Mathematical Foundations of Machine Learning

Massih-Reza Amini

Pierre Gaillard

Nicolas Gast

Master of Science in Informatics in Grenoble
Master of Science in Industrial and Applied Mathematics

UGA Université Grenoble Alpes

GRENOBLE INP UGA

Program

- Part I: **Offline learning** (Sep 25. – Nov. 7)

  Massih-Reza Amini (LIG)

  - Supervised Learning:
    - Empirical Risk Minimization principle
    - Quantitative Models for Machine Learning their link with the ERM principle (+ mini-project)
    - Unconstrained Convex Optimization
    - Consistency of the ERM principle
    - Timeline of Deep Learning
    - Multi-class classification
  - Unsupervised and semi-supervised learning

Program[1]

- Part I: **Offline learning** (Sep 25. – Nov. 7)

  Massih-Reza Amini (LIG)

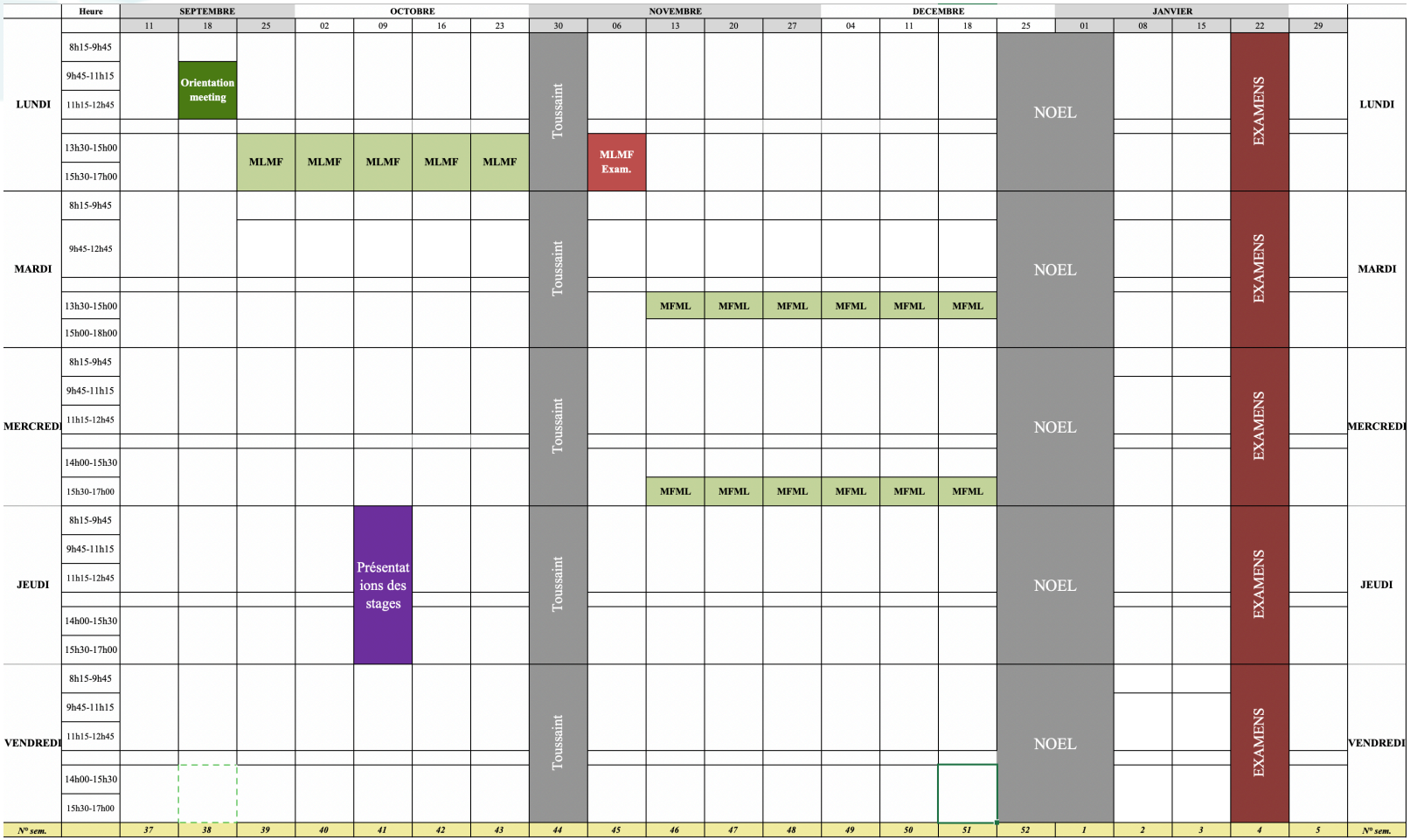
  - Supervised Learning:
    - Empirical Risk Minimization principle
    - Quantitative Models for Machine Learning their link with the ERM principle (+ mini-project)
    - Unconstrained Convex Optimization
    - Consistency of the ERM principle
    - Timeline of Deep Learning
    - Multi-class classification
  - Unsupervised and semi-supervised learning

[1] Program of Part I. based on Chapters 1, 2, 3 & 5 of [Amini 15]

Program

- Part II: **Online and Reinforcement learning**
  1. Adversarial bandits and online learning (Nov. 14 – 28)
     Pierre Gaillard (Inria)
     - Online prediction with expert advice
     - Online convex optimization (+ mini-project)
     - Adversarial bandits
  2. Reinforcement learning (Dec. 5 – 19)
     Nicolas Gast (Inria)
     - Markov decision processes (+ mini-project)
     - Classical RL algorithms
     - "Modern RL" (Deep RL, MCTS)

Schedule at a glance



	Heure	SEPTEMBRE			OCTOBRE				NOVEMBRE					DECEMBRE				JANVIER					
		11	18	25	02	09	16	23	30	06	13	20	27	04	11	18	25	01	08	15	22	29	
LUNDI	8h15-9h45								Toussaint								NOEL				EXAMENS		LUNDI
	9h45-11h15		Orientation meeting																				
	11h15-12h45																						
	13h30-15h00			MLMF	MLMF	MLMF	MLMF	MLMF		MLMF Exam.													
	15h30-17h00																						
MARDI	8h15-9h45								Toussaint								NOEL				EXAMENS		MARDI
	9h45-12h45																						
	13h30-15h00										MFML	MFML	MFML	MFML	MFML	MFML							
	15h00-18h00																						
MERCREDI	8h15-9h45								Toussaint								NOEL				EXAMENS		MERCREDI
	9h45-11h15																						
	11h15-12h45																						
	14h00-15h30																						
	15h30-17h00										MFML	MFML	MFML	MFML	MFML	MFML							
JEUDI	8h15-9h45								Toussaint								NOEL				EXAMENS		JEUDI
	9h45-11h15					Présentations des stages																	
	11h15-12h45																						
	14h00-15h30																						
	15h30-17h00																						
VENDREDI	8h15-9h45								Toussaint								NOEL				EXAMENS		VENDREDI
	9h45-11h15																						
	11h15-12h45																						
	14h00-15h30																						
	15h30-17h00																						
N° sem.		*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*	*48*	*49*	*50*	*51*	*52*	*1*	*2*	*3*	*4*	*5*	*N° sem.*

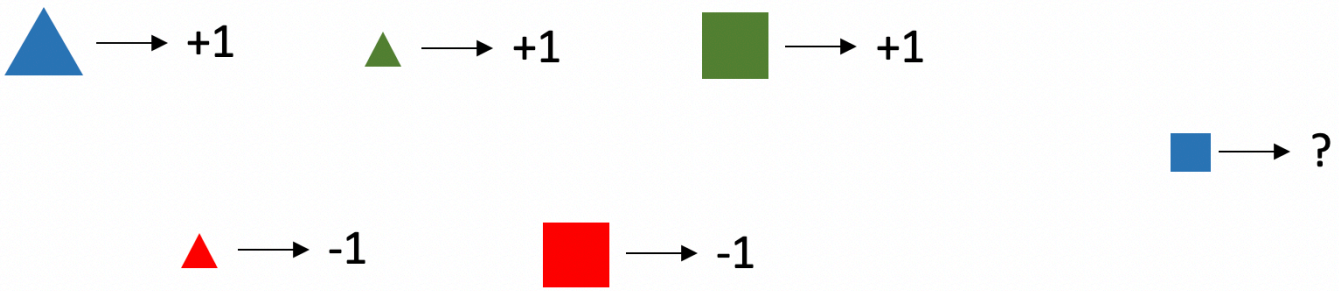
Grading: average of 3 mini-projects (30%) + average of two exams (70%).

What is Machine Learning?

- Wikipedia:*Machine Learning is a field of computer science that gives computers the ability to learn without being explicitly programmrucuted for it!*

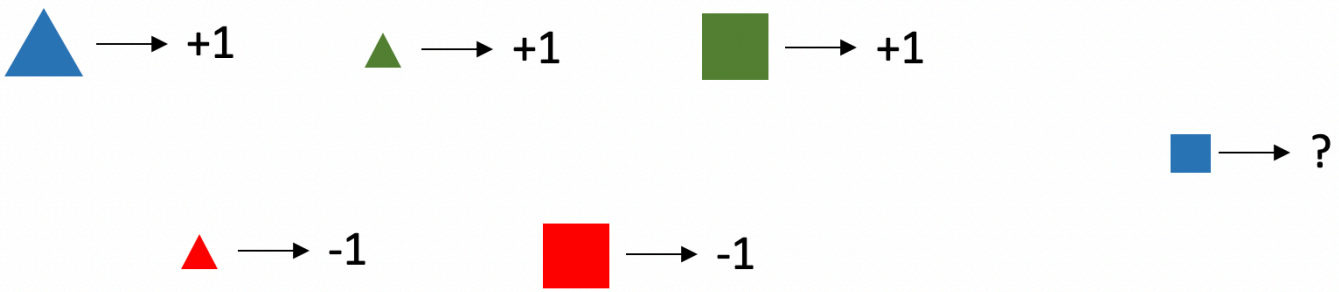
What is Machine Learning?

- Wikipedia:*Machine Learning is a field of computer science that gives computers the ability to learn without being explicitly programmrucuted for it!*



What is Machine Learning?

- Wikipedia:*Machine Learning is a field of computer science that gives computers the ability to learn without being explicitly programmrucuted for it!*



- Machine Learning programs are hence designed to do inference.

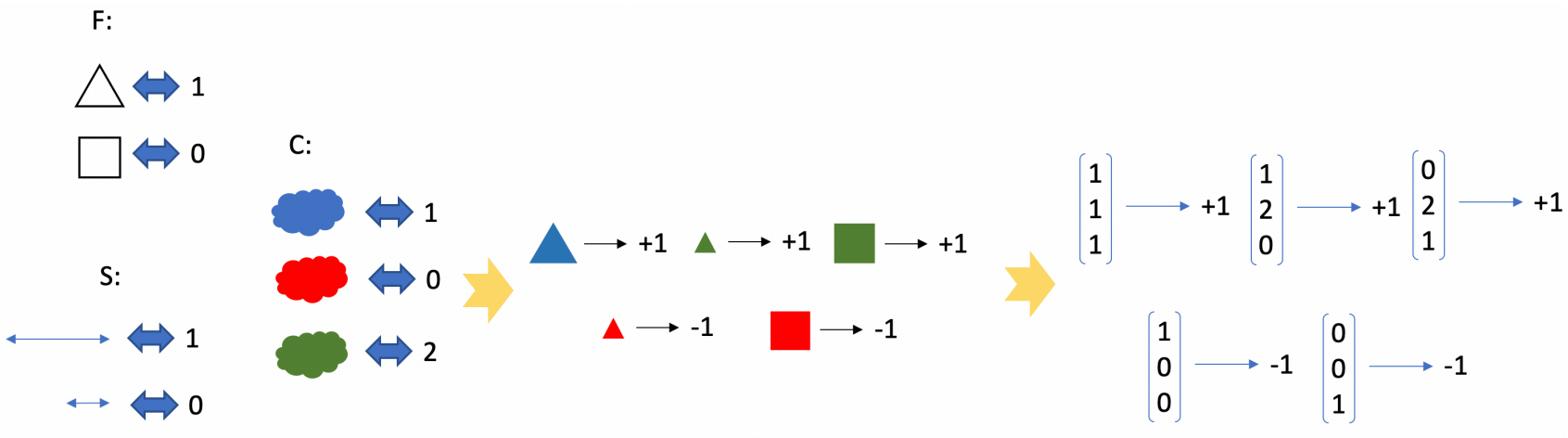
Learning and Inference

The process of inference is done in three steps:

Learning and Inference

The process of inference is done in three steps:

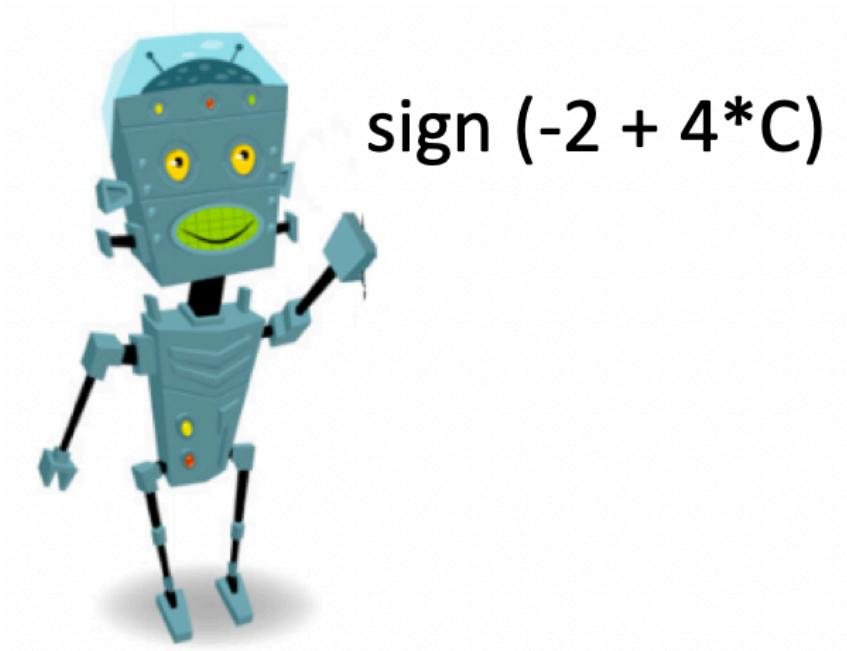
1. Observe a phenomenon



Learning and Inference

The process of inference is done in three steps:

1. Observe a phenomenon
2. Construct a model of the phenomenon



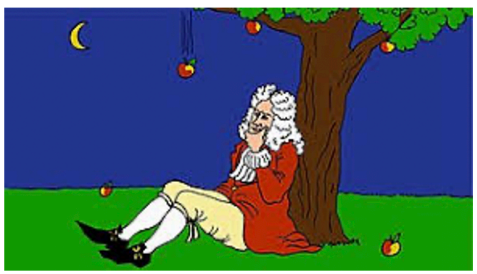
Learning and Inference

The process of inference is done in three steps:

1. Observe a phenomenon
2. Construct a model of the phenomenon
3. Do predictions

Learning and Inference

These steps are involved in more or less all natural sciences!



1. Observe a phenomenon

2. Construct a model of the phenomenon

3. Do predictions

Learning and Inference

These steps are involved in more or less all natural sciences!

1. Observe a phenomenon

2. Construct a model of the phenomenon

3. Do predictions

All that is necessary to reduce the whole nature of laws similar to those which Newton discovered with the aid of calculus, is to have a sufficient number of observations and a mathematics that is complex enough.
(Marquis de Condorcet, 1785)

Learning and Inference

These steps are involved in more or less all natural sciences!

1. Observe a phenomenon

2. Construct a model of the phenomenon

3. Do predictions

All that is necessary to reduce the whole nature of laws similar to those which Newton discovered with the aid of calculus, is to have a sufficient number of observations and a ~~mathematics that is complex enough.~~ GPU that is powerful enough.
(Marquis de Condorcet, 1785)

Induction vs. deduction

- Induction is the process of deriving general principles from particular facts or instances using mathematics.

People who wish to analyze nature without using mathematics must settle for a reduced understanding.
(Richard Feynman)

Induction vs. deduction

- Induction is the process of deriving general principles from particular facts or instances using mathematics.

- Deduction is, in the other hand, the process of reasoning in which a conclusion follows necessarily from the stated premises; it is an inference by reasoning from the general to the specific.

  This is how mathematicians prove theorems from axioms.

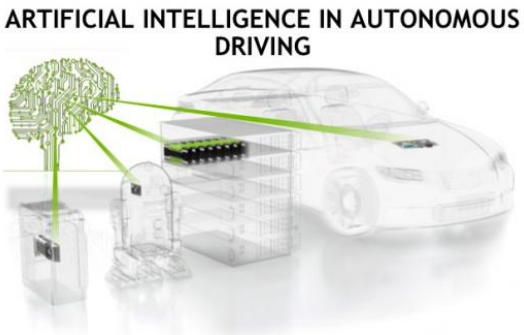
Main Hypotheses

Two types of hypotheses:

- Past observations are related to the future ones
  $\rightarrow$ The phenomenon is stationary

- Observations are independently generated from a source
  $\rightarrow$ Notion of independence

Some popular applications



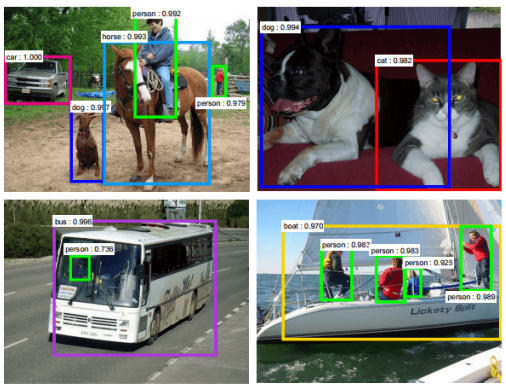
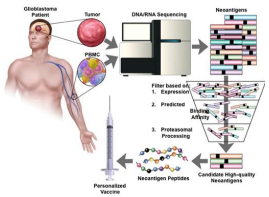
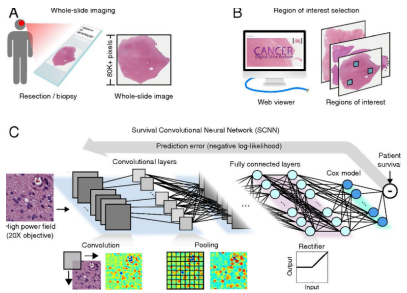


Image recognition



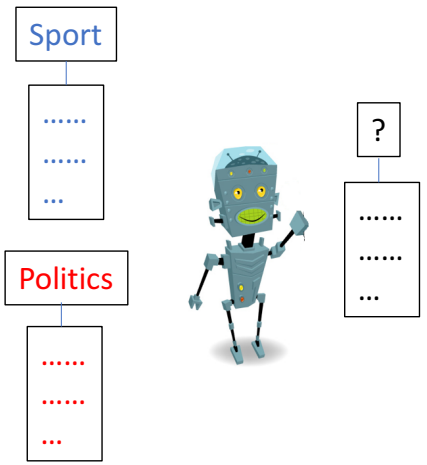


Medicine

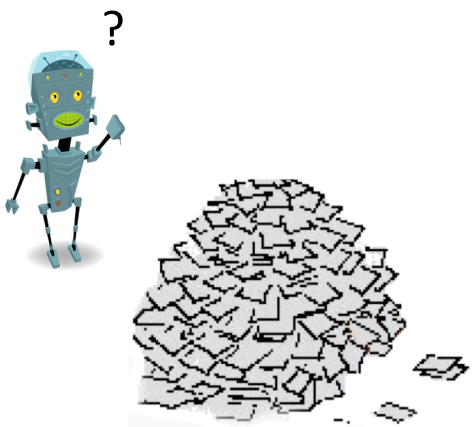
Machine Learning for

Finance and buisness

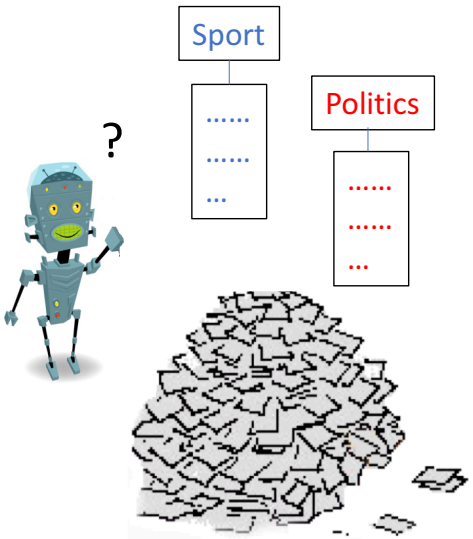
Three main Frameworks



Supervised Learning



Unsupervised Learning



Semi-supervised Learning

All related to Inductive reasoning !

Empirical risk minimization

Probabilistic model

Relations between the past and future observations.

- Independence: Each new observation provides a maximum individual information,
- identically Distributed : Observations provide information on the phenomenon which generates the observations.

Formally

We consider an input space $\mathcal{X} \subseteq \mathbb{R}^d$ and an output space $\mathcal{Y}$.

Assumption: Example pairs $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ are *identically* and *independently* distributed (`i.i.d`) with respect to an unknown but fixed probability distribution $\mathcal{D}$.

Samples: We observe a sequence of m pairs of examples $(\mathbf{x}_i, y_i)$ generated `i.i.d` from $\mathcal{D}$.

Aim: Construct a prediction function $f : \mathcal{X} \to \mathcal{Y}$ which predicts an output y for a given new $\mathbf{x}$ with a minimum probability of error.

Notations

Symbol	Definition
$\mathcal{X} \subseteq \mathbb{R}^d$	Input space
$\mathcal{Y}$	Output space
$\mathcal{S} = (\mathbf{x}_i, y_i)_{1 \leq i \leq m}$	Training set of size m
$\mathcal{D}$	Probability distribution generating the data (i.i.d)
$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$	Instantaneous loss
$\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$	Class of functions
$\mathcal{L}(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(f(x), y)]$	Generalization error of
$\hat{\mathcal{L}}_m(f, S)$ or $\hat{\mathcal{L}}(\boldsymbol{w})$	Empirical Loss of f over S
$\boldsymbol{w}$	Parameters of the prediction function
$\mathbb{1}_\pi$	Indicator function equals 1 if π is true and 0 otherwise
$\mathcal{R}_m(\mathcal{F})$	Rademacher complexity of the class of functions $\mathcal{F}$
$\hat{\mathcal{R}}_m(\mathcal{F}, S)$	Empirical Rademacher complexity of $\mathcal{F}$ estimated over S

Supervised Learning

- Discriminant models directly find a classification function $f : \mathcal{X} \to \mathcal{Y}$ from a given class of functions $\mathcal{F}$;
- The function found should be the one having the lowest probability of error

$$\mathcal{L}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(f(\mathbf{x}), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(\mathbf{x}), y) d\mathcal{D}(\mathbf{x}, y)$$

Where ℓ is an instantaneous loss defined as

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$$

The risk function considered in classification is usually the misclassification error:

$$\forall(\mathbf{x}, y); \ell(f(\mathbf{x}), y) = \mathbb{1}_{f(\mathbf{x})\neq y}$$

Where $\mathbb{1}_\pi$ is equal to 1 if the predicate π is true and 0 otherwise.

