Success of deep learning (3/3) Natural Language processing



Shallow vs. Deep Learning
Vapnik vs. LeCun

The first algorithms to reach human performance on a visual task

- LeCun, Boser, et al. (1989). Backpropagation Applied to Handwritten Zip Code Recognition, in Neural Computation.

  Architecture : 1000 units - 70,000 connections

- C. Cortes and V. Vapnik (1995). Support-Vector Networks, in Machine Learning Journal.

  Architecture : 1 kernel - 2 parameters

USPS ZIP code database

Goal for the class today

- Develop insights about deep learning and neural networks : when it works and when it does not work, and what it means to "work" (open discussion)
- Practical guide to deep learning optimization and engineering
- Learn about the three mysteries of deep learning... and connect to the machine learning concepts seen so far (such as approximation error, complexity, and regularization)

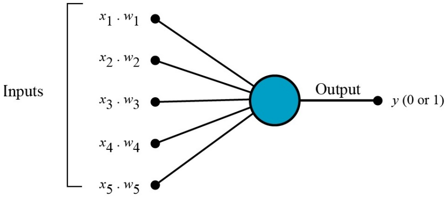
Historical perspective on neural networks

- Cybernetics (1940s-1960s)
  - Achievement : modeling and training one neuron
  - Key algorithm : Perceptron
  - Paper : Rosenblatt (1958)
- Connectionism (1980s)
  - Achievement : training one or two hidden layers
  - Key algorithm : Backpropagation
  - Paper : Rumelhart-Hinton-Williams (1986)
- Deep Learning (2007-....)
  - Achievement : training multiple layers of representation
  - Key algorithm : Stochastic gradient
  - Papers : Hinton (2006), Bengio-LeCun (2007)

First wave : 1960s

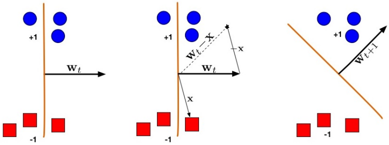
The Perceptron

Primitive neural network
Single neuron Perceptron



`Batch Perceptron`

input: A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize: $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  for $t = 1, 2, \ldots$
    if $(\exists\ i \ \text{ s.t. } \ y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0)$ then
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      output $\mathbf{w}^{(t)}$



Second wave : 1980's

Multilayer perceptrons

1. Theory : Universal approximators
2. Algorithm : Backpropagation algorithm

Second wave : 1980's

Multilayer perceptrons

1. Existence theorems of universal approximators

Stone-Weierstrass theorem

- Consider any continuous function $f : [a,b] \to \mathbb{R}$, then for any $\varepsilon > 0$, there exists a polynomial P such that :

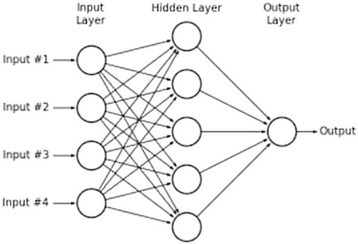
$$\sup_{x \in [a,b]} |f(x) - P(x)| < \varepsilon \; .$$

Single-Layer Neural Network
Definition

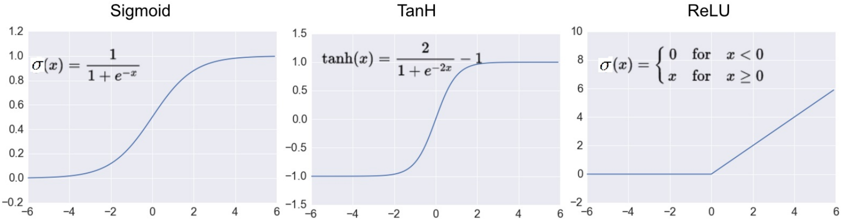
- Single-layer neural network : Let σ a 'smooth' activation function. A single-layer neural network with N units and an *activation function* σ, is a function of this form :

$$h(x) = \sum_{k=1}^{N} \sigma(a_k^T x + b), \ \forall x \in \mathbb{R}^d$$

where $a \in \mathbb{R}^d$, $b \in \mathbb{R}$, N integer. The number N corresponds to the number of *units* in the hidden layer of the network.