Empirical risk minimization (ERM): the 1^{st} ML principle

- As the probability distribution $\mathcal{D}$ is unknown, the analytic form of the true risk cannot be driven, so the prediction function cannot be found directly on $\mathcal{L}(f)$.
- Empirical risk minimization (ERM) principle: Find f by minimizing the unbiased estimator of its generalization error $\mathcal{L}(f)$ on a given training set $S = (\mathbf{x}_i, y_i)_{i=1}^m$:

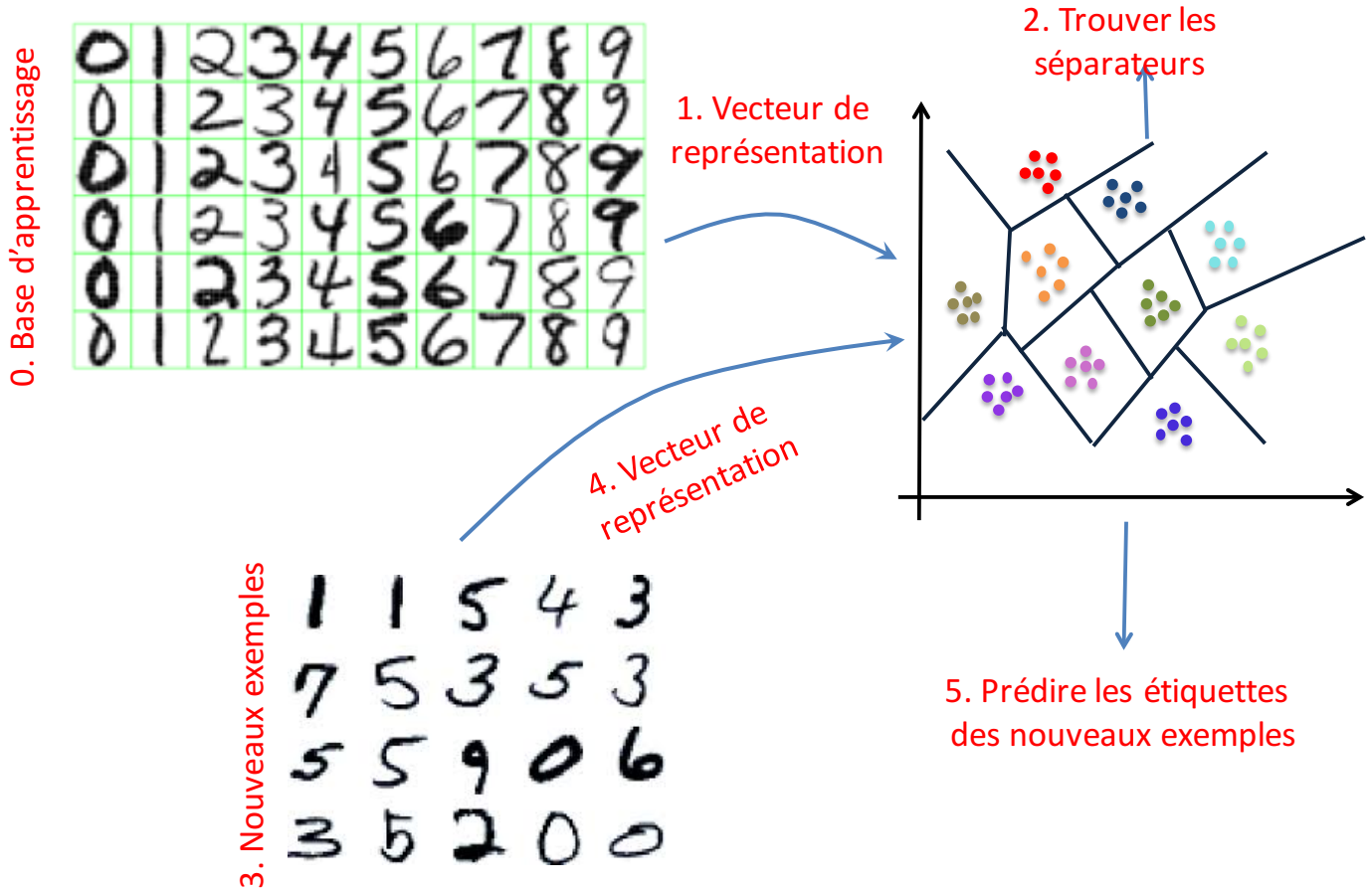
$$\hat{\mathcal{L}}_m(f, S) = \frac{1}{m} \sum_{i=1}^{m} \ell(f(\mathbf{x}_i), y_i)$$

- However, without restricting the class of functions this is not the right way of proceeding (occam razor) ...

Empirical Risk Minimization principle

- From a finite set of observations, called the training set, constituted by the vector representation of examples and their desired outputs
- Find a function that associates the vector representation of an observation to its desired output, by minimizing the empirical error of the function on the training set.
- This function is sought to make few errors on unseen examples.

Example

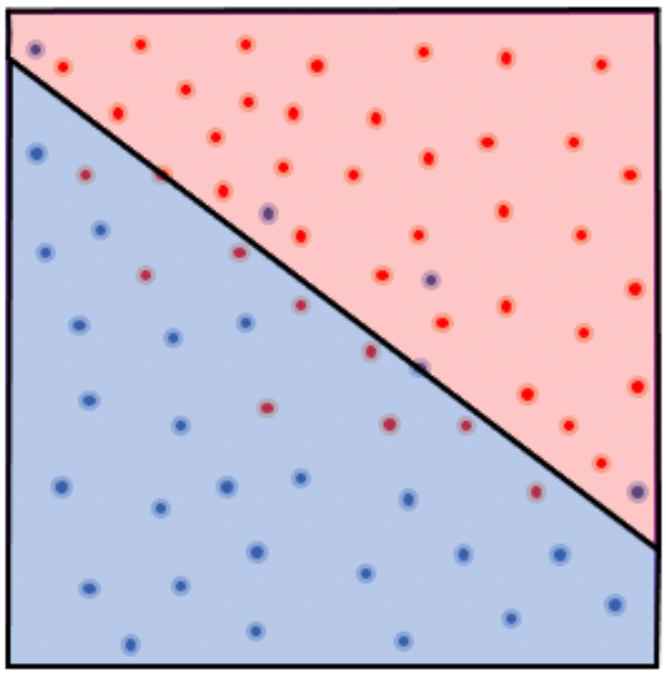

0. Base d'apprentissage
1. Vecteur de représentation
2. Trouver les séparateurs
3. Nouveaux exemples
4. Vecteur de représentation
5. Prédire les étiquettes des nouveaux exemples

Is Inference finally, interpolation?

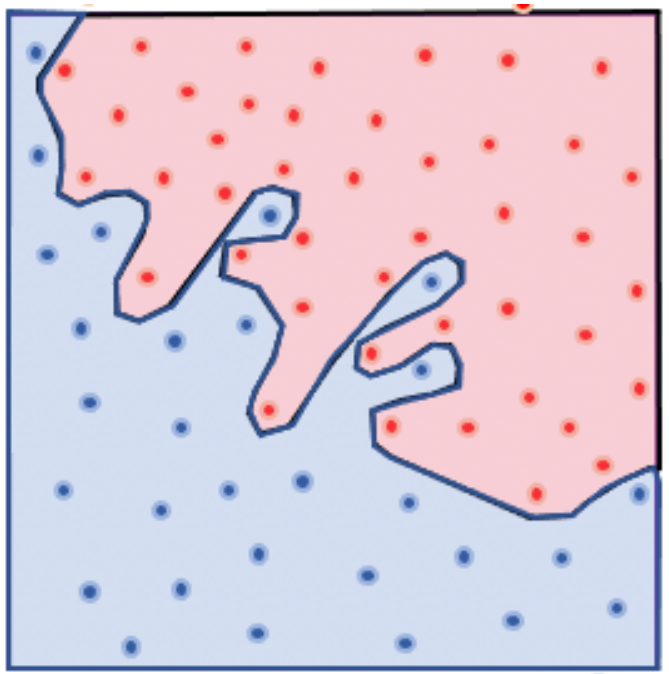
In fact, it is always possible to construct a function that exactly fits the data.

Is Inference finally, interpolation?

In fact, it is always possible to construct a function that exactly fits the data.



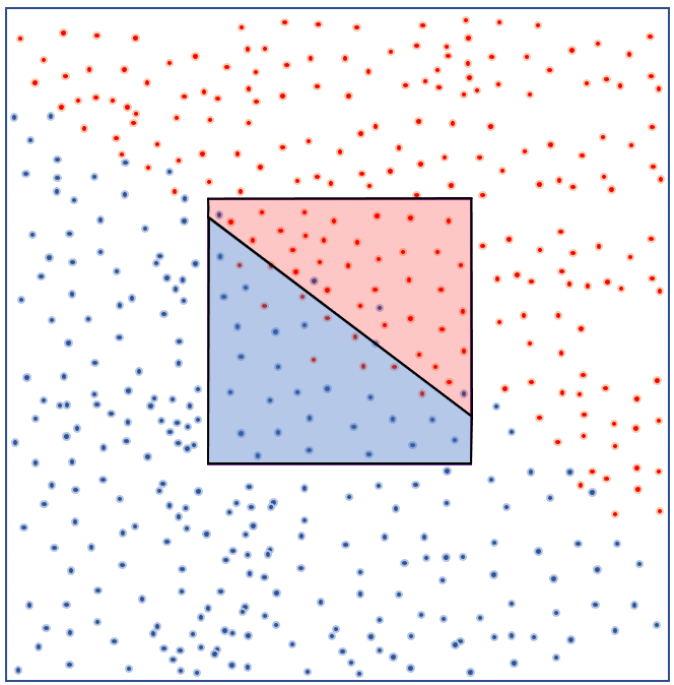
Simple model on train



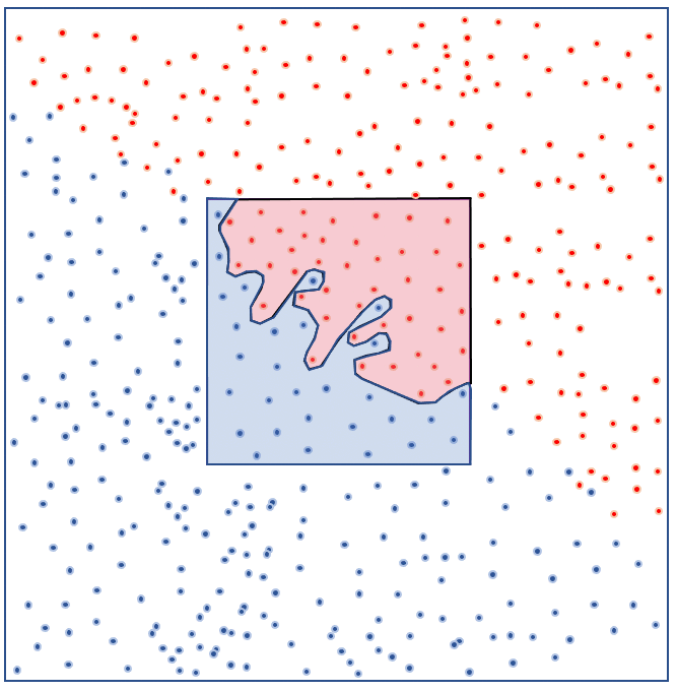
Complex model on train

Is Inference finally, interpolation?

In fact, it is always possible to construct a function that exactly fits the data.



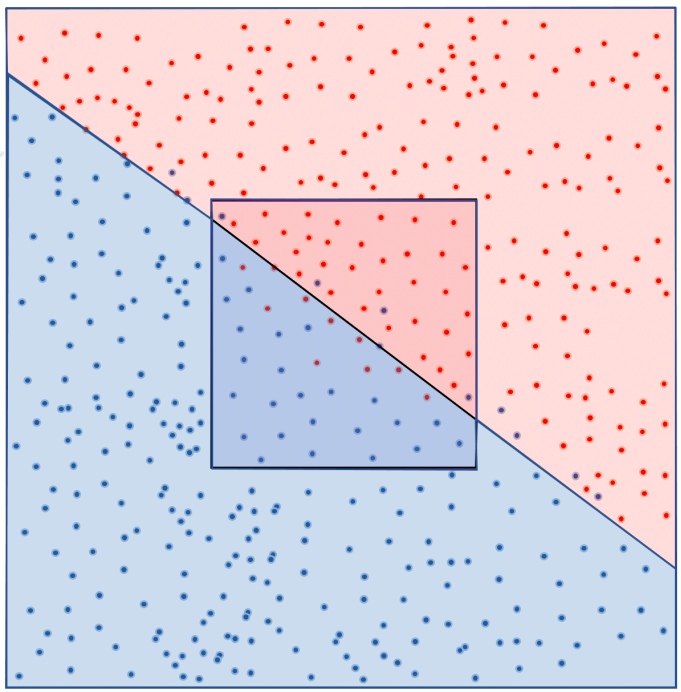
Simple model on test



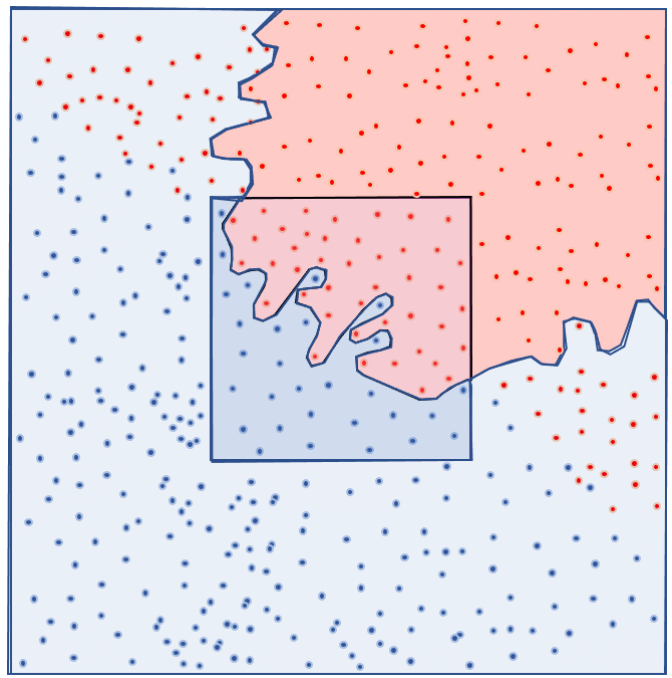
Complex model on test

Is Inference finally, interpolation?

In fact, it is always possible to construct a function that exactly fits the data.



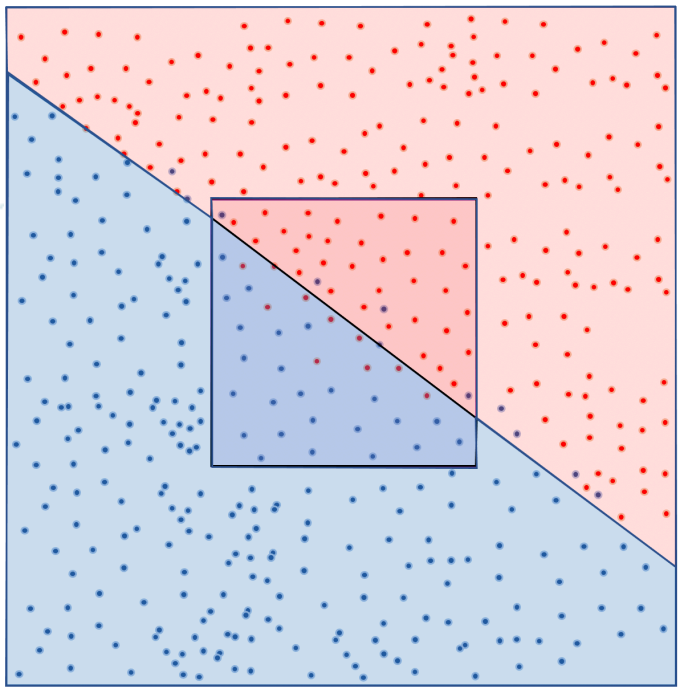
Simple model on test



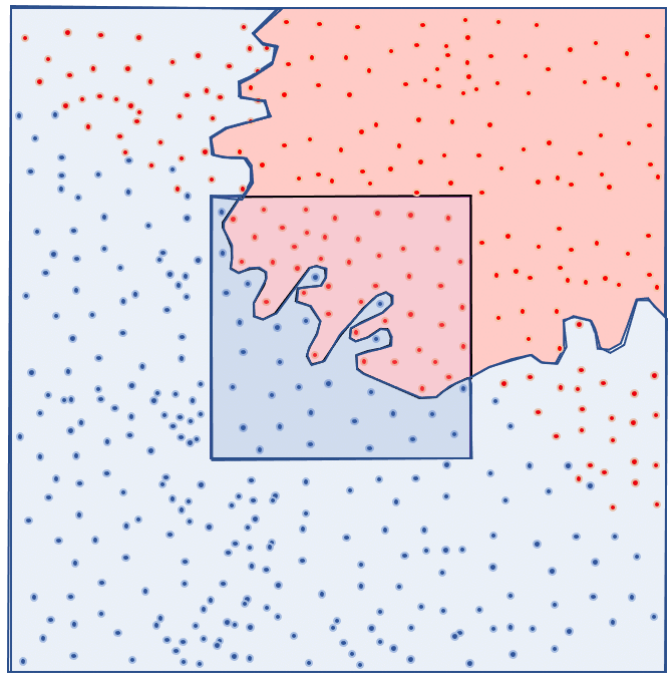
Complex model on test

Is Inference finally, interpolation?

In fact, it is always possible to construct a function that exactly fits the data.



Simple model on test



Complex model on test

But inference is not just interpolation, as the aim is to predict well on unseen examples!

Occam razor

Idea: Take the most simplest model for searching repetitions in the observed phenomenon and do inference on new examples from the passed ones ...

Simplicity is measured by ...

1. the number de parameters,
2. the number of constantes,
3. ...

Occam razor

Idea: Take the most simplest model for searching repetitions in the observed phenomenon and do inference on new examples from the passed ones ...

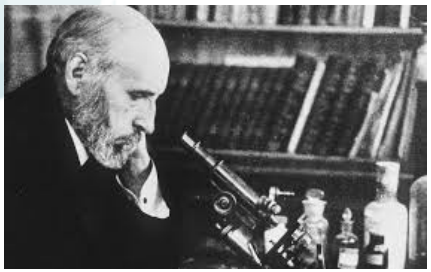
Simplicity is measured by ...

1. the number de parameters,
2. the number of constantes,
3. ...

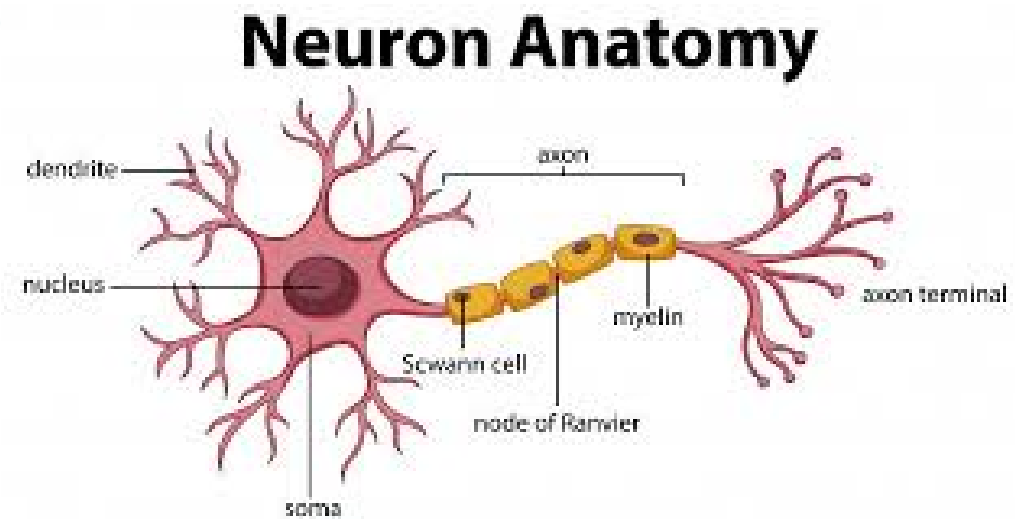
This gives raise to the second principle in Machine Learning, called *Structural Risk Minimization*, which states that learning is a compromise between low empirical error and a high capacity of interpolation.

Quantitative Learning Models

First attempts to build learnable artificial models

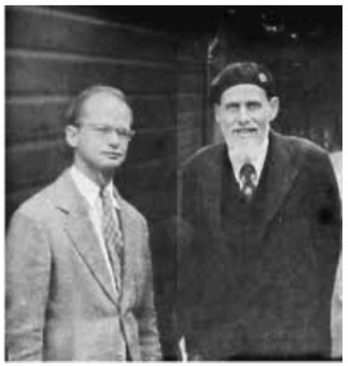


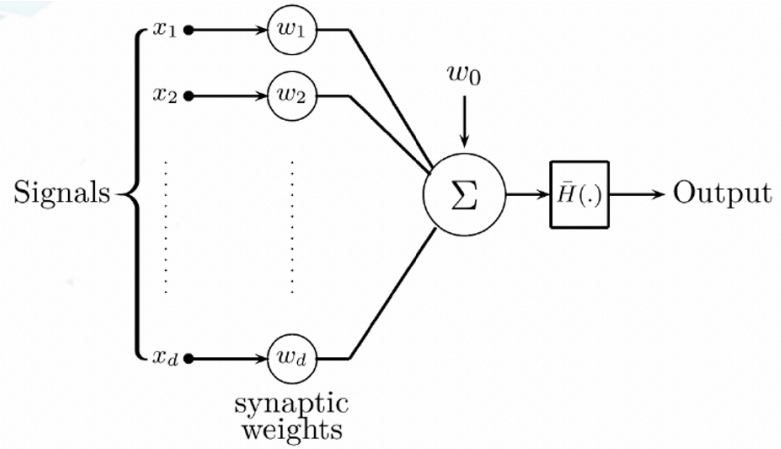
It begun at the end of the 19^{th} century with the works of Santiago Ramón y Cajal who first represented the biological neuron:



- Human brain contains about 86 billion neurons.

MuCulloch & Pitts formal neuron (1943)

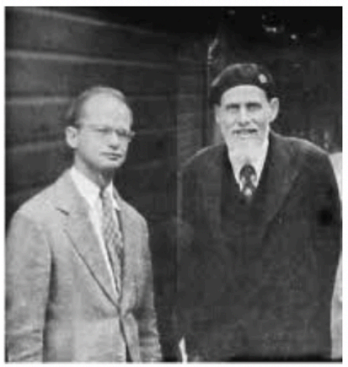


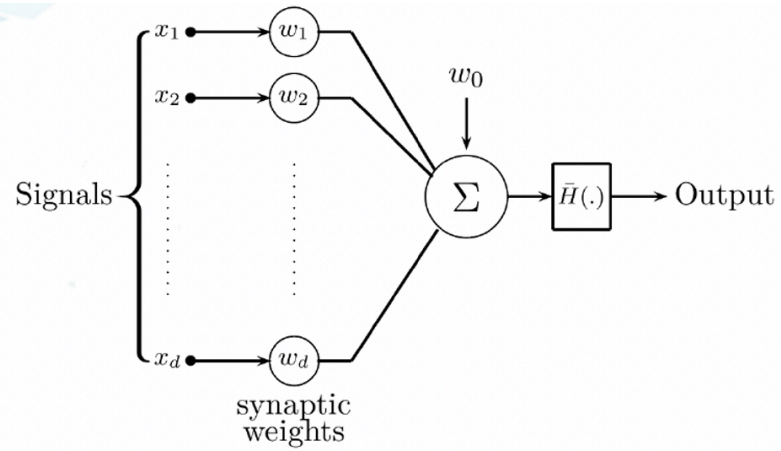


- Linear prediction function

$$\begin{aligned} h_{\boldsymbol{w}} : \mathbb{R}^d &\rightarrow \mathbb{R} \\ \mathbf{x} &\mapsto \langle \bar{\boldsymbol{w}}, \mathbf{x} \rangle + w_0 \end{aligned}$$

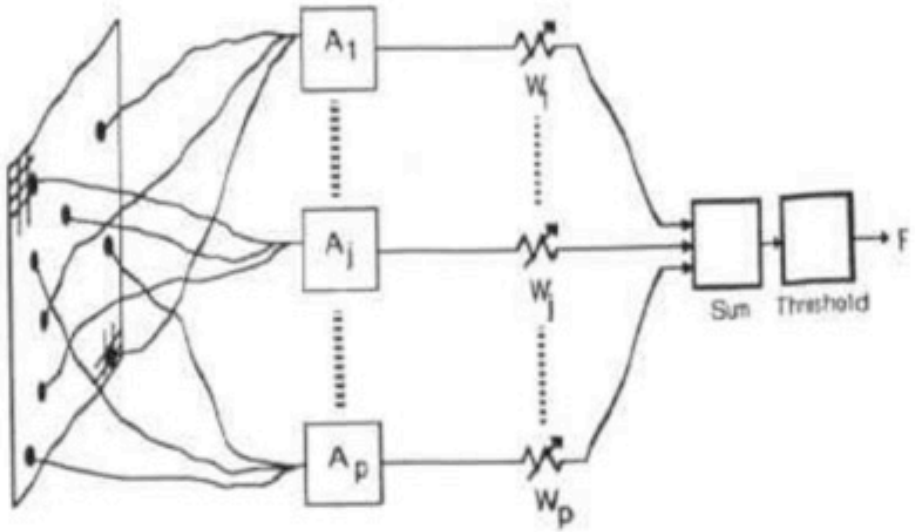
MuCulloch & Pitts formal neuron (1943)

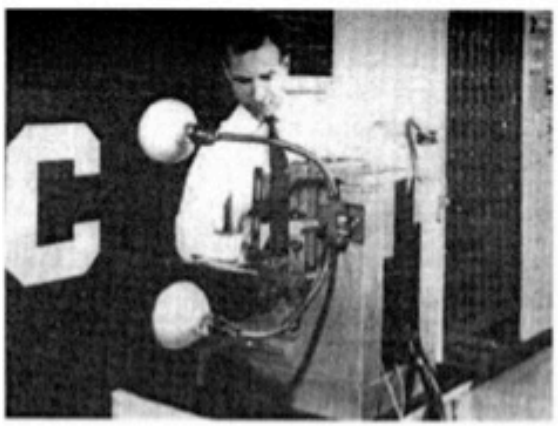




- Linear prediction function
- Different learning rules have been proposed - the most popular one was the Hebb's rule (1949): *neurons that fire together, wire together.*

Perceptron [Rosenblatt, 1958]



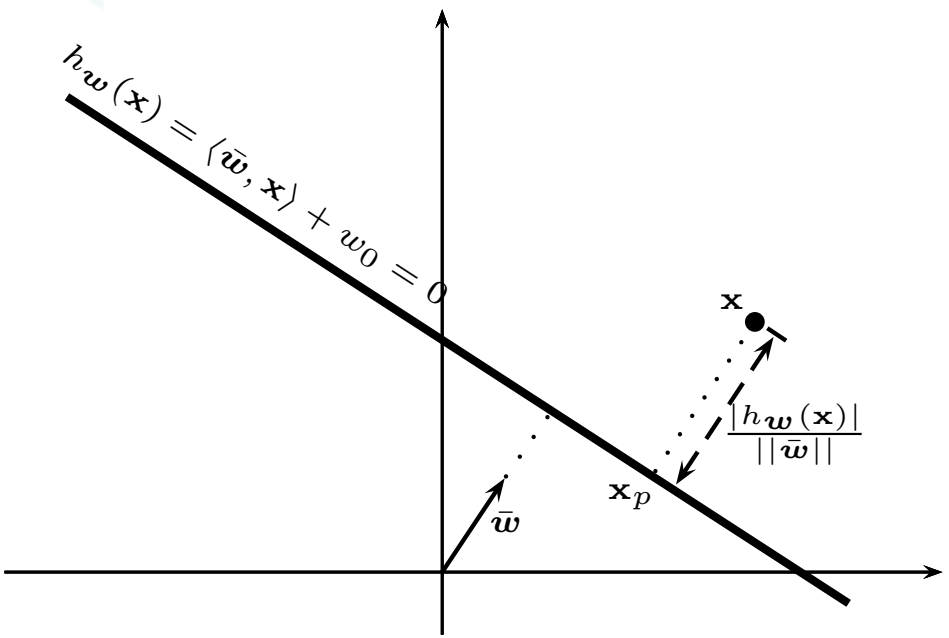


Perceptron [Rosenblatt, 1958]

- Linear prediction function

$$\begin{array}{rccl} h_{\boldsymbol{w}} : \mathbb{R}^d & \rightarrow & \mathbb{R} \\ \mathbf{x} & \mapsto & \langle \bar{\boldsymbol{w}}, \mathbf{x} \rangle + w_0 \end{array}$$

- A principle way to learn: find the parameters $\boldsymbol{w} = (\bar{\boldsymbol{w}}, w_0)$ by minimising the distance between the misclassified examples to the decision boundary.



Learning Perceptron parameters

- Objective function for a misclassified example $(\mathbf{x}_{i'}, y_{i'})$

$$\ell_p(h_{\boldsymbol{w}}(\mathbf{x}_{i'}), y_{i'}) = -y_{i'}(\langle \bar{\boldsymbol{w}}, \mathbf{x}_{i'} \rangle + w_0)$$

- Update rule: Gradient descente

$$\forall t \geq 1, \boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \nabla_{\boldsymbol{w}^{(t-1)}} \ell_p(h_{\boldsymbol{w}^{(t-1)}}(\mathbf{x}_{i'}), y_{i'})$$

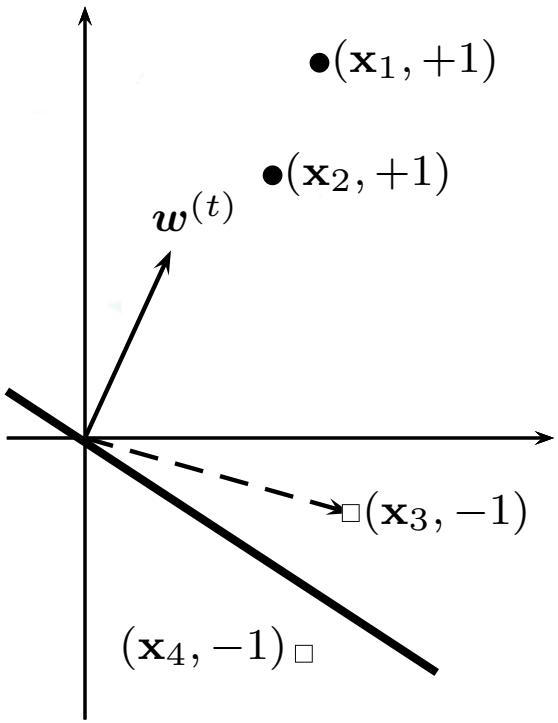
- Derivatives of with respect to the parameters

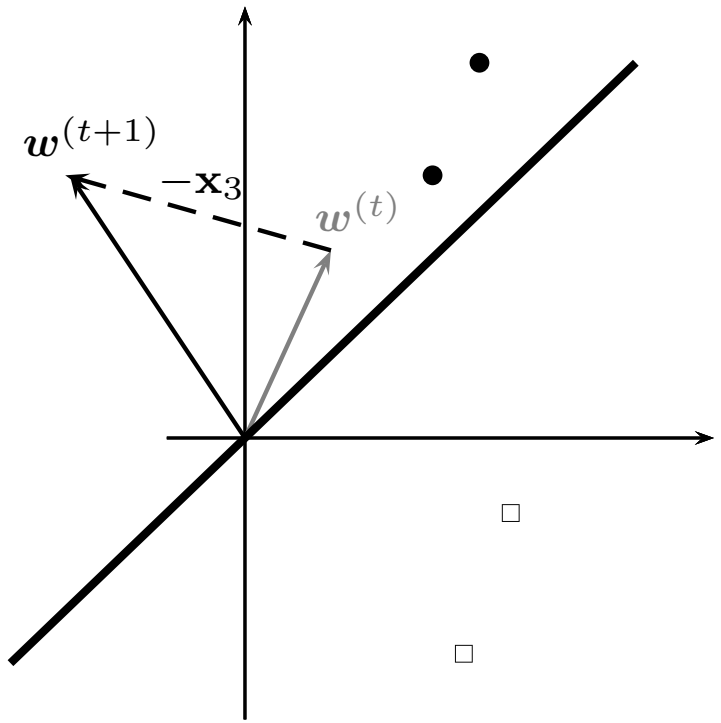
$$\begin{aligned} \frac{\partial \ell_p(h_{\boldsymbol{w}}(\mathbf{x}_{i'}), y_{i'})}{\partial w_0} &= -y_{i'}, \\ \nabla \ell_p(h_{\boldsymbol{w}}(\mathbf{x}_{i'}), y_{i'}) &= -y_{i'} \mathbf{x}_{i'} \end{aligned}$$

- Stochastic gradient descente

$$\forall (\mathbf{x}, y), \text{ if } y(\langle \bar{\boldsymbol{w}}, \mathbf{x} \rangle + w_0) \leq 0 \text{ then } \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} + \eta \begin{pmatrix} y \\ y\mathbf{x} \end{pmatrix}$$

Graphical depiction of the online update rule





Perceptron (algorithm)

Algorithm 1 The algorithm of perceptron

Training set $S = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, m\}\}$
Initialize the weights $\boldsymbol{w}^{(0)} \leftarrow 0$
$t \leftarrow 0$
Learning rate $\eta > 0$
repeat
    Choose randomly an example $(\mathbf{x}^{(t)}, y^{(t)}) \in S$
    if $y \left\langle \boldsymbol{w}^{(t)}, \mathbf{x}^{(t)} \right\rangle < 0$ **then**
        $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \times y^{(t)}$
        $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \eta \times y^{(t)} \times \mathbf{x}^{(t)}$
        $t \leftarrow t + 1$
    end if
until $t > T$

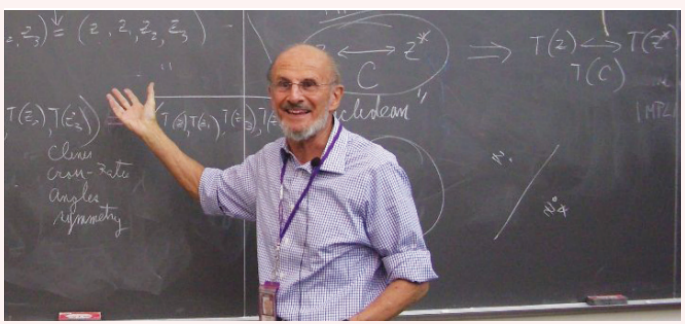
- ☞ Generic program that can be applied to any binary classification problems (i.e. the Wikipedia definition of ML)
- ☞ But does this program converge?

Perceptron (convergence)

Theorem (Novikoff, 1962)

Let $S = (\mathbf{x}_i, y_i)_{1 \leq i \leq m}$ *be a training set of size* m*; if there exists a weight* $\bar{\boldsymbol{w}}^*$*, such that* $\forall i \in \{1, \ldots, m\}, y_i \times \langle \bar{\boldsymbol{w}}^*, \mathbf{x}_i \rangle > 0$*, then, by denoting* $\rho = \min_{i \in \{1,\ldots,m\}} \left(y_i \left\langle \frac{\bar{\boldsymbol{w}}^*}{||\bar{\boldsymbol{w}}^*||}, x_i \right\rangle \right)$*, and,* $R = \max_{i \in \{1,\ldots,m\}} ||\mathbf{x}_i||$*, and,* $\bar{\boldsymbol{w}}^{(0)} = 0$*,* $\eta = 1$*, the number of iterations* k *to find the hyperplan with normal vector* $\bar{\boldsymbol{w}}^*$ *that separates the classes is bounded by :*

$$k \leq \left\lceil \left(\frac{R}{\rho} \right)^2 \right\rceil$$



ADAptive LInear NEuron [Widrow & Hoff, 1960]

- Find parameters of the formal neuron by minimising the the mean squared error

$$\hat{\mathcal{L}}(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - h_{\boldsymbol{w}}(\mathbf{x}_i))^2$$

- Update rule : stochastic gradient descent algorithm with a learning rate $\eta > 0$

$$\forall(\mathbf{x}, y), \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} + \eta(y - h_{\boldsymbol{w}}(\mathbf{x})) \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \tag{1}$$

Adaline

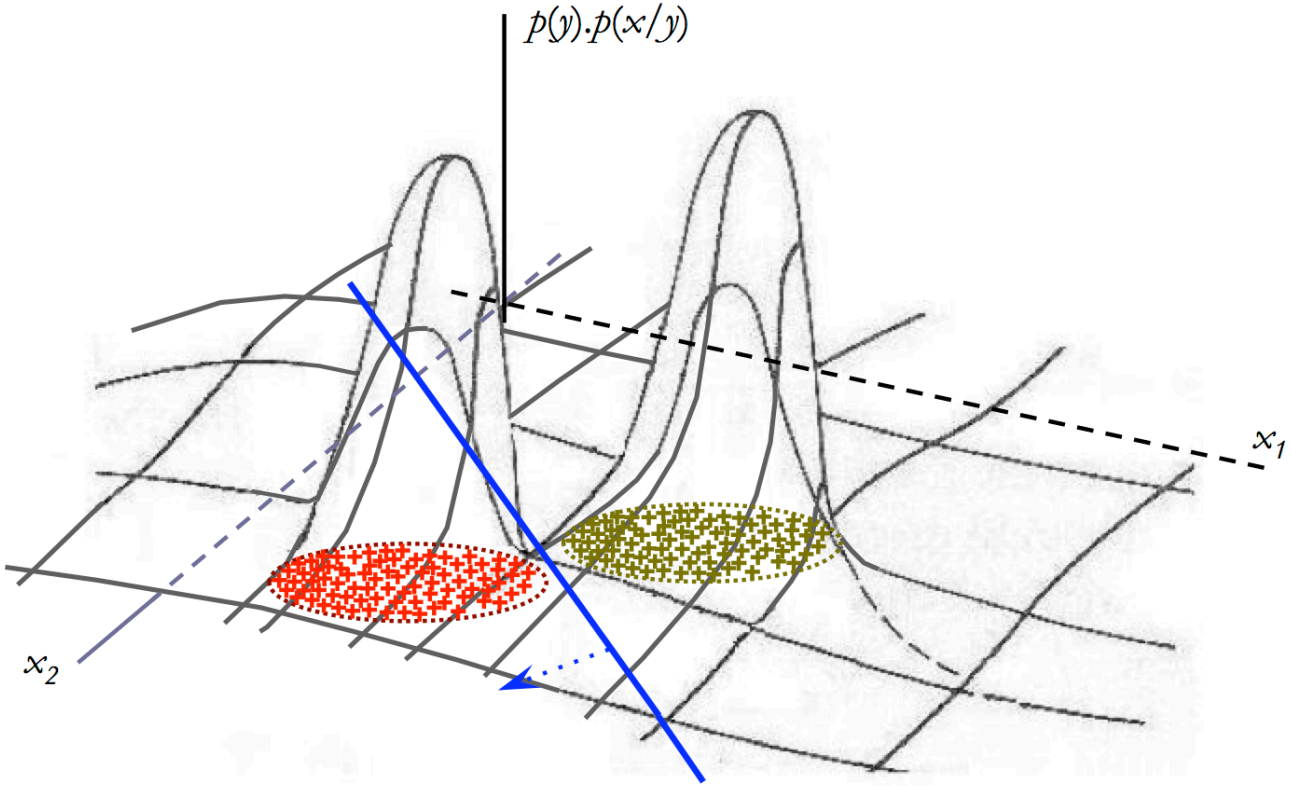
- ADAptive LInear NEuron
- Linear prediction function :

$$\begin{array}{rccc} h_{\boldsymbol{w}} : \mathcal{X} & \to & \mathbb{R} \\ \mathbf{x} & \mapsto & \langle \bar{w}, \mathbf{x} \rangle + w_0 \end{array}$$

Algorithm 2 The algorithm of Adaline

Training set $S = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, m\}\}$
Initialize the weights $w^{(0)} \leftarrow 0$
$t \leftarrow 0$
Learning rate $\eta > 0$
repeat
    Choose randomly an example $(\mathbf{x}^{(t)}, y^{(t)}) \in S$
    $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \times (y^{(t)} - h_w(\mathbf{x}^{(t)}))$
    $\bar{w}^{(t+1)} \leftarrow \bar{w}^{(t)} + \eta \times (y^{(t)} - h_w(\mathbf{x}^{(t)})) \times \mathbf{x}^{(t)}$
    $t \leftarrow t + 1$
until $t > T$

Generative models for classification



Each example $\mathbf{x}$ is supposed to be generated by a mixture model of parameters Θ:

$$P(\mathbf{x} \mid \Theta) = \sum_{k=1}^{K} P(y = k) P(\mathbf{x} \mid y = k, \Theta)$$

Generative models for classification

- The aim is then to find the parameters Θ for which the model explains the best the observations,
- That is done by maximizing the complete log-likelihood of data $S = \{(\mathbf{x}_i, y_i); i \in \{1, \ldots, m\}\}$

$$L(S, \Theta) = \ln \prod_{i=1}^{m} P(\mathbf{x}_i, y_i \mid \Theta)$$

- Classical density functions are Gaussian density functions

$$P(\mathbf{x} \mid y = k, \Theta) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu_k)^\top \Sigma_k^{-1} (\mathbf{x}-\mu_k)}$$

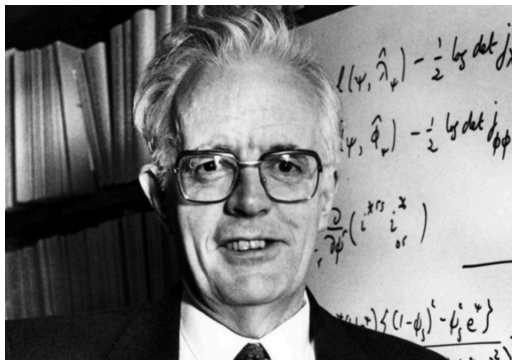
Generative models for classification

- Once the parameters Θ are estimated; the generative model can be used for classification by applying the Bayes rule:

$$\begin{aligned} \forall \mathbf{x}; y^* &= \underset{k}{\operatorname{argmax}}\, P(y = k \mid \mathbf{x}) \\ &\propto \underset{k}{\operatorname{argmax}}\, P(y = k) \times P(\mathbf{x} \mid y = k, \Theta) \end{aligned}$$

- Problem: in most real life applications the distributional assumption over data does not hold,

Logistic Regression [Cox 58]



- The Logistic Regression model does not make any assumption except that

$$\ln \frac{P(y=1 \mid \mathbf{x})}{P(y=0 \mid \mathbf{x})} = \langle \bar{w}, \mathbf{x} \rangle + w_0 = h_{\boldsymbol{w}}(\mathbf{x})$$

- The logistic regression has been proposed to model the posterior probability of classes via logistic functions.

$$P(y \mid \mathbf{x}) = (\sigma(h_{\boldsymbol{w}}(\mathbf{x})))^y (1 - \sigma(h_{\boldsymbol{w}}(\mathbf{x})))^{1-y}$$

Logistic Regression

- where

$$\begin{aligned} \sigma : \mathbb{R} &\rightarrow]0,1[\\ z &\mapsto \frac{1}{1+e^{-z}} \end{aligned}$$

with the derivative:

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

- Model parameters $\boldsymbol{w}$ are found by maximizing the complete log-liklihood, which by assuming that m training examples are generated independently, writes

$$\begin{aligned} \ln \prod_{i=1}^{m} P(\mathbf{x}_i, y_i \mid \Theta, \boldsymbol{w}) &= \ln \prod_{i=1}^{m} P(y_i \mid \mathbf{x}_i, \boldsymbol{w}) + \ln \prod_{i=1}^{m} P(\mathbf{x}_i \mid \Theta) \\ &\approx \sum_{i=1}^{m} \ln \left[(\sigma(h_{\boldsymbol{w}}(\mathbf{x}_i)))^{y_i} (1-\sigma(h_{\boldsymbol{w}}(\mathbf{x}_i))^{1-y_i} \right] \end{aligned}$$

Logistic Regression : link with the ERM principle

- The maximization of the log-liklihood $\mathcal{L}$ is equivalent to the minimization of the empirical logistic loss in the case where $\forall i, y_i \in \{-1, +1\}$.

$$\hat{\mathcal{L}}(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} \ln(1 + e^{-y_i h_w(\mathbf{x}_i)})$$

- Update rule using stochastic gradient descent algorithm with a learning rate $\eta > 0$

$$\forall (\mathbf{x}, y), \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} + \eta y (1 - \sigma(h_{\boldsymbol{w}}(\mathbf{x}))) \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \qquad (2)$$

Logistic Regression

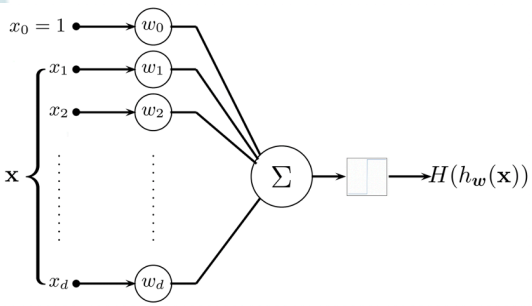
- Linear prediction function :

$$\begin{array}{rccc} h_{\boldsymbol{w}} : \mathcal{X} & \to & \mathbb{R} \\ \mathbf{x} & \mapsto & \langle \bar{w}, \mathbf{x} \rangle + w_0 \end{array}$$

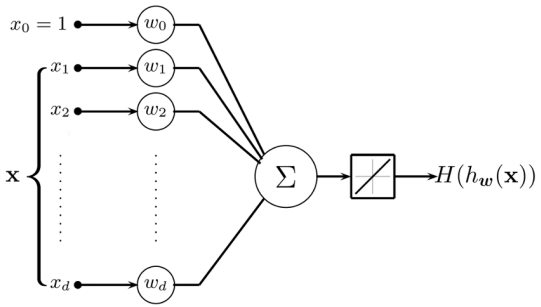
Algorithm 3 The algorithm of Logistic Regression

Training set $S = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, m\}\}$
Initialize the weights $w^{(0)} \leftarrow 0$
$t \leftarrow 0$
Learning rate $\eta > 0$
repeat
    Choose randomly an example $(\mathbf{x}^{(t)}, y^{(t)}) \in S$
    $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \times y^{(t)}(1 - \sigma(h_{\boldsymbol{w}^{(t)}}(\mathbf{x}^{(t)})))$
    $\bar{w}^{(t+1)} \leftarrow \bar{w}^{(t)} + \eta \times y^{(t)}(1 - \sigma(h_{\boldsymbol{w}^{(t)}}(\mathbf{x}^{(t)}))) \times \mathbf{x}^{(t)}$
    $t \leftarrow t + 1$
until $t > T$

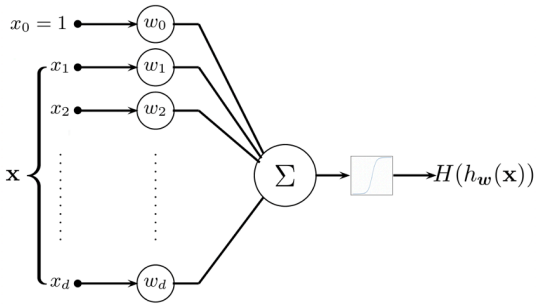
Formal models



Perceptron



Adaline



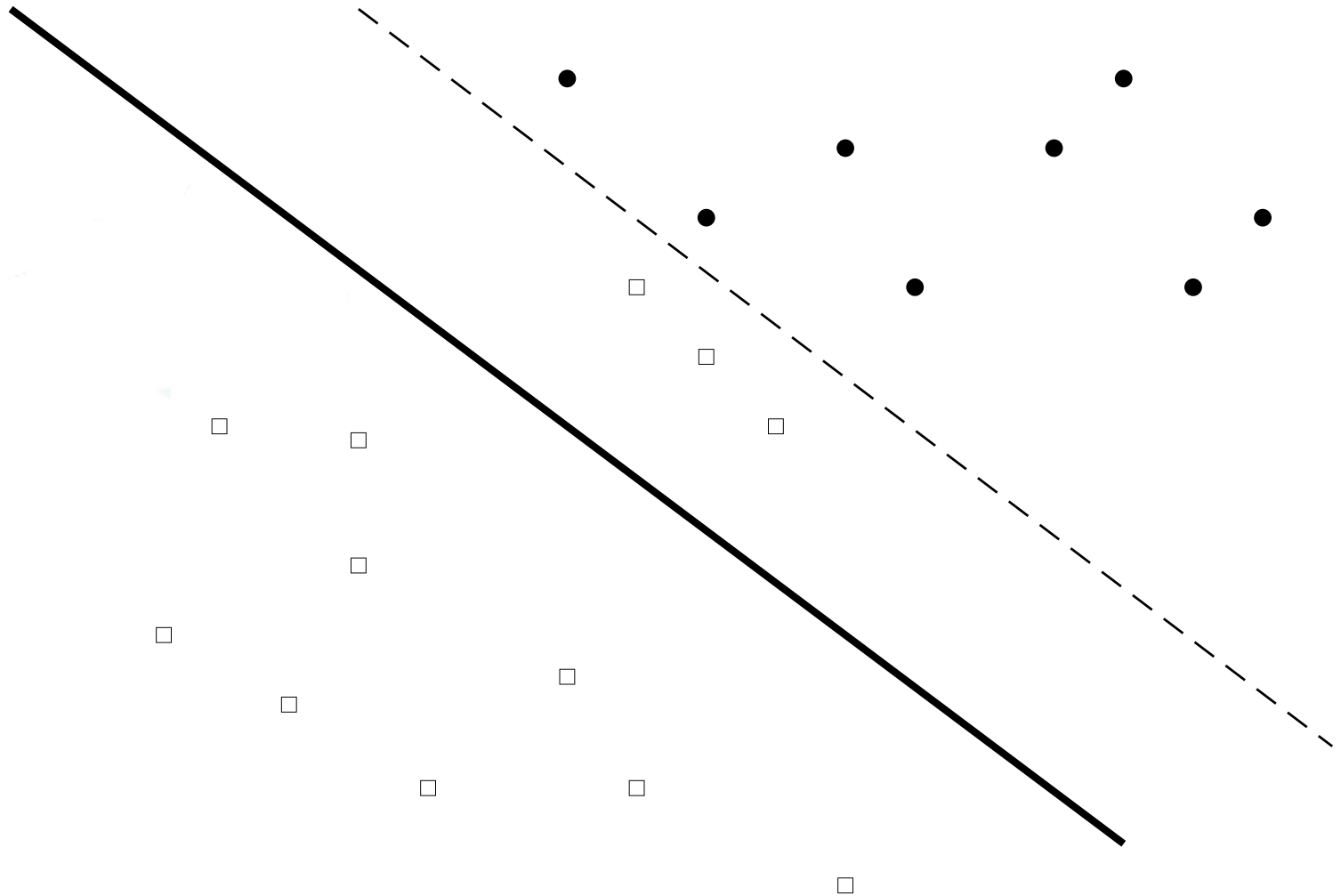
Logistic Regression

- Update rule

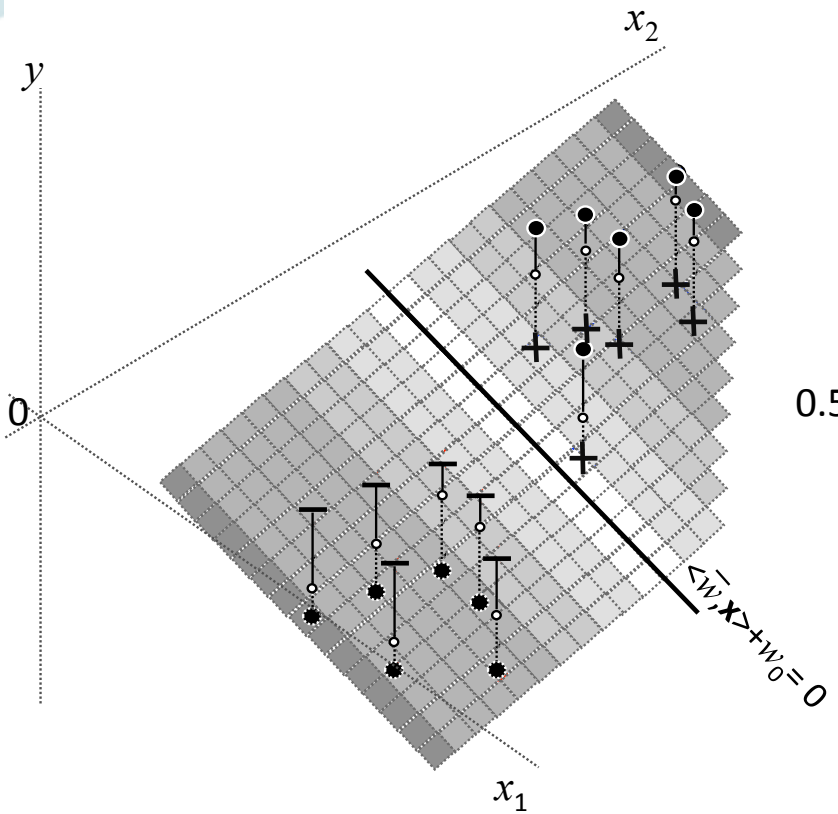
$$\forall(\mathbf{x}, y), \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ \bar{\boldsymbol{w}} \end{pmatrix} + \eta\kappa(\mathbf{x}, y) \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \tag{3}$$