Activation function
Examples



Single-Layer Neural Network are universal approximators

- Cybenko's theorem : consider any continuous function $f : [0,1]^d \to \mathbb{R}$, then for any $\varepsilon > 0$, there exists a single-layer neural network $h(x) = \sum_{k=1}^{N} \sigma(a_k^T x + b)$ (i.e. some N, a, b) such that :
$$\sup_{x \in [a,b]} |f(x) - h(x)| < \varepsilon \ .$$
- Further work by Hornik-Stinchcombe-White (1989), Barron (1993).

2. Backpropagation algorithm :

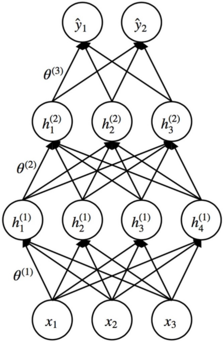
The key to multilayer perceptron calibration

Multilayer perceptron
More than one hidden layer !

- Hypothesis space : functions of the form

  $$h(x, \theta) = \sigma \circ A_m \circ \sigma \circ \ldots \circ A_2 \circ \sigma \circ A_1 x$$

  where $\theta = (A_1, \ldots, A_m)$ sequence of parameters to be estimated through learning and σ activation function applied componentwise



Backpropagation Principle

- Consider the square loss, then given a weight vector θ_1, we can evaluate the error as :

$$\mathcal{L}(\theta_1) = \frac{1}{n}\sum_{i=1}^{n}\big(h(X_i,\theta_1) - Y_i\big)^2$$

- The idea is to propagate the error backwards in the network to update θ_1 by the following rule :

$$\theta_2 = \theta_1 - \eta\nabla_\theta\mathcal{L}(\theta_1)$$

where η is the so-called learning rate.

- Consider the composition of three functions :

$$f(u) = \ell \circ \sigma \circ g(x)$$

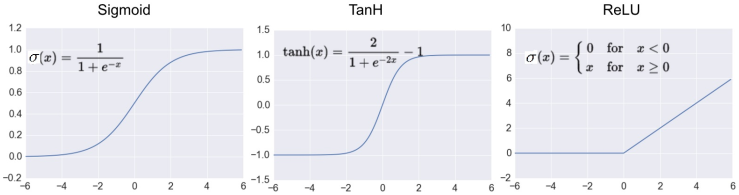
  with $t = \sigma \circ g(u)$ and $z = g(u)$ (everything in $\mathbb{R}$ here)

- The chain rule provides the expression for the derivative of f :

$$\frac{df}{du}(u) = \frac{d\ell}{dt}(t)\frac{d\sigma}{dz}(z)\frac{dg}{du}(u) = \ell'(\sigma \circ g(u))\sigma'(g(u))g'(u)$$

Background : Activation function

- Typical examples :



- For the logistic activation function : $\sigma(z) = \dfrac{1}{1+e^{-z}}$, we have by standard algebra :

$$\sigma'(z) = \frac{d\sigma}{dz}(z) = \sigma(z)(1 - \sigma(z))$$

Backpropagation
Toy example : the single unit case

- Consider a single unit (neuron) : $h(x, a) = \sigma(a^T x)$ which is connected to the output Y of the network
- The error of the predictions produced by this neuron on the training data is the following :

$$\mathcal{L}(a) = \frac{1}{n} \sum_{i=1}^{n} \ell(\sigma(a^T X_i), Y_i)$$

where $\ell(t, y) = (t - y)^2$ considering the square loss here.

- Apply the chain rule with three compositions in the case where the last function is linear
- The gradient of $\mathcal{L}$ wrt a is given by :

$$\frac{\partial \mathcal{L}}{\partial a_j}(a) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \ell}{\partial t}(\sigma(a^T X_i), Y_i)\sigma'(a^T X_i)X_{ij}$$

- Special case here : square loss, logistic activation function

$$\frac{\partial \ell}{\partial t}(t, Y_i) = 2(t - Y_i) \text{ and } \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- The gradient update applied at input 'neurons' is the following