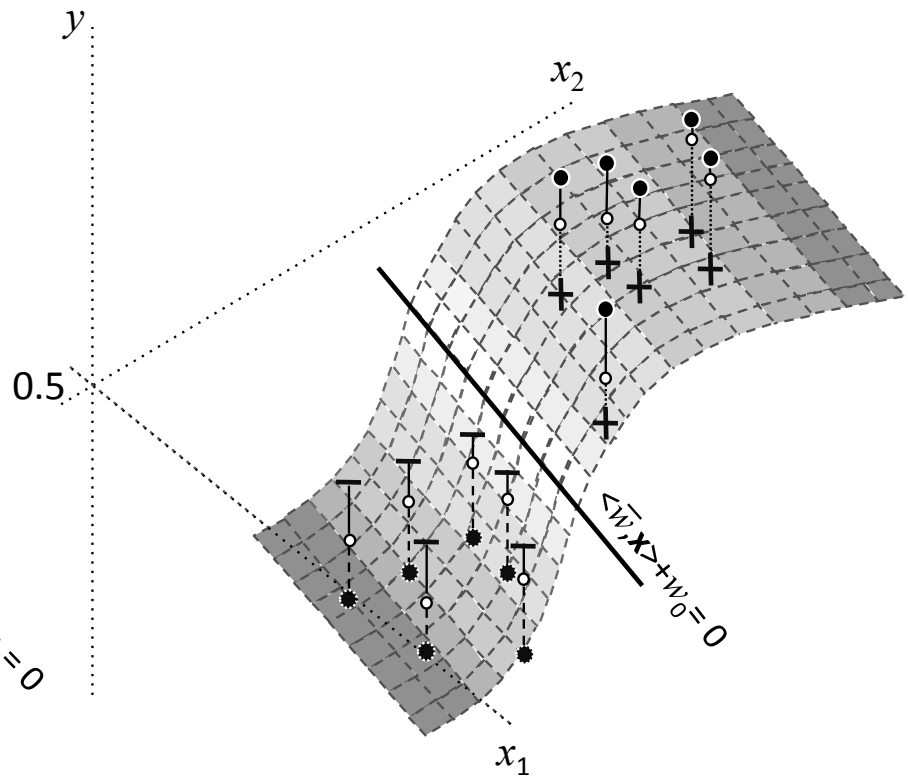
Model	Perceptron	Adaline	Logistic Regression
$\kappa(\mathbf{x}, y)$	$y1_{yh_{\boldsymbol{w}}(\mathbf{x})\leqslant 0}$	$(y - h_{\boldsymbol{w}}(\mathbf{x}))$	$y(1 - \sigma(h_{\boldsymbol{w}}(\mathbf{x})))$

Perceptron vs Adaline

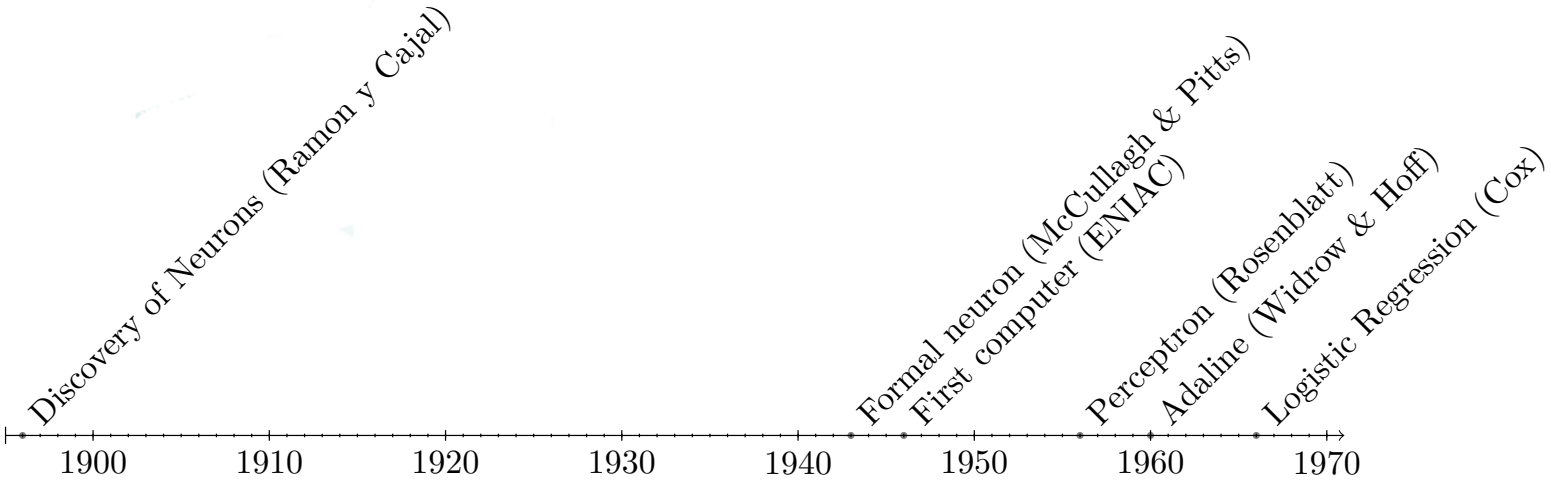


Adaline vs Logistic regression





Perceptron, Adaline & LR sparked excitement

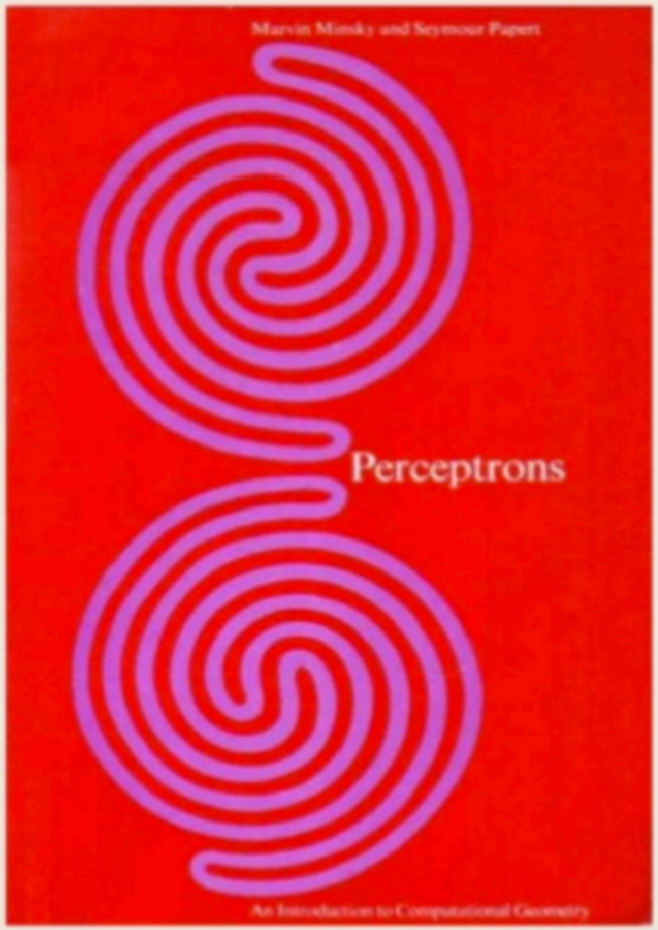


Perceptron, Adaline & LR sparked excitement but

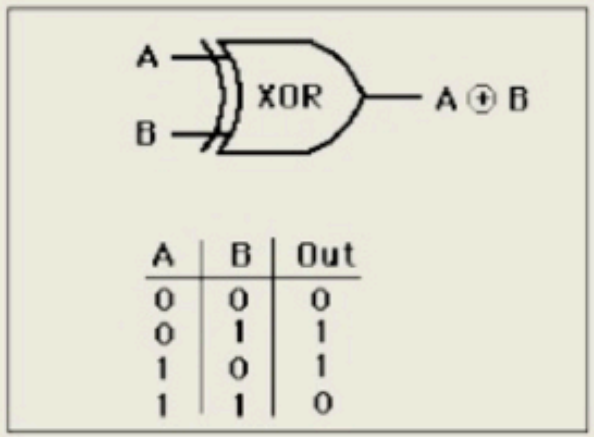
- Linearly separable problems are few
- Elementary learning problems need complex circuits (XOR, parity, etc.)
- Learning deep circuits means solving the credit assignment pb
- Circuit theory is poorly known

The 1st winter of NNs

1969: Perceptrons can't do XOR!



http://www.i-programmer.info/images/stories/BabBag/AI/book.jpg



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

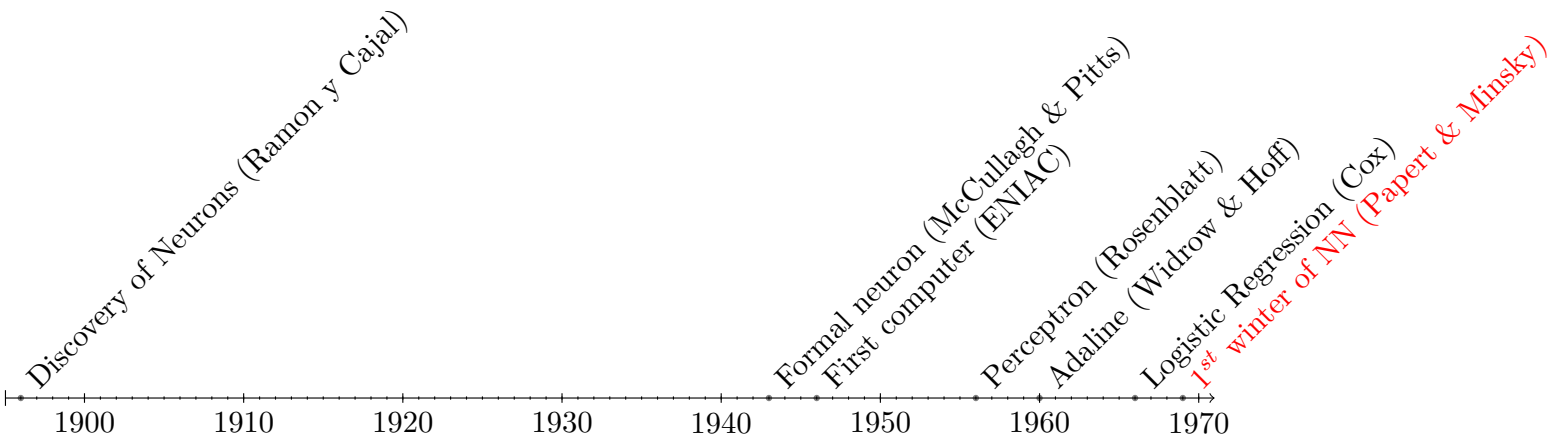
http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/ietron/xor.gif

Minsky & Papert

https://constructingkids.files.wordpress.com/2013/05/minsky-papert-71-csolomon-x640.jpg

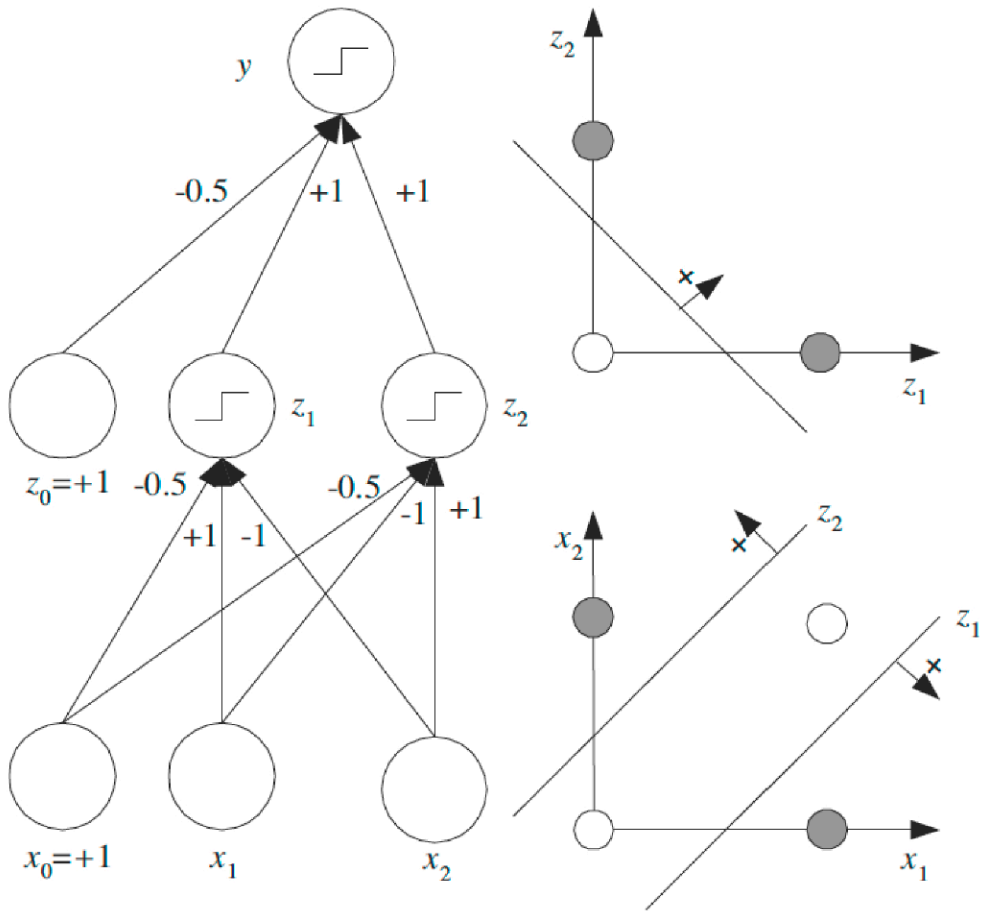
The 1st winter of NNs

- This marked the 1^{st} winter of NN;
  $\Rightarrow$ Abandon Perceptrons and other analog models,
  - Develop symbolic computers and Symbolic AI techniques and,
  - The search for non-linear models



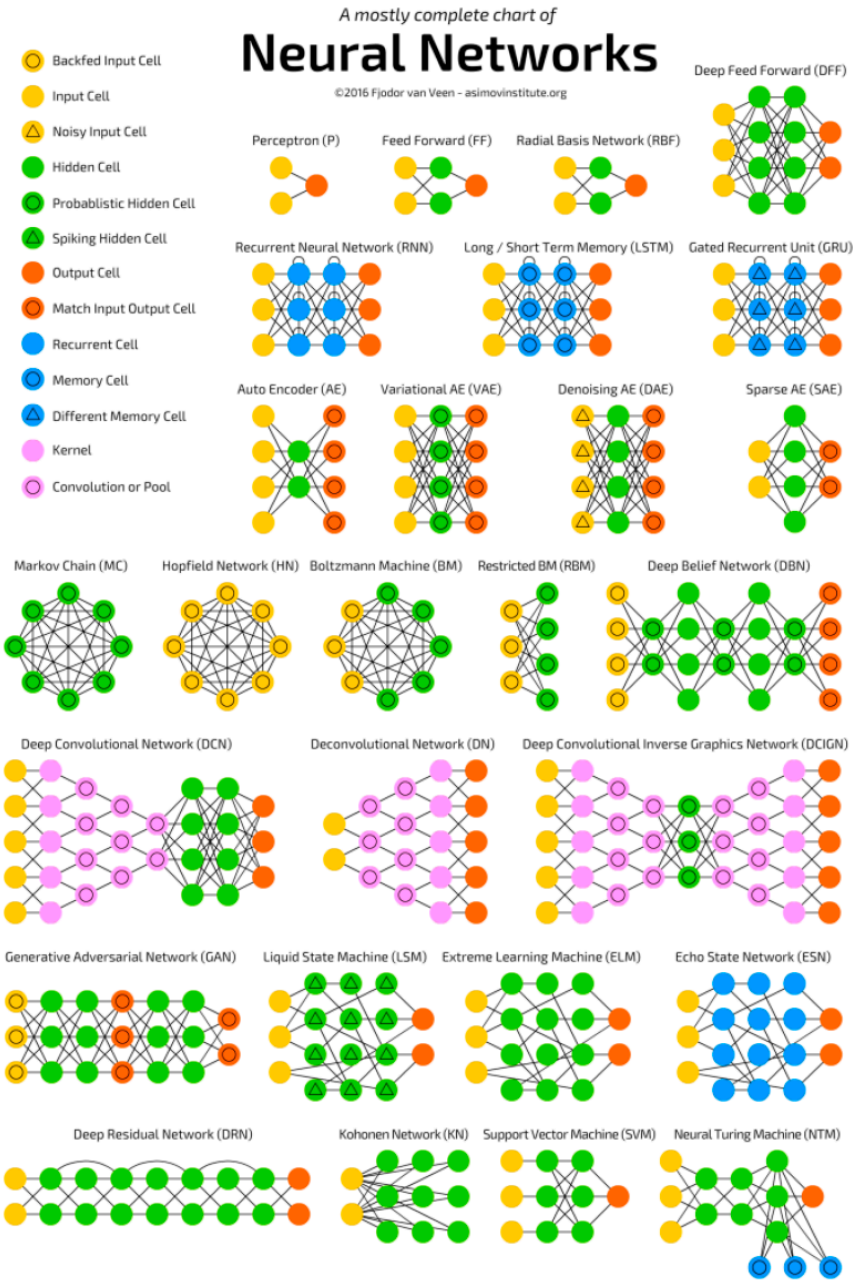
Neural Networks (1985)

- Solving non-linear problems by combining linear neurons: XOR problem



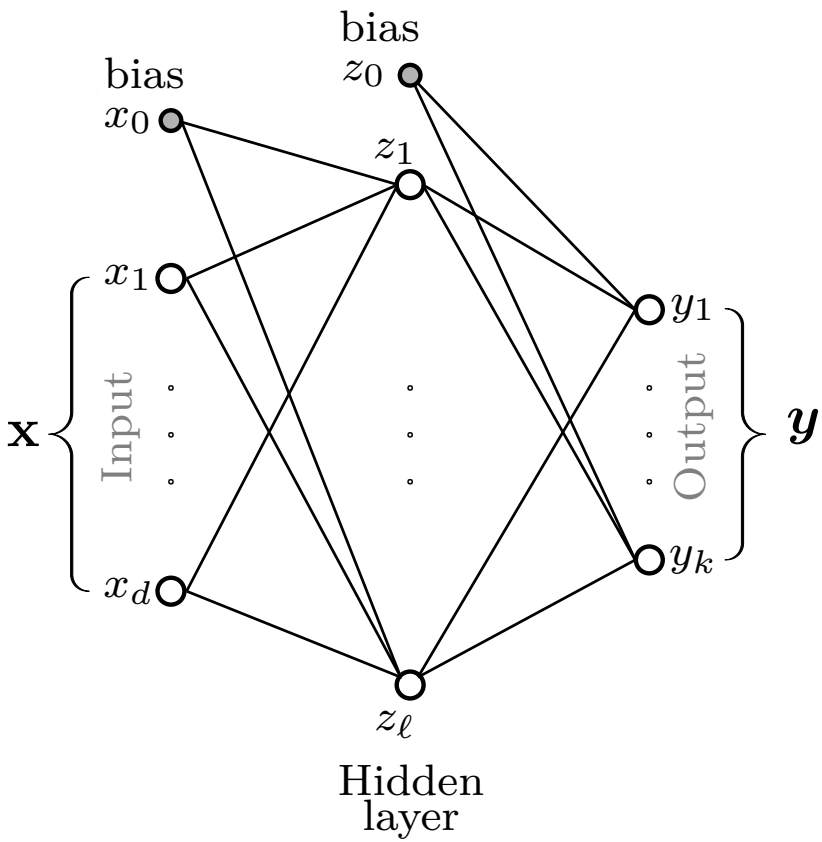
Neural Networks

- A Neural Network is an oriented weighted graph of formal neurons,
- When two neurons are connected (linked by an oriented edge of the graph), the output of the head neuron is used as an input by the tail neuron
- Three neurons are considered :
  - input neurons (connected with the input)
  - output neurons
  - hidden neurons


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

MultiLayer Perceptrons (MLP)

- Multi-Layer Perceptron is an acyclic Neural Network, where the neurons are structured in successive layers, beginning by an input layer and finishing with an output layer.



MLP

For the model above, the value of j^{th} unit of the hidden layer for an observation $\mathbf{x} = (x_i)_{i=1...d}$ in input is obtained by composition :

- of a dot product a_j, between $\mathbf{x}$ and the weight vector $\boldsymbol{w}_{j.}^{(1)} = (w_{ji}^{(1)})_{i=1,...,d}; j \in \{1, \ldots, \ell\}$ the features of $\mathbf{x}$ to this j^{th} unit and the parameters of the bias $w_{j0}^{(1)}$:

$$\forall j \in \{1, \ldots, \ell\}, a_j = \langle \boldsymbol{w}_{j.}^{(1)}, \mathbf{x} \rangle + w_{j0}^{(1)}$$
$$= \sum_{i=0}^{d} w_{ji}^{(1)} x_i$$

- and a bounded transfert function, $\bar{H}(.) : \mathbb{R} \to \mathbb{R}$:

$$\forall j \in \{1, \ldots, \ell\}, z_j = \bar{H}(a_j)$$

MLP

For the model above, the value of j^{th} unit of the hidden layer for an observation $\mathbf{x} = (x_i)_{i=1...d}$ in input is obtained by composition :

- The values of units of the output $(h_1, \ldots, h_K)$ is obtained in the same manner between the vector of the hidden layer $z_j, j \in \{0, \ldots, \ell\}$ and the weights linking this layer to the output $\boldsymbol{w}_{k.}^{(2)} = (w_{kj}^{(2)})_{j=1,\ldots,\ell}; k \in \{1, \ldots, K\}$,
- the predicted output for an observation $\mathbf{x}$ is a composite transformation of the input, which for the previous model is

$$\forall \mathbf{x}, \forall k \in \{1, \ldots, K\}, h(\mathbf{x}, k) = \bar{H}(a_k) = \bar{H}\left(\sum_{j=0}^{\ell} w_{kj}^{(2)} \times \bar{H}\left(\sum_{i=0}^{d} w_{ji}^{(1)} \times x_i\right)\right)$$

MLP

- An efficient way to estimate the parameters of NNs is the backpropagation algorithm [Rumelhart et al., 1986],
- For the mono-label classification case, an indicator vector is associated to each class

$$\forall (\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}, y = k \Leftrightarrow \mathbf{y}^\top = \left(\underbrace{y_1, \ldots, y_{k-1}}_{\text{all equal to } 0}, \underbrace{y_k}_{=1}, \underbrace{y_{k+1}, \ldots, y_K}_{\text{all equal to } 0} \right)$$

- After the phase of propagation of information for an example $(\mathbf{x}, y)$, an error is estimated between the prediction and the desired output, for example:

$$\forall (\mathbf{x}, \mathbf{y}), \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y}) = -\sum_{k=1}^{K} y_k \log h_k$$

MLP

- And the weights are corrected accordingly from the output to the input using the gradient descent algorithm

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y})}{\partial w_{ji}}$$

- Using the chain rule

$$\frac{\partial \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y})}{\partial w_{ji}} = \underbrace{\frac{\partial \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y})}{\partial a_j}}_{=\delta_j} \frac{\partial a_j}{\partial w_{ji}}$$

  where $\frac{\partial a_j}{\partial w_{ji}} = z_i$.

- In the case where, the unit j is on the output layer we have

$$\delta_j = \frac{\partial \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y})}{\partial a_j} = -y_j \frac{\bar{H}'(a_j)}{\bar{H}(a_j)}$$

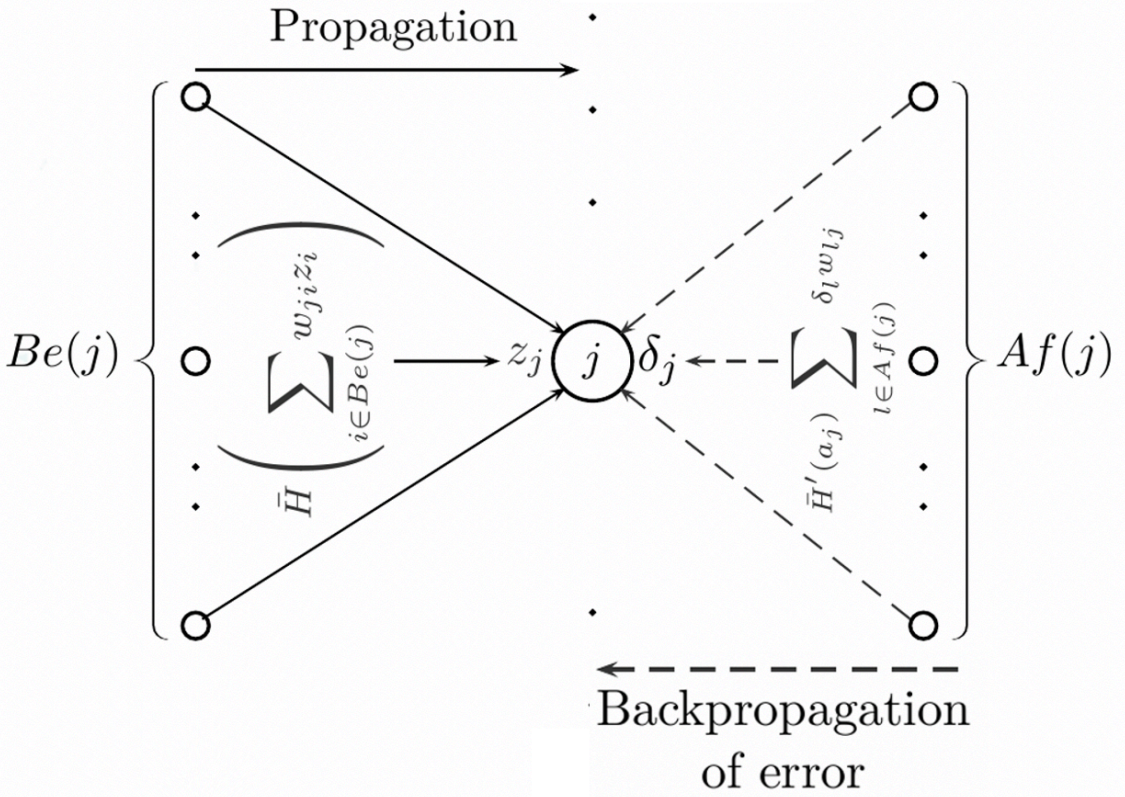
MLP

- If the unit j is on the hidden layer, we have by applying the chain rule again :

$$\delta_j = \frac{\partial \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y})}{\partial a_j} = \sum_{l \in Af(j)} \frac{\partial \ell(\boldsymbol{h}(\mathbf{x}), \mathbf{y})}{\partial a_l} \frac{\partial a_l}{a_j}$$
$$= \bar{H}'(a_j) \sum_{l \in Af(j)} \delta_l \times w_{lj}$$

where $Af(j)$ is the set of units that are on the layer which succeeds the one containing unit j.

Backpropagation [Hinton & Rumelhart, 1986]



MLP with L hidden layers - pseudocode

Algorithm 4 MLP

1: Training set $S = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, m\}\}$

2: Initialize the weights $\mathbf{W}^\ell = [w_{ji}^\ell]_{i,j}; \ell \in \{1, \ldots, L+1\}$

▷ Layer 0 is the input, Layer $L+1$ is the output

3: Learning rate $\eta > 0$

4: **repeat**

5: Choose randomly an example $(\mathbf{x}, \mathbf{y}) \in S$ ▷, $\mathbf{x} = \mathbf{z}^0$

▷ Propagation phase

$$\forall \ell \in \{1, \ldots, L+1\}, \forall j \in \{1, \ldots, d_\ell\}; a_j^\ell = \sum_{i=1}^{d_\ell - 1} w_{ji}^\ell z_i^{\ell-1} \text{ and } z_j^\ell = H(a_j^\ell)$$

▷ Retropropagation phase - estimation of δ

$$\forall j \in \{1, \ldots, K\}; \delta_j^{L+1} = -y_j \frac{H'(a_j^{L+1})}{H(a_j^{L+1})}$$

$$\forall \ell \in \{L, \ldots, 1\}, \forall j \in \{1, \ldots, d_\ell\}; \delta_j^\ell = H'(a_j^\ell) \sum_{s=1}^{d_{\ell+1}} \delta_s^{\ell+1} w_{sj}^{\ell+1}$$

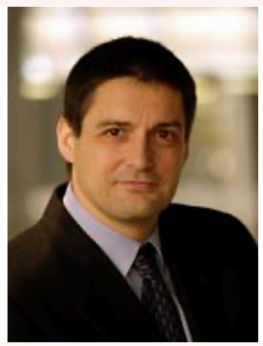
▷ Retropropagation phase - update of the weights

$$\forall \ell \in \{1, \ldots, L+1\}, \forall i \in \{1, \ldots, d_\ell\}, \forall j \in \{1, \ldots, d_\ell)\}; w_{ji}^\ell = w_{ji}^\ell - \eta \times \delta_j^\ell z_i^\ell$$

6: **until** $t > T$

MLP with a hidden layer is a universal approximator

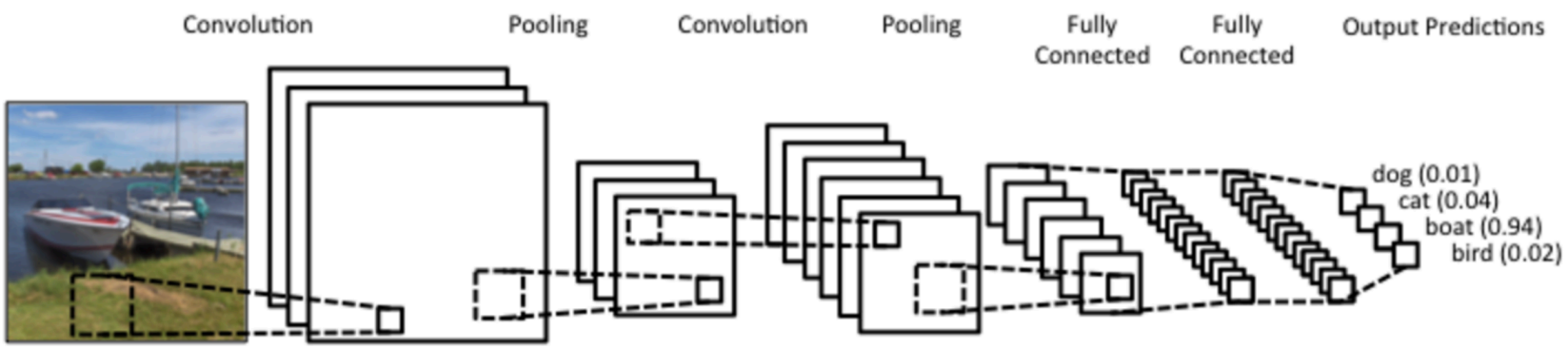
Theorem (Cybenko, Hornik & Funahashi)


For any continuous function f on compact subsets of $\mathbb{R}^d$. For any admissible activation non-polynomial function $\bar{H}$; there exists h; $W_1 \in \mathbb{R}^{d \times h}, b \in \mathbb{R}^h, c \in \mathbb{R}$ and $w_2 \in \mathbb{R}^h$ such that

$$\forall \epsilon > 0, \forall \mathbf{x} \in \mathbb{R}^d; \|f(\mathbf{x}) - w_2 \bar{H}(W_1 \mathbf{x} + b) + c\|_\infty \leq \epsilon$$

A Convolutional NN for image recognition [LeCun, 1997]



ADAptive BOOSTing [Schapire, 1999]

- The Adaboost algorithm generates a set of weak learners and combines them with a majority vote in order to produce an efficient final classifier.
- Each weak classifier is trained sequentially in the way to take into account the classification errors of the previous classifier
  - ☞ This is done by assigning weights to training examples and at each iteration to increase the weights of those on which the current classifier makes misclassification.
  - ☞ In this way the new classifier is focalized on *hard* examples that have been misclassified by the previous classifier.

AdaBoost, algorithm

Algorithm 5 The algorithm of Boosting

1: Training set $S = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, m\}\}$
2: Initialize the initial distribution over examples $\forall i \in \{1, \ldots, m\}, D_1(i) = \frac{1}{m}$
3: T, the maximum number of iterations (or classifiers to be combined)
4: **for each** t=1,...,T **do**
5: Train a weak classifier $f_t : \mathcal{X} \to \{-1, +1\}$ by using the distribution D_t
6: Set $\epsilon_t = \sum_{i: f_t(\mathbf{x}_i) \neq y_i} D_t(i)$
7: Choose $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
8: Update the distribution of weights

$$\forall i \in \{1, \ldots, m\}, D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i f_t(\mathbf{x}_i)}}{Z_t}$$

Where,

$$Z_t = \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i f_t(\mathbf{x}_i)}$$

9: **end for each**
10: The final classifier: $\forall \mathbf{x}, F(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(\mathbf{x})\right)$

source: http://ama.liglab.fr/~amini/RankBoost/

AdaBoost, algorithm

Algorithm 6 The algorithm of Boosting

Training set $S = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, m\}\}$

Initialize the initial distribution over examples $\forall i \in \{1, \ldots, m\}, D_1(i) = \frac{1}{m}$

T, the maximum number of iterations (or classifiers to be combined)

for each t=1,...,T **do**

Train a weak classifier $f_t : \mathcal{X} \to \{-1, +1\}$ by using the distribution D_t

Set $\epsilon_t = \sum_{i: f_t(\mathbf{x}_i) \neq y_i} D_t(i)$

Choose $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

Update the distribution of weights

$$\forall i \in \{1, \ldots, m\}, D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i f_t(\mathbf{x}_i)}}{Z_t}$$

Where,

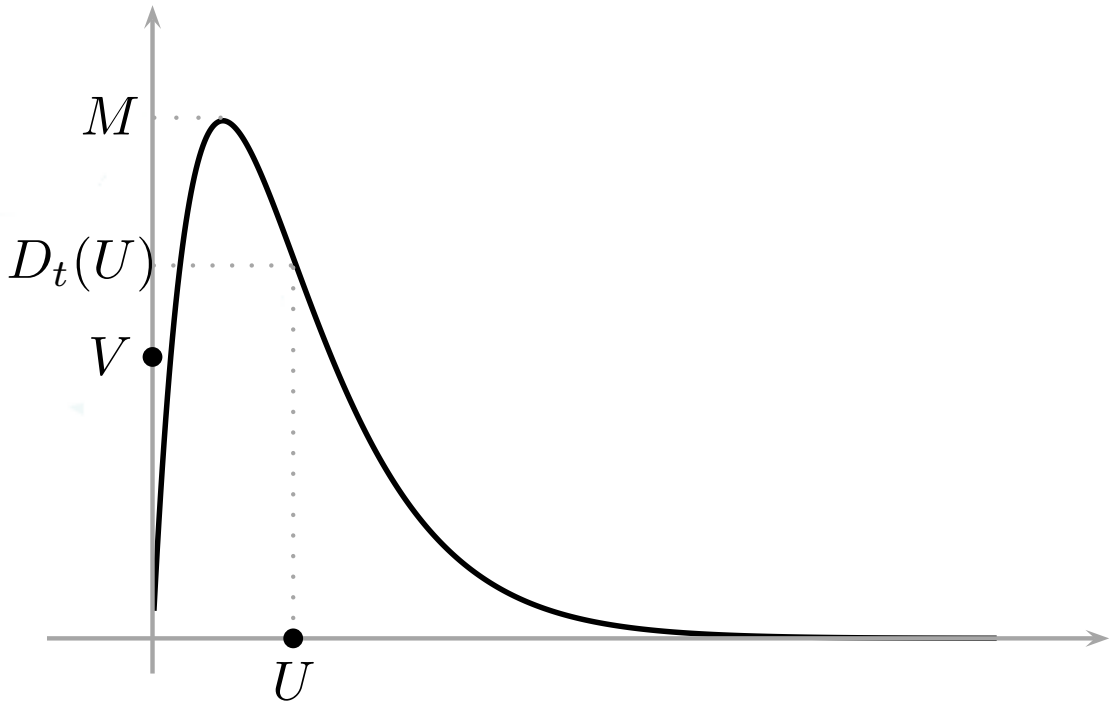
$$Z_t = \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i f_t(\mathbf{x}_i)}$$

end for each

The final classifier: $\forall \mathbf{x}, F(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(\mathbf{x})\right)$

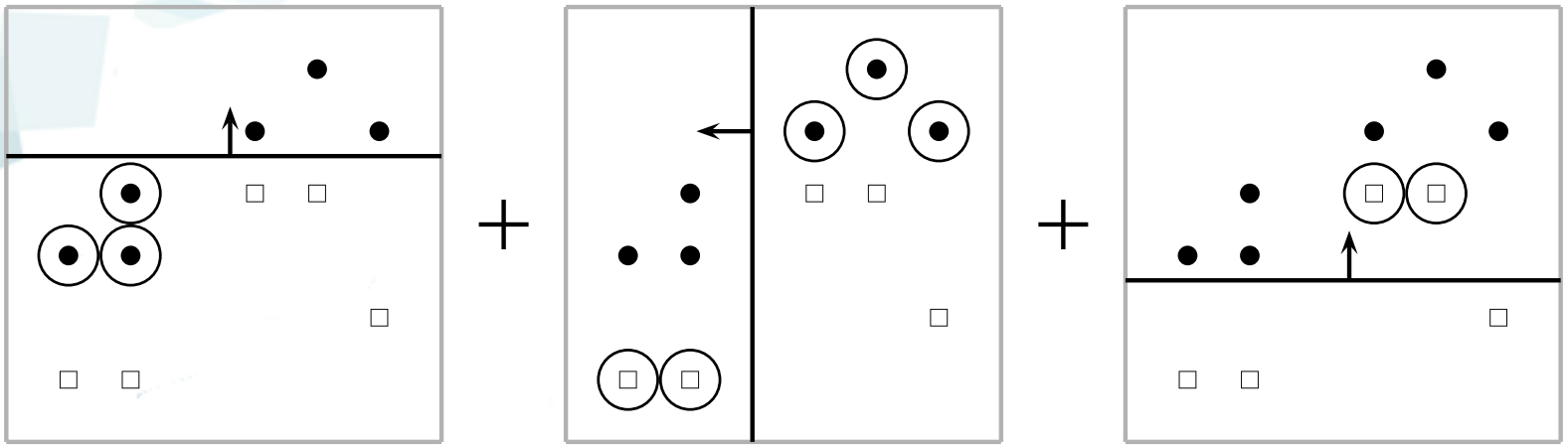
source: http://ama.liglab.fr/~amini/RankBoost/

How to sample using a distribution D_t

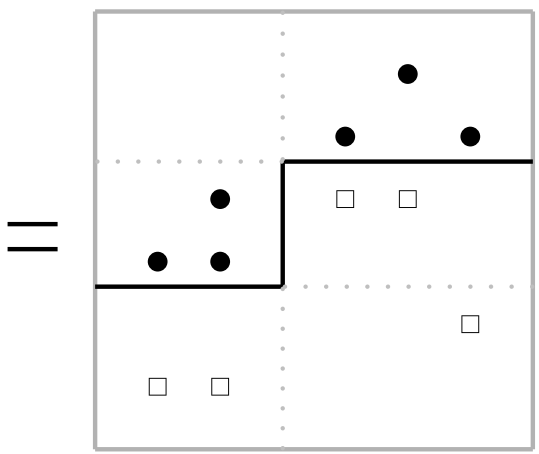


- Choose randomly an index $U \in \{1, \ldots, m\}$ and a real-value $V \in [0, \max_{i \in \{1,\ldots,m\}} D_t(i)]$, if $D_t(U) > V$ then accept the example $(\mathbf{x}_U, y_U)$.

AdaBoost, geometry interpretation



$\alpha_1 = 0.5$ $\alpha_2 = 0.1$ $\alpha_3 = 0.75$



Consistency of the ERM principle

Consistency of the ERM principle (1)

Suppose that the input dimension is $d = 1$, let the input space $\mathcal{X}$ be the interval $[a, b] \subset \mathbb{R}$ where a and b are real values such that $a < b$, and suppose that the output space is $\{-1, +1\}$. Moreover, suppose that the distribution $\mathcal{D}$ generating the examples $(\mathbf{x}, y)$ is an uniform distribution over $[a, b] \times \{-1\}$. Consider now, a learning algorithm which minimizes the empirical risk by choosing a function in the function class $\mathcal{F} = \{f : [a, b] \to \{-1, +1\}\}$ (also denoted as $\mathcal{F} = \{-1, +1\}^{[a,b]}$) in the following way ; after reviewing a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ the algorithm outputs the prediction function f_S such that

$$f_S(\mathbf{x}) = \begin{cases} -1, & \text{if } \mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \\ +1, & \text{otherwise} \end{cases}$$

Consistency of the ERM principle (2)

- For the above problem, the found classifier has an empirical risk equal to 0, and that for any given training set. However, as the classifier makes an error over the entire infinite set $[a, b]$ except on a finite training set (of measure zero), its generalization error is always equal to 1.

- So the question is : *in which case the ERM principle is likely to generate a general learning rule?*
  $\Rightarrow$ The answer of this question lies in a statistical notion called consistency.

Consistency of the ERM principle (3)

This concept indicates two conditions that a learning algorithm has to fulfill, namely

(a) the algorithm must return a prediction function whose empirical error reflects its generalization error when the size of the training set tends to infinity :
$\forall \epsilon > 0, \lim_{m \to \infty} \mathbb{P}(|\hat{\mathcal{L}}_m(f_S, S) - \mathcal{L}(f_S)| > \epsilon) = 0$, denoted as,

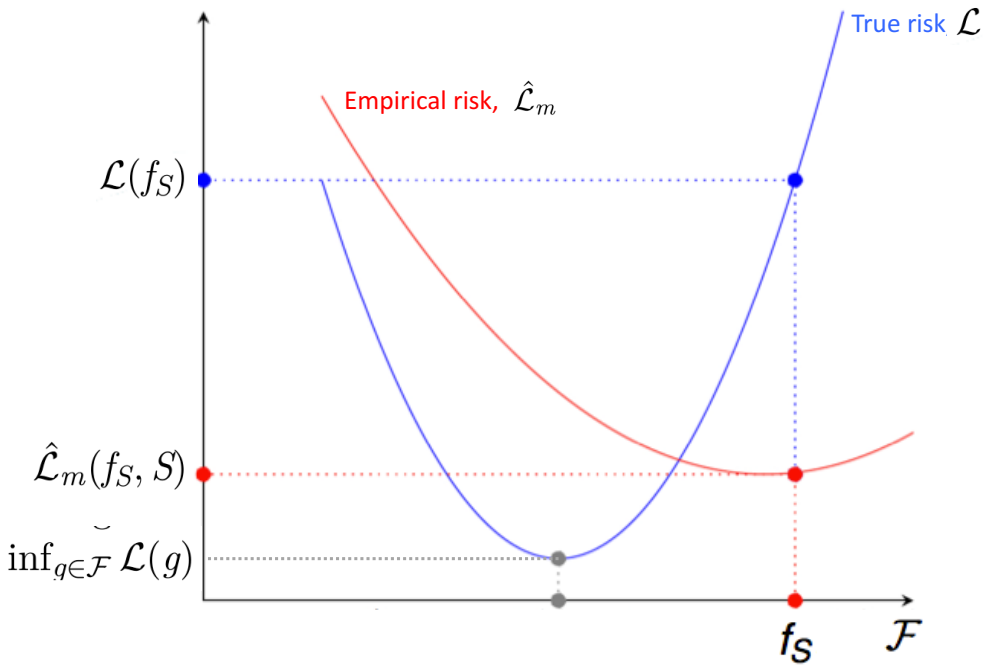
$$\hat{\mathcal{L}}_m(f_S, S) \xrightarrow{\mathbb{P}} \mathcal{L}(f_S)$$

(b) in the asymptotic case, the algorithm must allow to find the function which minimizes the generalization error in the considered function class :

$$\hat{\mathcal{L}}_m(f_S, S) \xrightarrow{\mathbb{P}} \inf_{g \in \mathcal{F}} \mathcal{L}(g)$$

Consistency of the ERM principle (4)

These two conditions imply that the empirical error $\hat{\mathcal{L}}_m(f_S, S)$ of the prediction function found by the learning algorithm over a training S, f_S, converges in probability to its generalization error $\mathcal{L}(f_S)$ and $\inf_{g \in \mathcal{F}} \mathcal{L}(g)$:



Consistency of the ERM principle: A worse case study

The fundamental result of the learning theory [Vapnik 88, theorem 2.1, p.38] concerning the consistency of the ERM principle, exhibits another relation involving the supremum over the function class in the form of an unilateral uniform convergence and which stipulates that :

The ERM principle is consistent if and only if :

$$\forall \epsilon > 0, \lim_{m \to \infty} \mathbb{P}\left(\sup_{f \in \mathcal{F}} \left[\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S) \right] > \epsilon \right) = 0$$

A uniform generalization error bound (1)

1. Link the supremum of $\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)$ on $\mathcal{F}$ with its expectation

consider the following function

$$\Phi : S \mapsto \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$$

Let $(\mathbf{x}', y')$ an example generated i.i.d. from the same probability distribution $\mathcal{D}$ that generated S; and let $\forall i \in \{1, \ldots, m\} S^i = S \setminus \{(\mathbf{x}_i, y_i)\} \cup \{(\mathbf{x}', y')\}$ then we have

$$\forall i \in \{1, \ldots, m\}; |\Phi(S) - \Phi(S^i)| \leq \frac{1}{m}$$

A uniform generalization error bound (1)

Theorem ([McDiarmid 89])

Let $I \subset \mathbb{R}$ be a real valued interval, and $(X_1, ..., X_m)$, m independent random variables taking values in I^m. Let $\Phi : I^m \to \mathbb{R}$ be defined such that : $\forall i \in \{1, ..., m\}, \exists c_i \in \mathbb{R}$ the following inequality holds for any $(\mathbf{x}_1, ..., \mathbf{x}_m) \in I^m$ and $\forall \mathbf{x}' \in I$:

$$|\Phi(\mathbf{x}_1, .., \mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, .., \mathbf{x}_m) - \Phi(\mathbf{x}_1, .., \mathbf{x}_{i-1}, \mathbf{x}', \mathbf{x}_{i+1}, .., \mathbf{x}_m)| \leq c_i$$

We have then

$$\forall \epsilon > 0, \mathbb{P}(\Phi(\mathbf{x}_1, ..., \mathbf{x}_m) - \mathbb{E}[\Phi] > \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^{m} c_i^2}}$$

A uniform generalization error bound (1)

1. Link the supremum of $\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)$ on $\mathcal{F}$ with its expectation

consider the following function

$$\Phi : S \mapsto \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$$

Mcdiarmid inequality can then be applied for the function Φ with $c_i = 1/m, \forall i$, thus :

$$\forall \epsilon > 0, \mathbb{P}\left(\sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)] - \mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)] > \epsilon \right) \leq e^{-2m\epsilon^2}$$

A uniform generalization error bound (2)

2. Bound $\mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$ with respect to $\mathfrak{R}_m(\ell \circ \mathcal{F})$

This step is a symmetrisation step and it consists in introducing a second virtual sample S' also generated i.i.d. with respect to $\mathcal{D}^m$ into $\mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$.

$$\begin{aligned} \rightarrow \mathbb{E}_S \sup_{f \in \mathcal{F}} (\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)) &= \mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathbb{E}_{S'} (\hat{\mathcal{L}}_m(f, S') - \hat{\mathcal{L}}_m(f, S))] \\ &\leq \mathbb{E}_S \mathbb{E}_{S'} \sup_{f \in \mathcal{F}} [\hat{\mathcal{L}}_m(f, S') - \hat{\mathcal{L}}_m(f, S)] \end{aligned}$$

$\rightarrow$ In the other hand,

$$\mathbb{E}_S \mathbb{E}_{S'} \sup_{f \in \mathcal{F}} [\hat{\mathcal{L}}_m(f, S') - \hat{\mathcal{L}}_m(f, S)]$$

$$= \mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\boldsymbol{\sigma}} \sup_{f \in \mathcal{F}} \left[\frac{1}{m} \sum_{i=1}^{m} \sigma_i (\ell(f(\mathbf{x}'_i), y'_i) - \ell(f(\mathbf{x}_i), y_i)) \right]$$

A uniform generalization error bound (2)

2. Bound $\mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$

By applying the triangular inequality $\sup = ||.||_\infty$ it comes

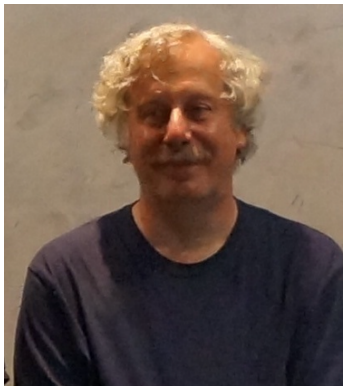
$$\mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\boldsymbol{\sigma}} \sup_{f \in \mathcal{F}} \left[\frac{1}{m} \sum_{i=1}^{m} \sigma_i (\ell(f(\mathbf{x}'_i), y'_i) - \ell(f(\mathbf{x}_i), y_i)) \right] \leq$$

$$\mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\sigma} \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(f(\mathbf{x}'_i), y'_i) + \mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\boldsymbol{\sigma}} \quad \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} (-\sigma_i) \ell(f(\mathbf{x}'_i), y'_i)$$

Finally as $\forall i, \sigma_i$ and $-\sigma_i$ have the same distribution we have

$$\mathbb{E}_S \mathbb{E}_{S'} \sup_{f \in \mathcal{F}} [\hat{\mathcal{L}}_m(f, S') - \hat{\mathcal{L}}_m(f, S)] \leq \underbrace{2 \mathbb{E}_S \mathbb{E}_{\boldsymbol{\sigma}} \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(f(\mathbf{x}_i), y_i)}_{\text{Relates to the complexity of } \mathcal{F}} \qquad (4)$$

Rademacher complexity [Koltchinskii 01]



- In the derivation of uniform generalization error bounds different capacity measures of the class of functions have been proposed. Among which the Rademacher complexity allows an accurate estimates of the capacity of a class of functions and it is dependent to the training sample
- The empirical Rademacher complexity estimates the richness of a function class $\mathcal{F}$ by measuring the degree to which the latter is able fit to random noise on a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ of size m generated i.i.d. with respect to a probability distribution $\mathcal{D}$.