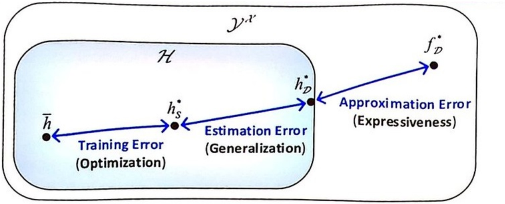
$$\frac{\partial \mathcal{L}}{\partial a_j}(a) = \frac{2}{n} \sum_{i=1}^{n} (z_i - Y_i) z_i (1 - z_i) X_{ij}$$

where $z_i = \sigma(a^T X_i)$

A new trade-off in Machine Learning
The three terms

$$\begin{aligned}\mathcal{E} &= \mathbb{E}\left[E(f_{\mathcal{F}}^{*}) - E(f^{*})\right] + \mathbb{E}\left[E(f_n) - E(f_{\mathcal{F}}^{*})\right] + \mathbb{E}\left[E(\tilde{f}_n) - E(f_n)\right] \\ &= \mathcal{E}_{\mathrm{app}} + \mathcal{E}_{\mathrm{est}} + \mathcal{E}_{\mathrm{opt}}.\end{aligned}$$

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\mathrm{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\mathrm{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\mathrm{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	$\searrow$



$f_{\mathcal{D}}^{*}$ – ground truth ($\mathrm{argmin}_{f \in \mathcal{Y}^{\mathcal{X}}} L_{\mathcal{D}}(f)$)

$h_{\mathcal{D}}^{*}$ – optimal hypothesis ($\mathrm{argmin}_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$)

h_S^{*} – empirically optimal hypothesis ($\mathrm{argmin}_{h \in \mathcal{H}} L_S(h)$)

$\bar{h}$ – returned hypothesis

Here : n sample size, ρ numerical tolerance in the optimization

Third wave : 2010s

From shallow to deep networks

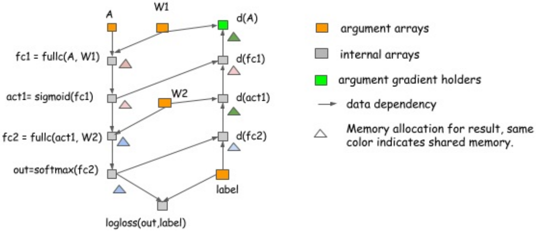
1. How to build deep networks
2. The mysteries of deep learning

From shallow to deep networks

1. How to build deep networks

Engineering of deep learning

- Software environments for deep learning designed as computational graphs (Theano, Keras, TensorFlow...)



- A computational graph is a way to represent a math function in the language of graph theory.
- In a computational graph nodes are either input values or functions for combining values.

Edges receive their weights as the data flows through the graph. Outbound edges from an input node are weighted with that input value ; outbound nodes from a function node are weighted by combining the weights of the inbound edges using the specified function.

Regularization in deep learning and why DL theory is difficult

Implicit in the objective, but lots of engineering tricks in the computational graph :

- Weight decay
- Weight sharing
- Early stopping
- Model averaging
- Dropout
- Data augmentation
- Adversarial training

Implementation of Deep Learning Examples on github

- https://github.com/enggen/Deep-Learning-Coursera/
- https://github.com/aymericdamien/TensorFlow-Examples/

The design problem (1/2)
Setup

- Denote by T the structural parameters of the deep network (architecture, activity functions, regularization modes...) and $\hat{f}_T$ the function produced by deep learning given T
- Some estimate of the predictive error $\hat{L}(\hat{f}_T)$ of the function supposed to be available (can be estimated by hold out, cross validation...).
- Finding T is key to address the estimation-approximation tradeoff

The design problem (2/2)
Selecting the structure

- Selecting the structure of a deep network is a meta-learning problem
- The optimal architecture can be obtained if it is possible to solve the following optimization problem :

$$\min_{T} \hat{L}(\hat{f}_T)$$

  which is generally nonconvex, nonsmooth
- Main approaches to find T : experience, heuristics, discrete optimization, experimental design ?

Other popular deep learning architectures

- Convolutional Neural Networks
- Recurrent Neural Networks
- Long Short Term Memory
- Auto-Encoders
- Boltzmann Machines, Belief Networks
- Generative Adversarial Networks

Beyond algorithms...

Further research topics

- Adapting these concepts to other problems :
  - either in terms of objectives : such as preference learning, scoring, ranking, anomaly detection, novelty detection...
  - or in terms of learning setups : online learning, unsupervised learning, transfer learning, multitask learning, budgeted learning, active learning...
- Scaling up : Large-scale, Monitoring...
- Adoption : Explainability, Biases, Privacy...