Rademacher complexity [Koltchinskii 01]

- This complexity is estimated through Rademacher variables $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)^\top$ which are independent discrete random variables taking values in $\{-1, +1\}$ with the same probability $1/2$, i.e.
  $\forall i \in \{1, \ldots, m\}; \mathbb{P}(\sigma_i = -1) = \mathbb{P}(\sigma_i = +1) = 1/2$, and is defined as :

$$\hat{\mathfrak{R}}_m(\mathcal{F}, S) = \frac{2}{m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f \in \mathcal{F}} \left| \sum_{i=1}^{m} \sigma_i f(\mathbf{x}_i) \right| \mid \mathbf{x}_1, \ldots, \mathbf{x}_m \right]$$

- Furthermore, we define the Rademacher complexity of the class of functions $\mathcal{F}$ independently to a given training set by

$$\mathfrak{R}_m(\mathcal{F}) = \mathbb{E}_{S \sim \mathcal{D}^m} \hat{\mathfrak{R}}_m(\mathcal{F}, S) = \frac{2}{m} \mathbb{E}_{S\boldsymbol{\sigma}} \left[\sup_{f \in \mathcal{F}} \left| \sum_{i=1}^{m} \sigma_i f(\mathbf{x}_i) \right| \right]$$

A uniform generalization error bound (2)

2. Bound $\mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$

By applying the triangular inequality $\sup = ||.||_\infty$ it comes

$$\mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\boldsymbol{\sigma}} \sup_{f \in \mathcal{F}} \left[\frac{1}{m} \sum_{i=1}^{m} \sigma_i (\ell(f(\mathbf{x}'_i), y'_i) - \ell(f(\mathbf{x}_i), y_i)) \right] \leq$$

$$\mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\sigma} \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(f(\mathbf{x}'_i), y'_i) + \mathbb{E}_S \mathbb{E}_{S'} \mathbb{E}_{\boldsymbol{\sigma}} \quad \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} (-\sigma_i) \ell(f(\mathbf{x}'_i), y'_i)$$

Finally as $\forall i, \sigma_i$ and $-\sigma_i$ have the same distribution we have

$$\mathbb{E}_S \mathbb{E}_{S'} \sup_{f \in \mathcal{F}} [\hat{\mathcal{L}}_m(f, S') - \hat{\mathcal{L}}_m(f, S)] \leq \underbrace{2 \mathbb{E}_S \mathbb{E}_{\boldsymbol{\sigma}} \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(f(\mathbf{x}_i), y_i)}_{\leq \mathfrak{R}_m(\ell \circ \mathcal{F})} \qquad (5)$$

A uniform generalization error bound (2)

2. Bound $\mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$ with respect to $\mathfrak{R}_m(\ell \circ \mathcal{F})$

In summarizing the results obtained so far, we have:

1. $\forall f \in \mathcal{F}, \forall S, \mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S) \leq \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)]$
2. $\forall \epsilon > 0, \mathbb{P}\left(\sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)] - \mathbb{E}_S \sup_{f \in \mathcal{F}} [\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)] > \epsilon \right) \leq e^{-2m\epsilon^2}$
3. $\mathbb{E}_S \sup_{f \in \mathcal{F}} (\mathcal{L}(f) - \hat{\mathcal{L}}_m(f, S)) \leq \mathfrak{R}_m(\ell \circ \mathcal{F})$

By resolving the equation $e^{-2m\epsilon^2} = \delta$ with respect to ϵ we hance get the following Theorem.

A uniform generalization error bound

Theorem (Rademacher Generalization bounds)

Let $\mathcal{X} \in \mathbb{R}^d$ be a vectorial space and $\mathcal{Y} = \{-1, +1\}$ an output space. Suppose that the pairs of examples $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ are generated i.i.d. with respect to the distribution probability $\mathcal{D}$. Let $\mathcal{F}$ be a class of functions having values in $\mathcal{Y}$ and $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$ a given instantaneous loss. Then for all $\delta \in]0, 1]$, we have with probability at least $1 - \delta$ the following inequality :

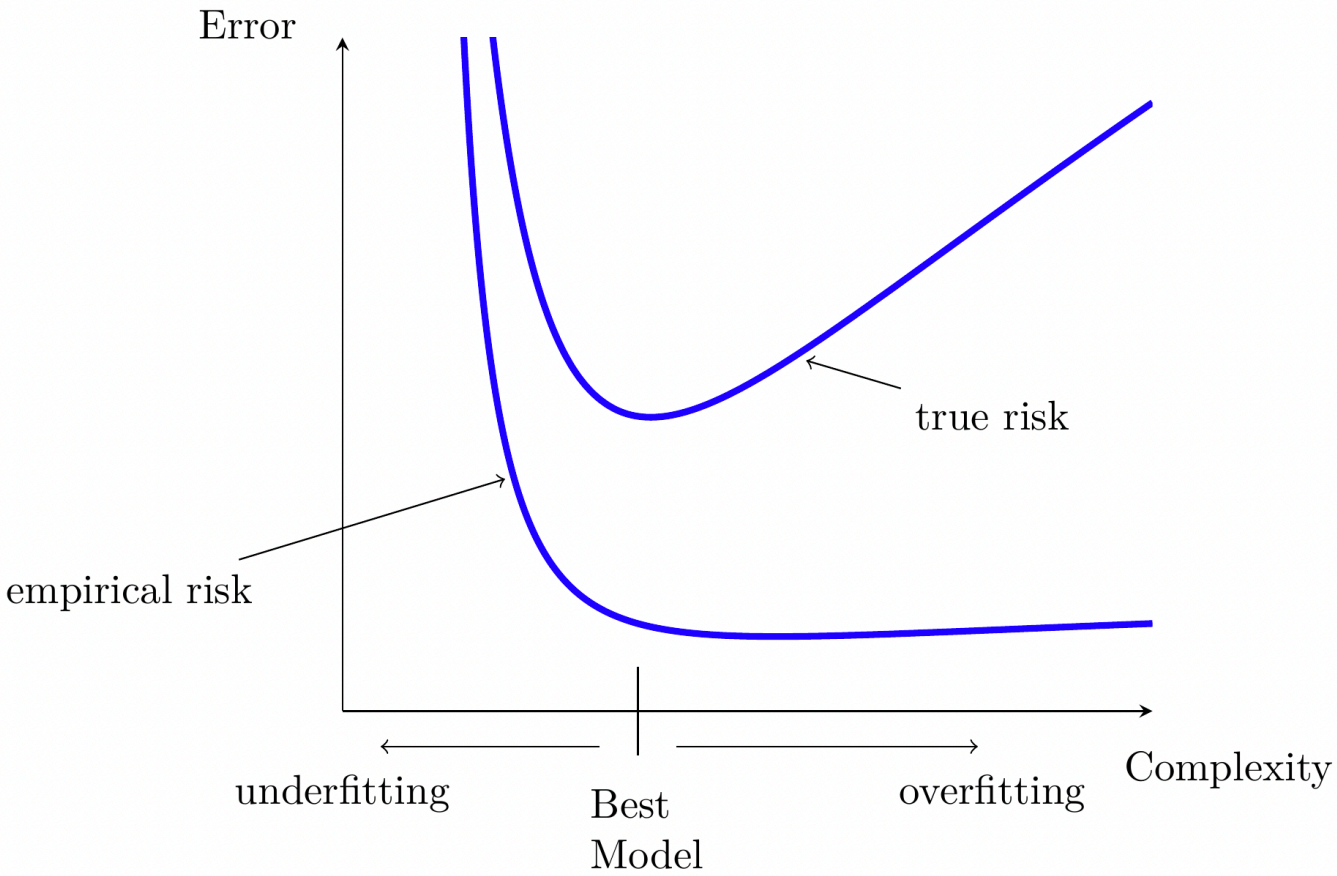
$$\forall f \in \mathcal{F}, \mathcal{L}(f) \leq \hat{\mathcal{L}}_m(f, S) + \mathfrak{R}_m(\ell \circ \mathcal{F}) + \sqrt{\frac{\ln \frac{1}{\delta}}{2m}} \tag{6}$$

Using the same steps we can also show that with probability at least $1 - \delta$

$$\mathcal{L}(f) \leq \hat{\mathcal{L}}_m(f, S) + \hat{\mathfrak{R}}_m(\ell \circ \mathcal{F}, S) + 3\sqrt{\frac{\ln \frac{2}{\delta}}{2m}} \tag{7}$$

Where $\ell \circ \mathcal{F} = \{(\mathbf{x}, y) \mapsto \ell(f(\mathbf{x}), y) \mid f \in \mathcal{F}\}$.

Structural Risk Minimization



Learning is a compromise between low empirical error and high complexity of the class of functions in use.

Regularization for SRM

- Find a predictor by minimising the empirical risk with an added penalty for the size of the model,
- A simple approach consists in choosing a large class of functions $\mathcal{F}$ and to define on $\mathcal{F}$ a *regularizer*, typically a norm $|| \, g \, ||$, then to minimize the regularized empirical risk

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \, \hat{\mathcal{L}}_m(f, S) + \underbrace{\gamma}_{hyperparameter} \times || \, f \, ||^2$$

- The hyper parameter, or the *regularisation parameter* allows to choose the right trade-off between fit and complexity.

Estimating hyperparameters with cross validation

- Create a K-fold partition of the dataset
  - For each of K experiments, use $K-1$ folds for training and a different fold for testing, this procedure is illustrated in the following figure for $K=4$

- The value of the hyper parameter corresponds to the value of γ_k for which the testing performance is the highest on one of the folds.

Support Vector Machines [Boser et al. 92]

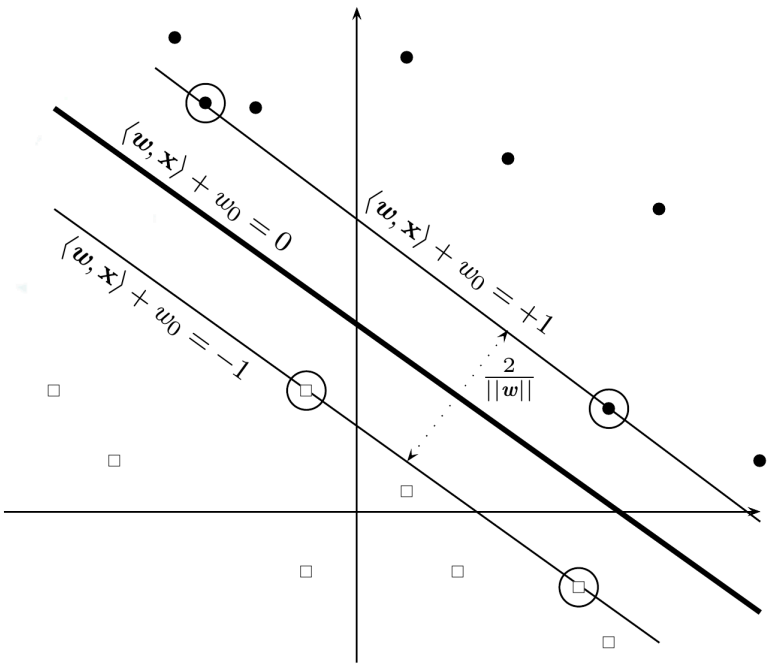
- Developed from Statistical Learning Theory mainly due to (Chervonenski & Vapnik).

- Many researchers from different communities (ML, Optimization, Statistics, etc.) work on them

"Nothing is more practical than a good theory."
V. Vapnik

$\Rightarrow$ A new way of developing learning models.

Support Vector Machines [Boser et al. 92]



Linearly separable case, learning objective:

$$\max_{\boldsymbol{w}} \frac{2}{\|\boldsymbol{w}\|}$$

s.t.

$$\langle \bar{\boldsymbol{w}}, \mathbf{x} \rangle + w_0 \geq \quad 1, \qquad \forall \mathbf{x} \text{ of class } \bullet$$
$$\langle \bar{\boldsymbol{w}}, \mathbf{x} \rangle + w_0 \leq \quad -1, \qquad \forall \mathbf{x} \text{ of class } \square$$

SVM (linearly separable case)

- The objective is equivalent to:

$$\min_{\boldsymbol{w}} \frac{\|\boldsymbol{w}\|^2}{2} \quad \text{s.t. } y_i(\langle \boldsymbol{w}, \mathbf{x}_i \rangle + w_0) \geq 1, \forall \mathbf{x}_i \tag{8}$$

- Which is equivalent to the minimization of the following objective function

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} \lfloor 1 - y_i(\langle \boldsymbol{w}, \mathbf{x}_i \rangle + w_0) \rfloor_+ + \lambda \frac{\|\boldsymbol{w}\|^2}{2}$$

where $\lfloor x \rfloor_+ = x$ if $x > 0$ and 0 otherwise. $\lfloor 1 - y_i h_{\boldsymbol{w}}(\mathbf{x}_i) \rfloor_+$, is called the Hinge loss.

Solving SVMs

- Different approaches have been proposed for solving the minimization problem, among which *Primal Estimated sub-Gradient SOlver for SVM* [Shalev, 2011] is one of the principle ones.

Algorithm 7 PEGaSOS

1: **Input**: Training set $S = (\mathbf{x}_i, y_i)_{1 \leq i \leq m}$, constant $\lambda > 0$ and maximum number of iterations T
2: **Initialize**: Set $\boldsymbol{w}^{(1)} \leftarrow 0$
3: **for each** $t = 1, 2, ..., T$ **do**
4: Set $S_t^+ = \{(\mathbf{x}, y) \in S; y\langle \boldsymbol{w}^{(t)}, \mathbf{x} \rangle < 1\}$
5: Set $\eta_t = \frac{1}{\lambda t}$
6: Update $\boldsymbol{w}^{(t+1)} \leftarrow (1 - \lambda \eta_t)\boldsymbol{w}^{(t)} + \frac{\eta_t}{m} \sum_{(\mathbf{x},y) \in S_t^+} y\mathbf{x}$
7: **end for each**
8: **Output**: $\boldsymbol{w}^{(T+1)}$

SVM (linearly separable case)

- The resolution of (8) can also be done through the Lagrangian:

$$L_p(w) = \frac{\|\boldsymbol{w}\|^2}{2} + \sum_{i=1}^{m} \alpha_i (1 - y_i(\langle \boldsymbol{w}, \mathbf{x}_i \rangle + w_0))$$

- The solution $\boldsymbol{w}^*$ of the optimization problem should verify are called Karush–Kuhn–Tucker conditions which are obtained by setting to zero the gradient of the Lagrangian and by writing the complementary conditions :

$$\nabla L_p(\bar{\boldsymbol{w}}^*) = \bar{\boldsymbol{w}}^* - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0, \Leftrightarrow \quad \bar{\boldsymbol{w}}^* = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{9}$$

$$\nabla L_p(w_0^*) = -\sum_{i=1}^{m} \alpha_i y_i = 0, \ \Leftrightarrow \ \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{10}$$

$$\forall i, \alpha_i \left[y_i(\langle \bar{\boldsymbol{w}}^*, \mathbf{x}_i \rangle + w_0^*) - 1 \right] = 0, \ \text{so} \ \begin{cases} \alpha_i = 0, \text{ or} \\ y_i(\langle \bar{\boldsymbol{w}}^*, \mathbf{x}_i \rangle + w_0^*) = 1 \end{cases} \tag{11}$$

SVM (linearly separable case)

By plugging back (9) et (10) to the primal objectif we get :

$$L_p = \frac{1}{2}\left\|\sum_{i=1}^{m}\alpha_i y_i \mathbf{x}_i\right\|^2 - \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle - w_0^* \underbrace{\sum_{i=1}^{m}\alpha_i y_i}_{=0} + \sum_{i=1}^{m}\alpha_i$$

$$= -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle + \sum_{i=1}^{m}\alpha_i$$

we obtain the dual optimization problem known as dual of *Wolfe* :

$$\max_{(\alpha_1,\ldots,\alpha_m)\in\mathbb{R}^m} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle \qquad (12)$$

$$\text{u.c.} \quad \sum_{i=1}^{m} y_i \alpha_i = 0 \text{ and } \forall i, \alpha_i \geq 0 \qquad (13)$$

SVM, the kernel trick

- The resolution of the dual optimization problem requires the knowing of the similarity between training examples $\forall i, j; \langle \mathbf{x}_i, \mathbf{x}_j \rangle$.
- The similarity measured by dot product can also be expressed by a kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ which may depend on a mapping of $\phi : \mathcal{X} \mapsto \mathcal{X}^\phi$ that transforms the input space into a higher-dimensional space, called the *feature space*; i.e.

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_\phi$$

  In general κ is a real-valued positive definite function, whose expression is fixed. For example:

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^\top \mathbf{x}' + c)^n, c \in \mathbb{R}_+^*; n \in \mathbb{N}^* && \text{polynomial kernel} \\ \kappa(\mathbf{x}, \mathbf{x}') &= \exp(-\tfrac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}) && \text{Gaussian radial basis function} \end{aligned}$$

- The substitution of dot product by a kernel function is referred to as the *kernel trick*.

Algorithm 8 SVM Hard Margin

1: Training set $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$

2: Find $\boldsymbol{\alpha}^*$, solution of the optimization problem

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{u.c.} \sum_{i=1}^{m} y_i \alpha_i = 0 \text{ and } \forall i, \alpha_i \geq 0$$

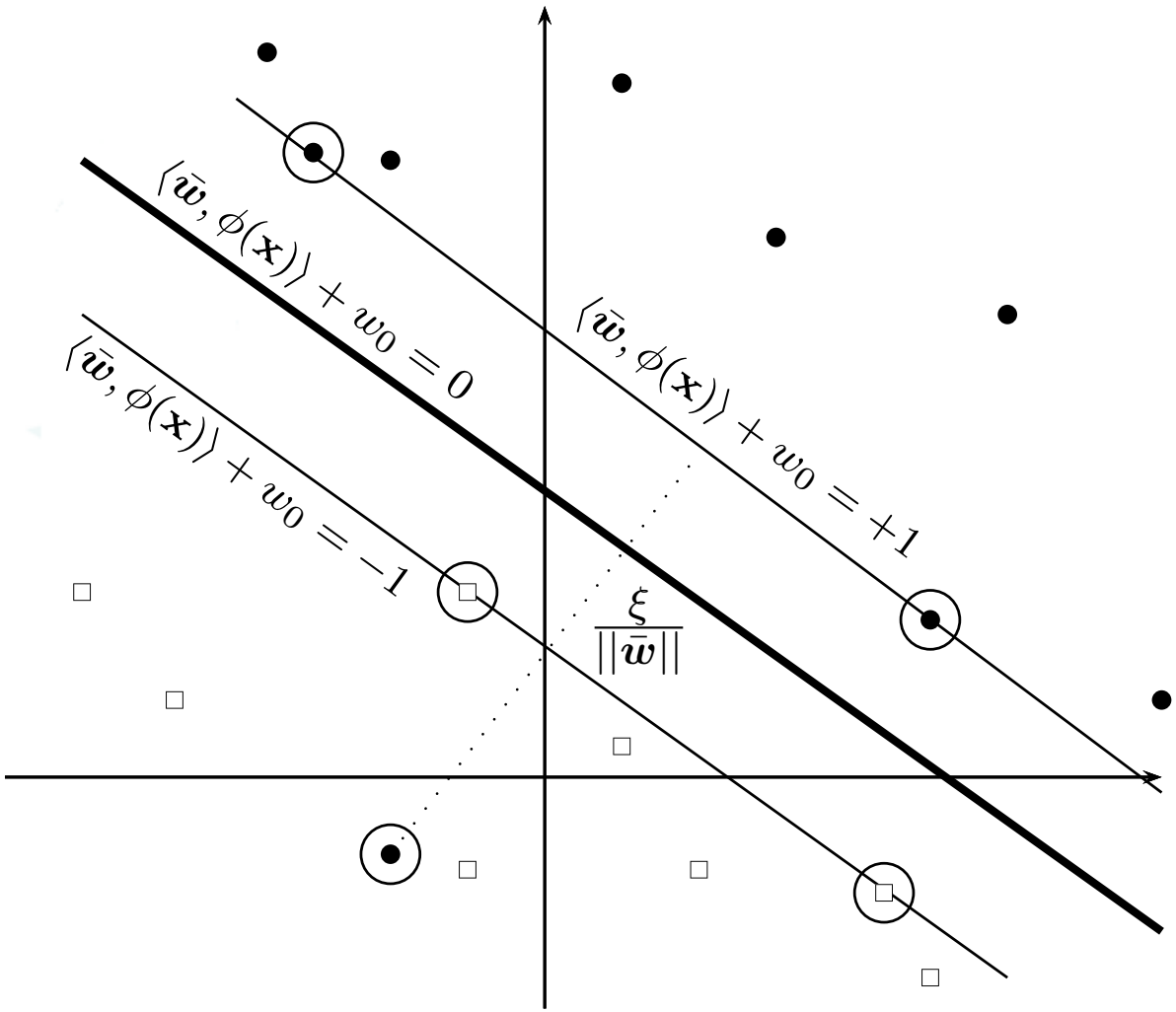
3: Choose $i \in \{1, \ldots, m\}$ such that $\alpha_i^* > 0$

4: Let $w_0^* = y_i - \sum_{j=1}^{m} \alpha_j^* y_j \kappa(\mathbf{x}_j, \mathbf{x}_i)$

5: $\bar{\boldsymbol{w}}^* = \sum_{i=1}^{m} y_i \alpha_i^* \phi(\mathbf{x}_i)$

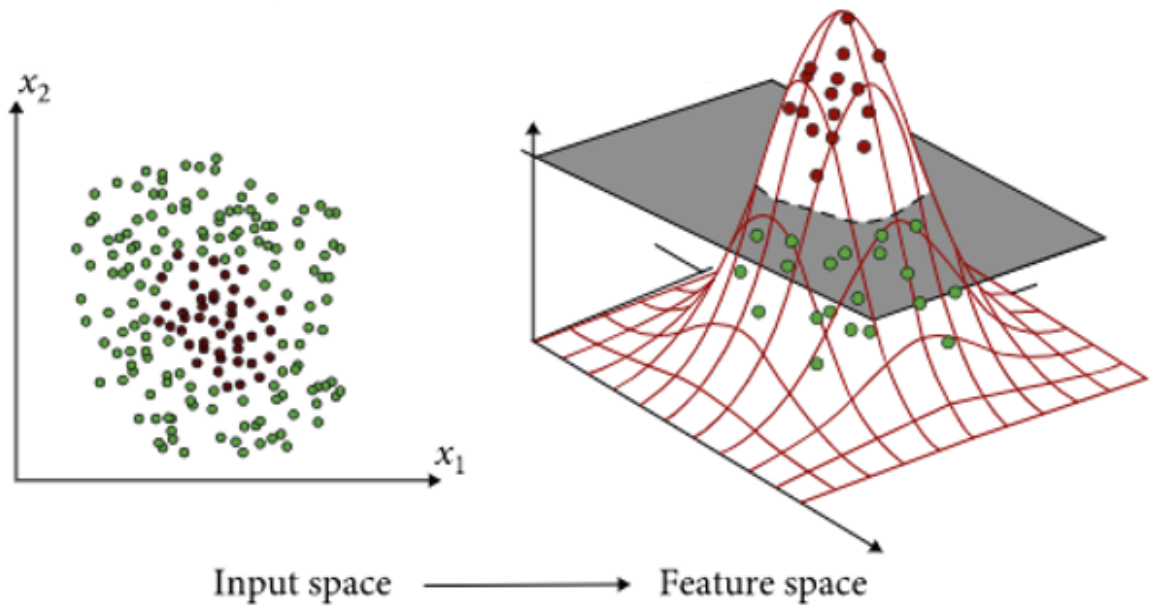
6: Output; $\forall \mathbf{x}, f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i^* \kappa(\mathbf{x}_i, \mathbf{x}) + w_0^* \right)$

SVM (non-linearly separable case)



SVM (non-linearly separable case)

- The idea is that by mapping the training data using ϕ is that training examples will become linearly separable in the higher-dimensional feature space.



SVM (non-linearly separable case)

- In this case, the optimization problem in the feature space writes

$$\begin{aligned} \forall i, & \quad \xi_i \geq 0 \\ \forall \mathbf{x}_i \in S_+, & \quad \langle \bar{\boldsymbol{w}}, \phi(\mathbf{x}_i) \rangle + w_0 \geq 1 - \xi_i \\ \forall \mathbf{x}_i \in S_-, & \quad \langle \bar{\boldsymbol{w}}, \phi(\mathbf{x}_i) \rangle + w_0 \leq -1 + \xi_i \end{aligned}$$

-

$$\min_{\bar{\boldsymbol{w}} \in \mathbb{H}, w_0 \in \mathbb{R}, \xi \in \mathbb{R}^m} \frac{1}{2} ||\bar{\boldsymbol{w}}||^2 + C \sum_{i=1}^{m} \xi_i$$

$$\text{u.c.} \quad \forall i, \xi_i \geq 0 \quad \text{et} \quad y_i(\langle \bar{\boldsymbol{w}}, \phi(\mathbf{x}_i) \rangle + w_0) \geq 1 - \xi_i$$

- The Lagrangian formulation

$$L_p(\bar{\boldsymbol{w}}, w_0, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\bar{\boldsymbol{w}}\|^2 + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i [y_i(\langle \phi(\mathbf{x}_i), \bar{\boldsymbol{w}} \rangle + w_0) - 1 + \xi_i] - \sum_{i=1}^{m} \beta_i \xi_i$$

SVM: soft margin

- The Lagrangian formulation

$$\nabla L_p(\bar{\boldsymbol{w}}^*) = \bar{\boldsymbol{w}}^* - \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i) = 0, \quad \text{i.e.} \quad \bar{\boldsymbol{w}}^* = \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i)$$

$$\nabla L_p(w_0^*) = -\sum_{i=1}^{m} \alpha_i y_i = 0, \quad \text{i.e.} \quad \sum_{i=1}^{m} \alpha_i y_i = 0$$

$$\nabla L_p(\boldsymbol{\xi}^*) = C \times \mathbf{1}_m - \boldsymbol{\alpha} - \boldsymbol{\beta} = 0, \quad \text{i.e.} \quad \forall i, C = \alpha_i + \beta_i$$

$$\forall i, \alpha_i \left[y_i h_{\boldsymbol{w}^*}(\mathbf{x}_i) - 1 + \xi_i^* \right] = 0, \quad \text{i.e.} \quad \alpha_i = 0 \text{ or } y_i h_{\boldsymbol{w}^*}(\mathbf{x}_i) = 1 - \xi_i^*$$

$$\forall i, \beta_i \xi_i^* = 0, \quad \text{i.e.} \quad \beta_i = 0 \text{ or } \xi_i^* = 0$$

- That to say

$$L_p = \frac{1}{2} \left\| \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i) \right\|^2 - \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) - w_0 \underbrace{\sum_{i=1}^{m} \alpha_i y_i}_{=0} + \sum_{i=1}^{m} \alpha_i$$

$$= -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{m} \alpha_i$$

SVM: soft margin

- We get the same dual form than for the linearly separable case

$$\max_{(\alpha_1,\ldots,\alpha_m)\in\mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{u.c.} \quad \sum_{i=1}^{m} y_i \alpha_i = 0 \text{ et } \forall i, 0 \leq \alpha_i \leq C$$

- With the only difference that $0 \leq \alpha_i \leq C$
- Using the kernel trick, the resolution of the optimization problem requires only the expression of the kernel function κ without explicitly knowing the mapping ϕ.
- Many free implementations, see `http://www.kernel-machines.org`

Universal approximation capability of SVMs

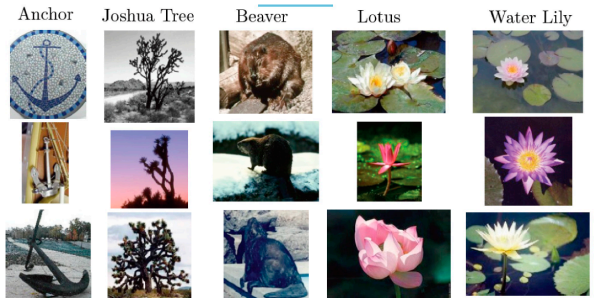
Theorem (Hammer & Gersmann (2001))

SVMs with standard kernels $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *(including Gaussian, polynomial, and several dot product kernels) learned over a training set* $S = (\mathbf{x}_i, y_i)_{1 \leq i \leq m}$ *can approximate any continuous function* f *on compact subsets of* $\mathbb{R}^d$ *up to any desired precision* $\epsilon \in \mathbb{R}_+^*$

$$\forall \epsilon > 0; \forall \mathbf{x} \in \mathbb{R}^d; \left\| f(\mathbf{x}) - \left(\sum_{i=1}^{m} y_i \alpha_i^* \kappa(\mathbf{x}_i, \mathbf{x}) + w_0^* \right) \right\|_\infty \leq \epsilon$$

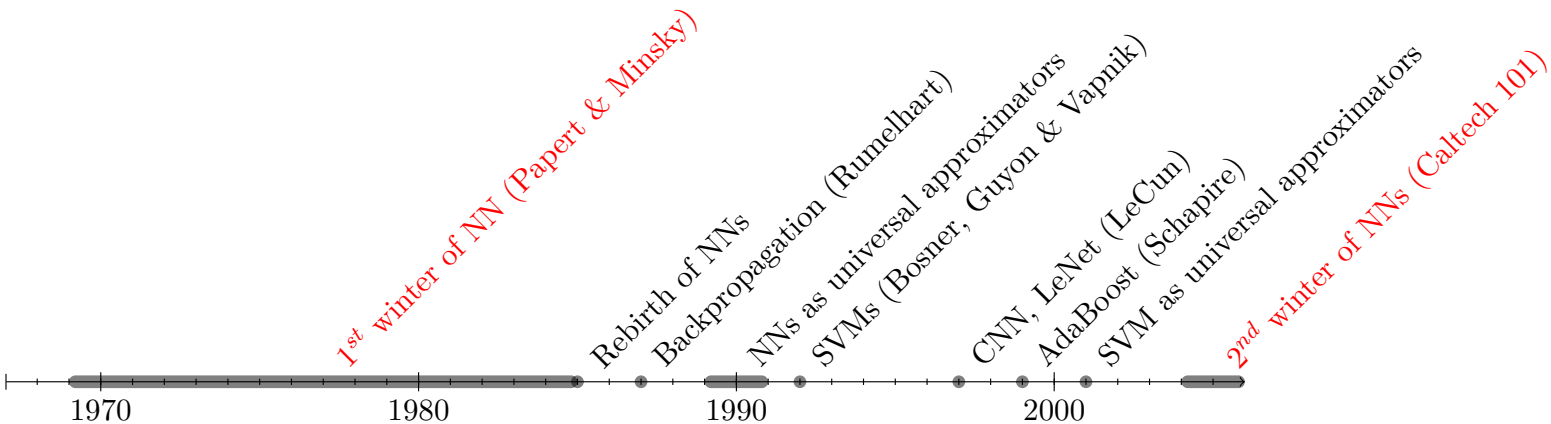
- For a given learning problem research were mainly focused on finding adapted feature characteristics and kernel functions and use SVMs for prediction.

The Caltech 101 database (2004)



- 101 classes, 30 training images per class ⇒ SVMs were the winner !

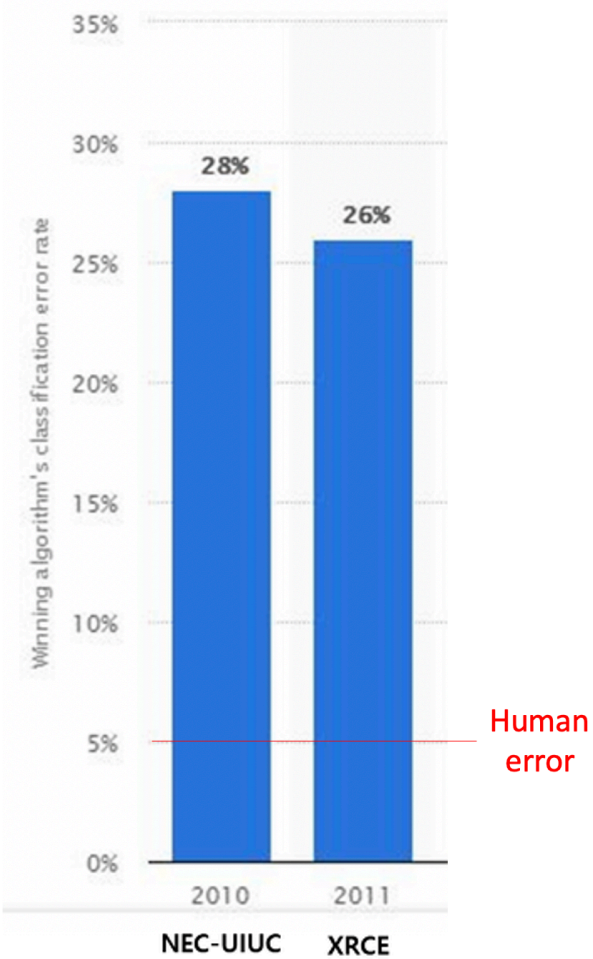
⇒ This marked the 2^{nd} winter of NNs, SVM became the most popular learning algorithms.



The ImageNet database (Deng et al., 2009)

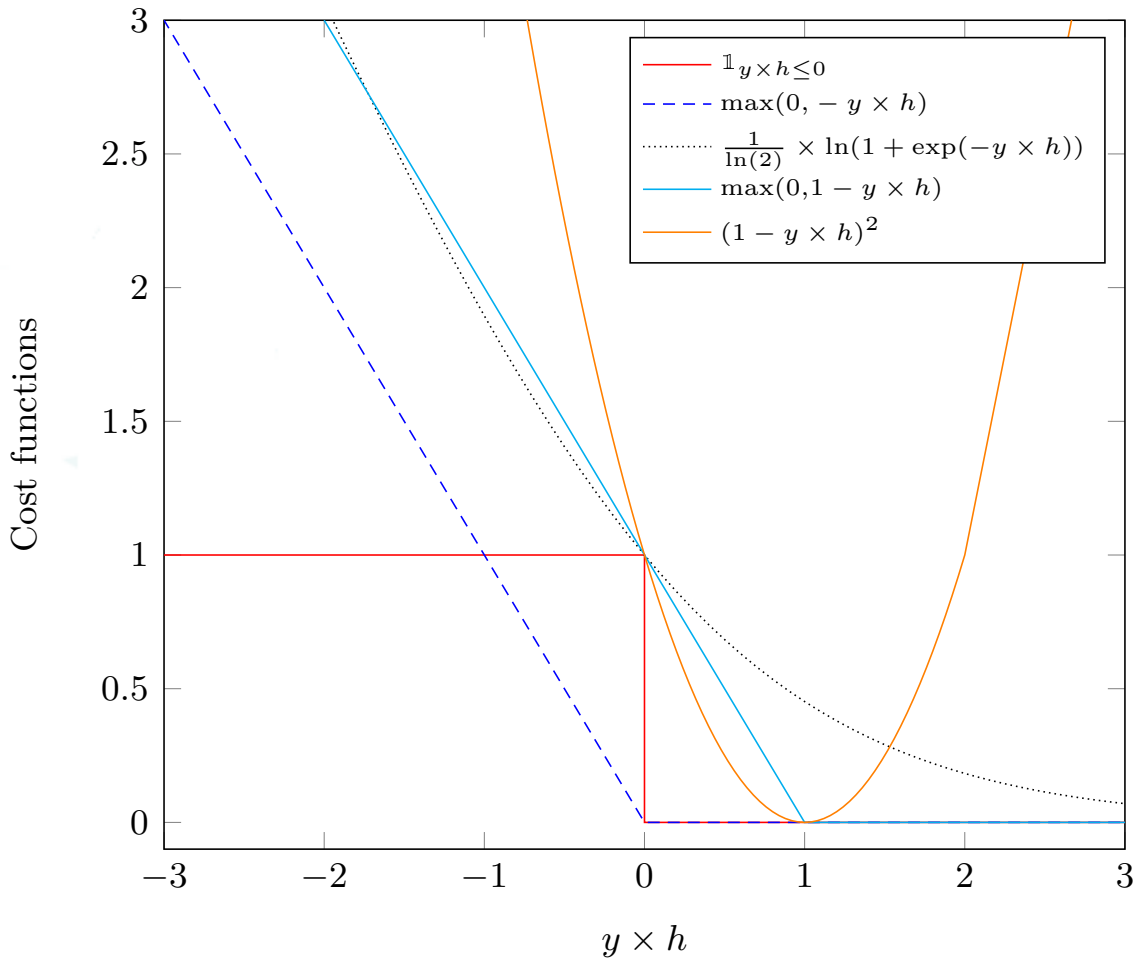
- 15 million high resolution images belonging to 22,000 classes.
- Large-Scale Visual Recognition Challenge (a subset of ImageNet - `http://image-net.org/challenges/LSVRC/2010/`)
  - 1000 classes,
  - 1.2 millions training images,
  - 50,000 validation images,
  - 150,000 test images.

SVMs at LSVRC 2010 & 2011



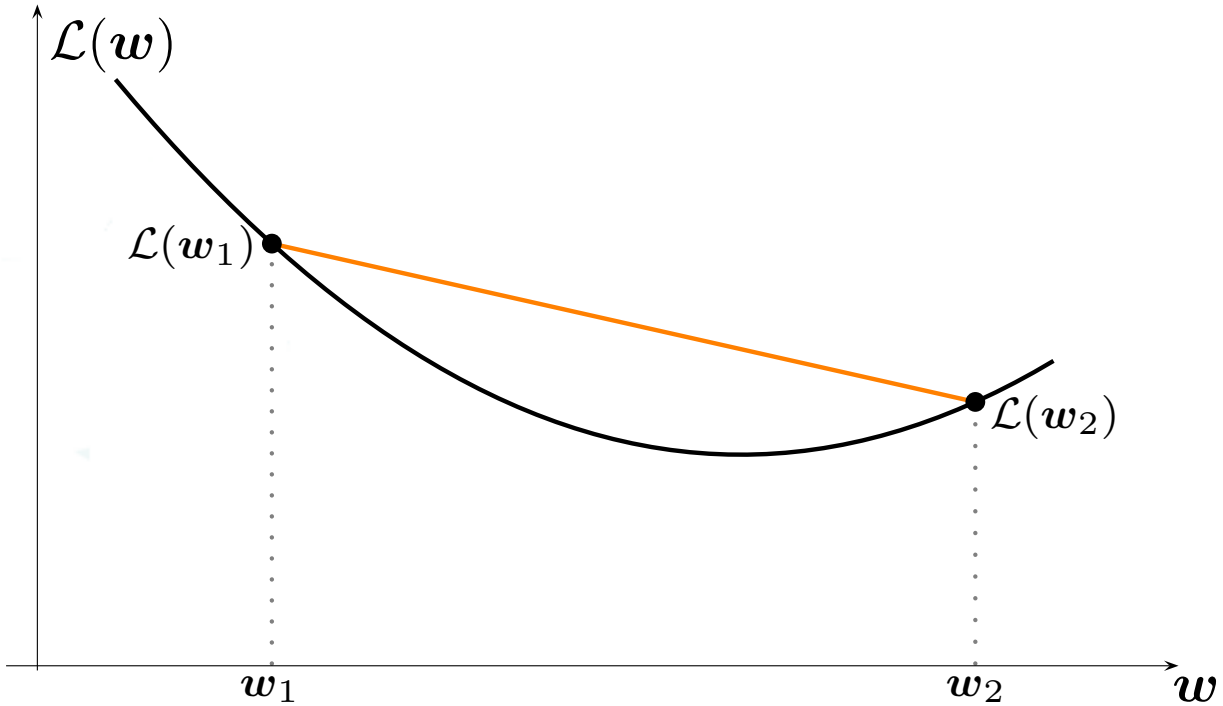
On the convergence of the Gradient Descent algorithm

Convex upper bounds of the misclassification error



$\Rightarrow$ Using a convex upper bound of the 0/1 loss, SRM casts into the problem of minimizing a convex objective function.

Convex functions: General definition

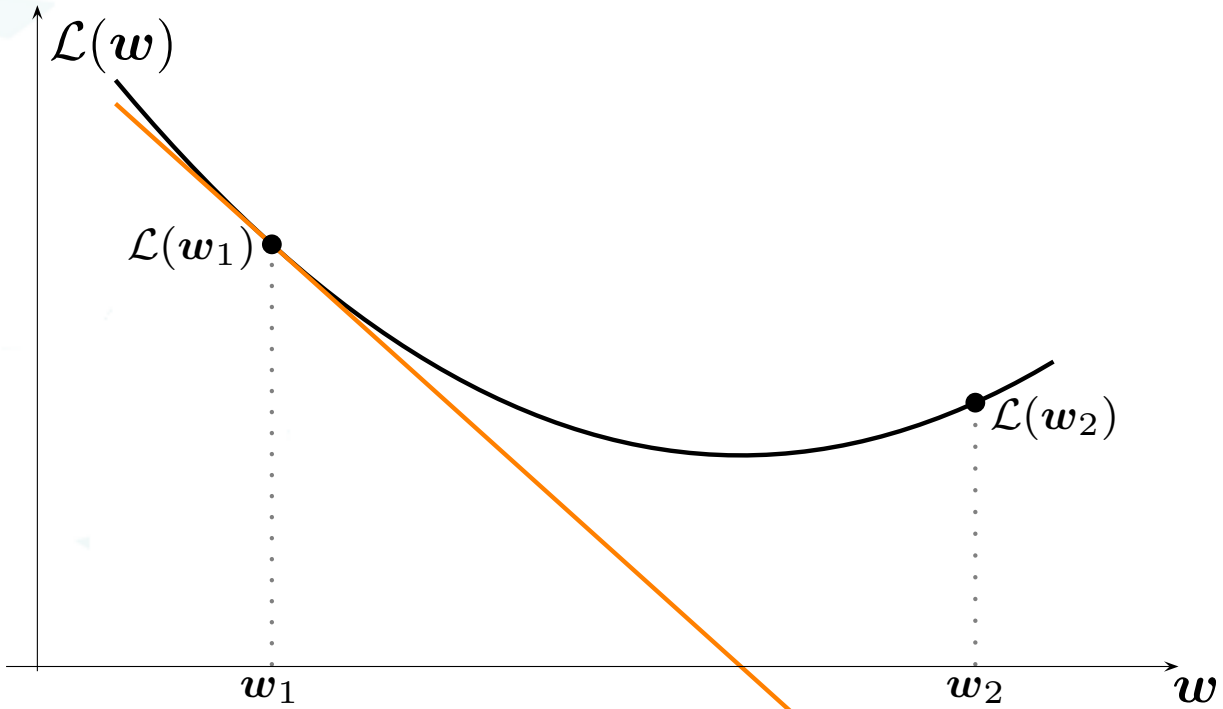


$$\forall \boldsymbol{w}_1, \boldsymbol{w}_2, \theta \in [0,1]; \mathcal{L}(\theta \boldsymbol{w}_1 + (1-\theta)\boldsymbol{w}_2) \leqslant \theta \mathcal{L}(\boldsymbol{w}_1) + (1-\theta)\mathcal{L}(\boldsymbol{w}_2)$$

- The extension of the above is called Jensen's inequality:

$$\forall \boldsymbol{w}_i, \theta_i; i \in \{1, \ldots, N\}; \text{ s.t. } \theta_i \geqslant 0, \sum_{i=1}^{N} \theta_i = 1; \mathcal{L}\left(\sum_{i=1}^{N} \theta_i \boldsymbol{w}_i\right) \leqslant \sum_{i=1}^{N} \theta_i \mathcal{L}(\boldsymbol{w}_i)$$

Convex functions: Differentiable case



If $\mathcal{L}$ is differentiable, it is convex iff

$$\forall \boldsymbol{w}_1, \boldsymbol{w}; \mathcal{L}(\boldsymbol{w}) \geqslant \mathcal{L}(\boldsymbol{w}_1) + \nabla \mathcal{L}(\boldsymbol{w}_1)(\boldsymbol{w} - \boldsymbol{w}_1)$$

If $\mathcal{L}$ is twice differentiable, it is convex iff its Hessian matrix $\mathbf{H}$ is positive semidefinite on the interior of the convex set $\mathcal{S}$;

$$\forall (\boldsymbol{w}, \boldsymbol{w}') \in \mathcal{S}^2; \boldsymbol{w}^\top \mathbf{H} \boldsymbol{w}' \geqslant 0.$$

Property

- Consider the Taylor expansion of a convex, continuous and twice differentiable objective function around its minimiser

$$\hat{\mathcal{L}}(\boldsymbol{w}) = \hat{\mathcal{L}}(\boldsymbol{w}^*) + (\boldsymbol{w}-\boldsymbol{w}^*)^\top \underbrace{\nabla\hat{\mathcal{L}}(\boldsymbol{w}^*)}_{=0} + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^\top \mathbf{H}(\boldsymbol{w}-\boldsymbol{w}^*) + o(\| w-w^* \|^2)$$

- The Hessian matrix is symmetric hence its eigenvectors $(\boldsymbol{v}_i)_{i=1}^d$ form an orthonormal basis.

$$\forall (i,j) \in \{1,\dots,d\}^2, \mathbf{H}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \text{ et } \boldsymbol{v}_i^\top \boldsymbol{v}_j = \begin{cases} +1 : \text{si } i = j, \\ 0 : \text{otherwise.} \end{cases}$$

A good lecture on symmetric matrices and eigendecomposition can be found here!

Property (2)

- Every weight vector $\boldsymbol{w} - \boldsymbol{w}^*$ can be uniquely decomposed in this basis

$$\boldsymbol{w} - \boldsymbol{w}^* = \sum_{i=1}^{d} q_i v_i$$

- That to say

$$\hat{\mathcal{L}}(\boldsymbol{w}) = \hat{\mathcal{L}}(\boldsymbol{w}^*) + \frac{1}{2} \sum_{i=1}^{d} \lambda_i q_i^2$$

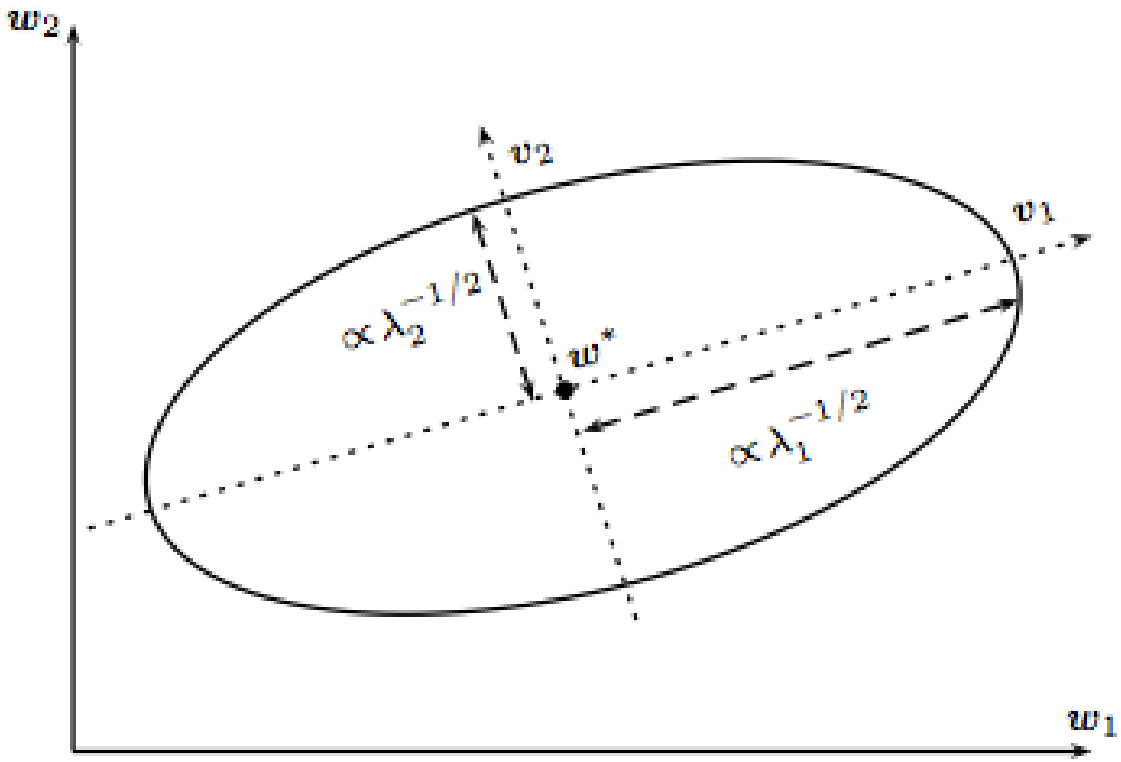
- Furthermore if $\boldsymbol{w}$ is in the neighborhood of the minimizer $\boldsymbol{w}^*$, we have by the definition of the minimum:

$$(\boldsymbol{w} - \boldsymbol{w}^*)^\top \mathbf{H} (\boldsymbol{w} - \boldsymbol{w}^*) = \sum_{i=1}^{d} \lambda_i q_i^2 = 2(\hat{\mathcal{L}}(\boldsymbol{w}) - \hat{\mathcal{L}}(\boldsymbol{w}^*)) > 0$$

$\mathbf{H}$ defined at $\boldsymbol{w}^*$ is hence definite positive and all its eigenvalues are strictly positive.

Property (3)

- This implies that the level lines of $\hat{\mathcal{L}}$ at the neighborhood of $\boldsymbol{w}^*$, defined by weight points for which $\hat{\mathcal{L}}$ is constant, are ellipses



Gradient descent algorithm [Cauchy, 1867]

At each iteration t, on $\boldsymbol{w}^{(t)}$

- Estimate the descent direction $\mathbf{p}_t$ (i.e. $\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) < 0$)
- Update

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t$$

// Where η_t is a positive learning rate making $\boldsymbol{w}^{(t+1)}$ be acceptable for the next iteration.

$\Rightarrow$ For its low complexity, GD is one of the main algorithms that is used for the minimization of learning objective functions.

Convergence of GD when $p_t = -\nabla\hat{\mathcal{L}}(\boldsymbol{w}^t)$

- Take the decomposition of any vector $\boldsymbol{w} - \boldsymbol{w}^*$ in the orthonormal basis $(\boldsymbol{v}_i)_{i=1}^d$ formed by the eigenvectors of the Hessian matrix

$$\nabla\hat{\mathcal{L}}(\boldsymbol{w}) = \sum_{i=1}^{d} q_i \lambda_i \boldsymbol{v}_i$$

- Let $\boldsymbol{w}^{(t)}$ be the weight vector obtained from $\boldsymbol{w}^{(t-1)}$ after applying the gradient descent rule with $p_t = -\nabla\hat{\mathcal{L}}(\boldsymbol{w}^t)$:

$$\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)} = \sum_{i=1}^{d} \left(q_i^{(t)} - q_i^{(t-1)} \right) \boldsymbol{v}_i = -\eta \nabla\hat{\mathcal{L}}(\boldsymbol{w}^{(t-1)}) = -\eta \sum_{i=1}^{d} q_i^{(t-1)} \lambda_i \boldsymbol{v}_i$$

- So

$$\forall i \in \{1, \ldots, d\}, q_i^{(t)} = (1 - \eta\lambda_i)^t q_i^{(0)}$$

and the algorithm converges iff

$$0 < \eta < \frac{2}{\lambda_{max}}$$

OK but how to find the good learning rate? Wolfe conditions

- To find the sequence $(\boldsymbol{w}^{(t)})_{t\in\mathbb{N}}$ following the line search rule, the following necessary condition

$$\forall t \in \mathbb{N}, \hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) < \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

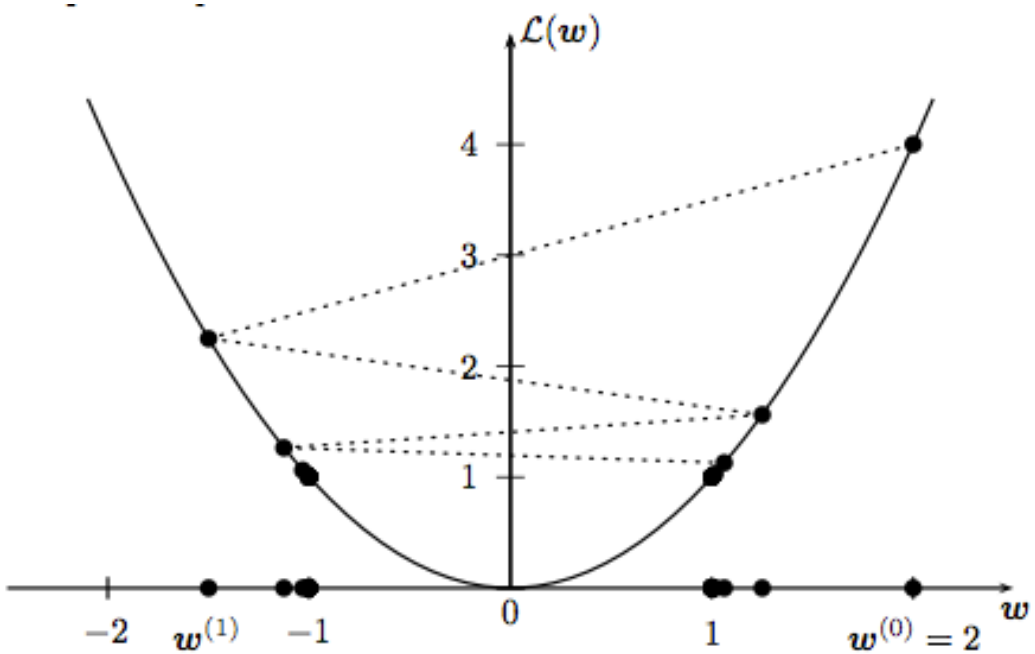
  is not sufficient to guarantee the convergence of the sequence to the minimiser of $\hat{\mathcal{L}}$.

- In two situations, the previous condition is satisfied but there is no convergence

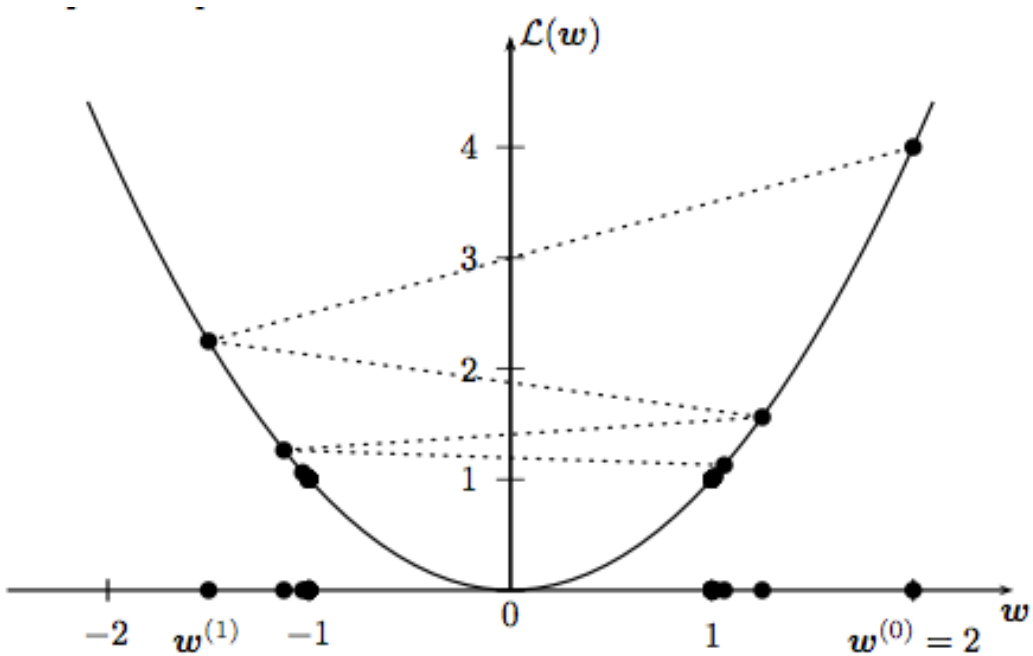
1. The decreasing of $\hat{\mathcal{L}}$ is too small with respect to the length of the jumps

Consider the following example $d = 1$; $\hat{\mathcal{L}}(\boldsymbol{w}) = \boldsymbol{w}^2$ with $\boldsymbol{w}^{(0)} = 2$, $(\mathbf{p}_t = (-1)^{t+1})_{t\in\mathbb{N}^*}$ and $(\eta_t = (2 + \frac{3}{2^{t+1}}))_{t\in\mathbb{N}^*}$. The sequence of updates would then be

$$\forall t \in \mathbb{N}^*, \boldsymbol{w}^{(t)} = (-1)^t(1 + 2^{-t})$$



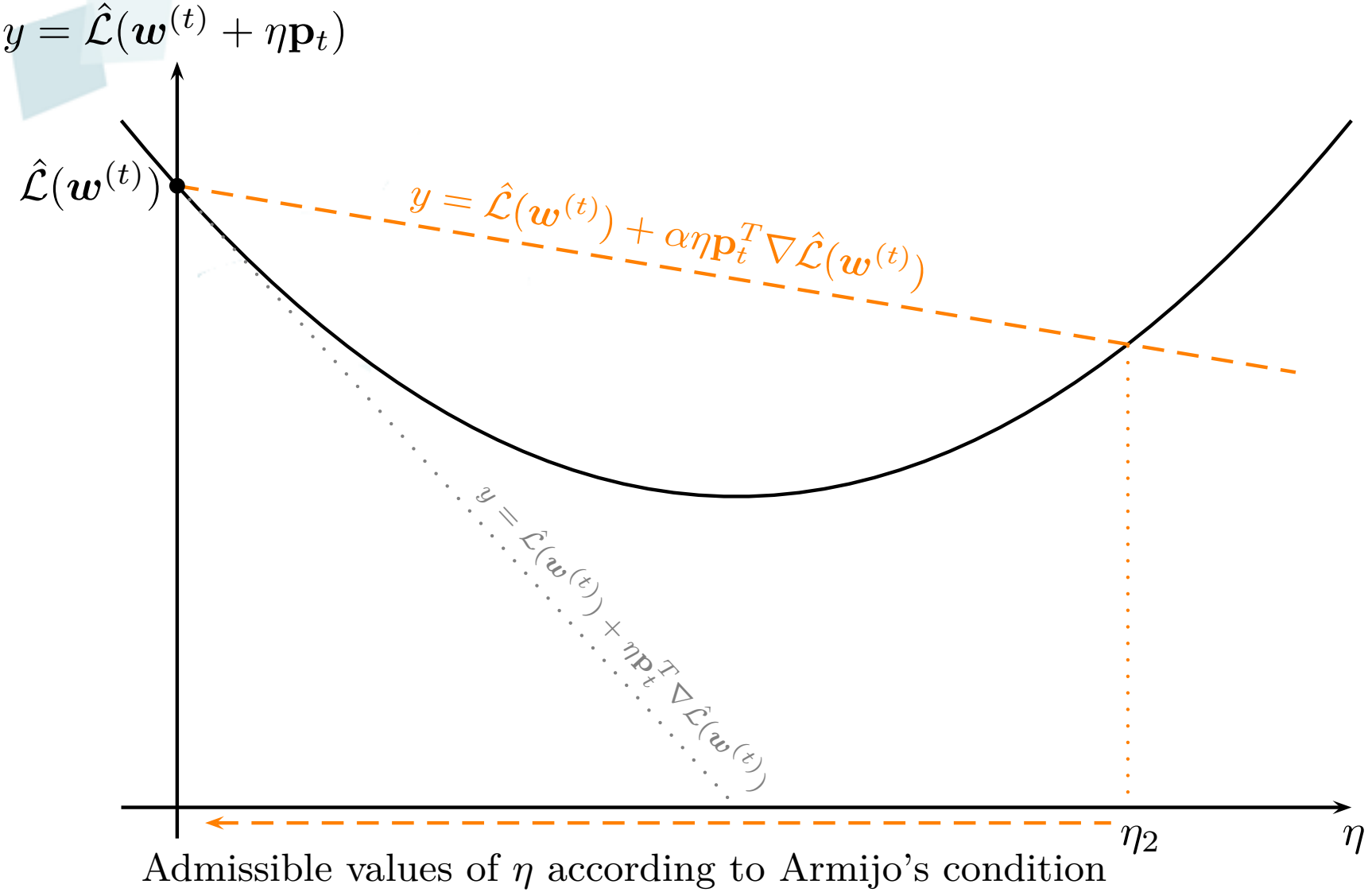
1. The decreasing of $\hat{\mathcal{L}}$ is too small with respect to the length of the jumps



$\Rightarrow$ Armijo condition : require that for a given $\alpha \in (0,1)$,

$$\forall t \in \mathbb{N}^*, \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t) \leqslant \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \eta_t \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

Armajio condition

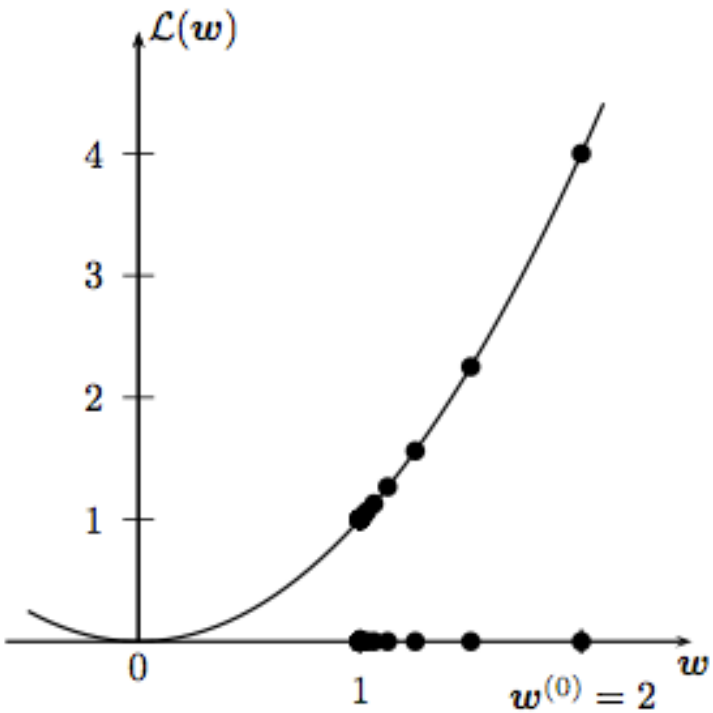


Admissible values of η according to Armijo's condition

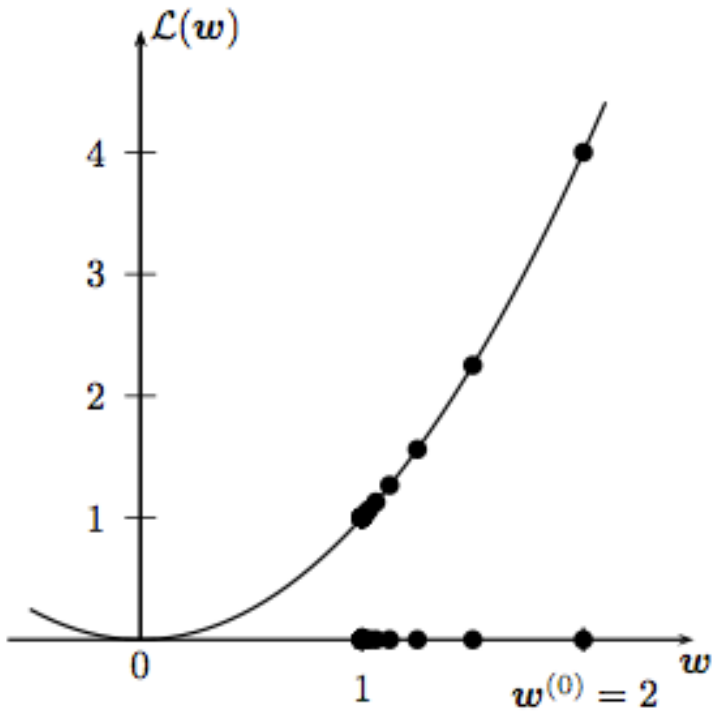
2. The jumps of the weight vectors are too small

Consider the following example $d = 1$; $\hat{\mathcal{L}}(\boldsymbol{w}) = \boldsymbol{w}^2$ with $\boldsymbol{w}^{(0)} = 2$, $(\mathbf{p}_t = -1)_{t \in \mathbb{N}^*}$ and $(\eta_t = (2^{-t+1}))_{t \in \mathbb{N}^*}$. The sequence of updates would then be

$$\forall t \in \mathbb{N}^*, \boldsymbol{w}^{(t)} = (1 + 2^{-t})$$



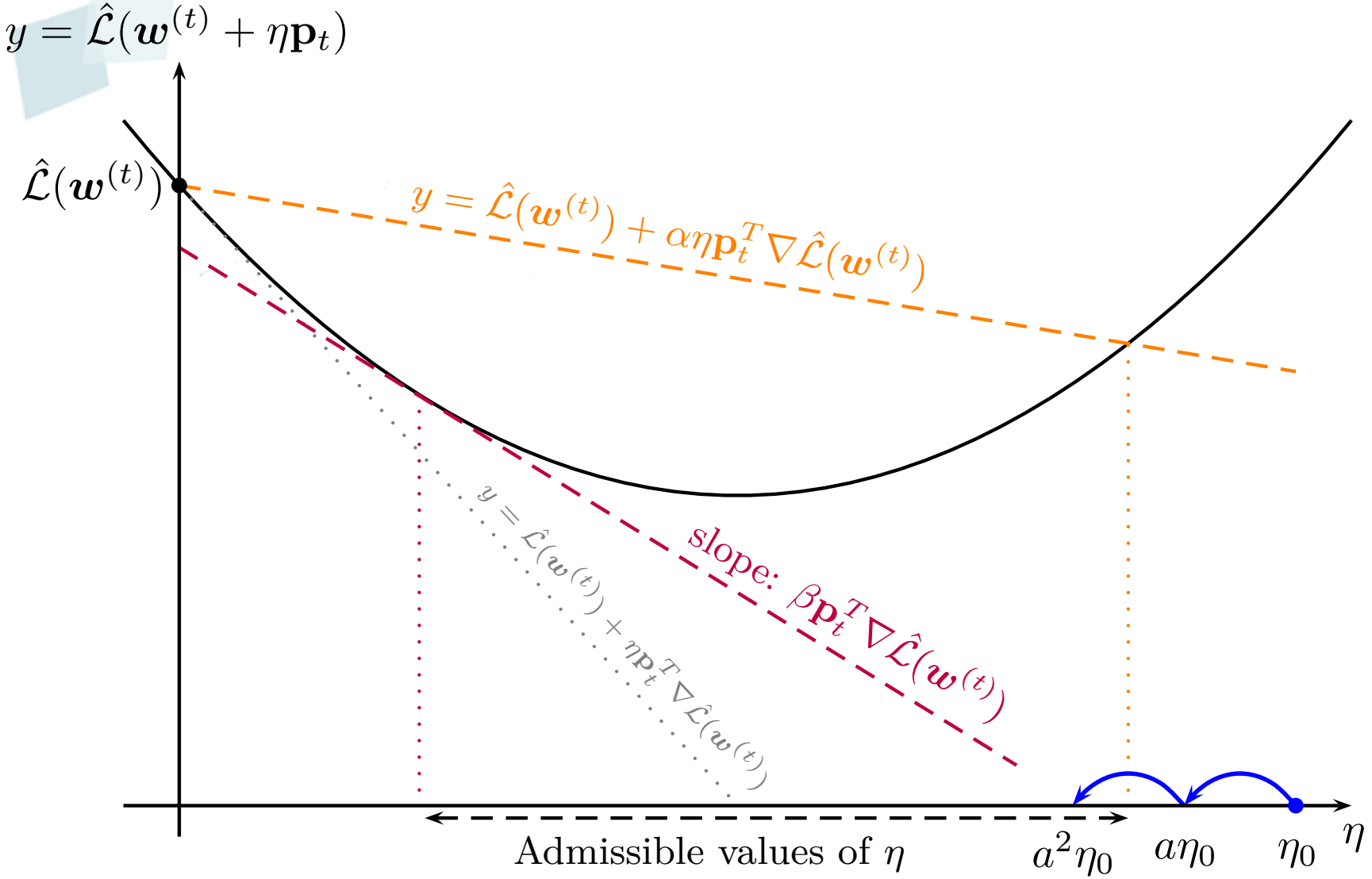
2. The jumps of the weight vectors are too small



$\Rightarrow \exists \beta \in (\alpha, 1)$ such that

$$\forall t \in \mathbb{N}^*, \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t) \geq \beta \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

Curvature condition



Existence of learning rates verifying Wolfe conditions

- Let $\mathbf{p}_t$ be a descent direction of $\hat{\mathcal{L}}$ at $\boldsymbol{w}^{(t)}$. Suppose that the function $\psi_t : \eta \mapsto \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta \mathbf{p}_t)$ is derivative and lower bounded, then there exists η_t verifying both Wolfe conditions.

proof:

1. consider

$$E = \{a \in \mathbb{R}_+ \mid \forall \eta \in]0, a], \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta \mathbf{p}_t) \leqslant \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \eta \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})\}$$

As $\mathbf{p}_t$ is a descent direction of $\hat{\mathcal{L}}$ at $\boldsymbol{w}^{(t)}$ then for all $\alpha < 1$ there exists $\bar{a} > 0$ such that

$$\forall \eta \in]0, \bar{a}], \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta \mathbf{p}_t) < \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \eta \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

Existence of learning rates verifying Wolfe conditions

2. So $E \neq \emptyset$. Furthermore, as the function ψ_t is lower bounded, the largest rate in E, $\hat{\eta}_t = \sup E$, exists. By continuity of ψ_t we have

$$\hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \hat{\eta}_t \mathbf{p}_t) < \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \hat{\eta}_t \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

3. Let $(\eta_n)_{n\in\mathbb{N}}$ be a convergence sequence to $\hat{\eta}_t$ by higher values, i.e. $\forall n \in \mathbb{N}, \eta_n > \hat{\eta}_t$ and $\lim_{n\to+\infty} \eta_n = \hat{\eta}_t$. As $(\eta_n)_{n\in\mathbb{N}} \notin E$ we get

$$\forall n \in \mathbb{N}, \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \eta_n \mathbf{p}_t) > \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \eta_n \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

So

$$\hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \hat{\eta}_t \mathbf{p}_t) = \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) + \alpha \hat{\eta}_t \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

Existence of learning rates verifying Wolfe conditions

4. We finally get

$$\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)} + \hat{\eta}_t \mathbf{p}_t) \geq \alpha \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) \geq \beta \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$$

Where $\beta \in (\alpha, 1)$ and $\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) < 0$.

$\Rightarrow$ The learning rate $\hat{\eta}_t$ verifies both Wolfe conditions

Does it work?

Theorem (Zoutendijk)

Let $\hat{\mathcal{L}} : \mathbb{R}^d \to \mathbb{R}$ be a differentiable objective function with a lipschtizien gradient and lower bounded. Let $\mathfrak{A}$ be an algorithm generating $(\boldsymbol{w}^{(t)})_{t\in\mathbb{N}}$ defined by

$$\forall t \in \mathbb{N}, \boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \mathbf{p}_t$$

where $\mathbf{p}_t$ is a descent direction of $\hat{\mathcal{L}}$ and η_t a learning rate verifying both Wolfe conditions. By considering the angle θ_t between the descent direction $\mathbf{p}_t$ and the direction of the gradient :

$$\cos(\theta_t) = \frac{\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})}{||\hat{\mathcal{L}}(\boldsymbol{w}^{(t)})|| \times ||\mathbf{p}_t||}$$

The following series is convergent

$$\sum_t \cos^2(\theta_t) ||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})||^2$$

Proof of Zoutendijk's theorem

1. Using the second Wolfe's condition and by subtracting $\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})$ from both terms of the inequality, we get

$$\forall t, \mathbf{p}_t^\top (\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) - \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})) \geq (\beta - 1) \left(\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) \right)$$

2. Using the lipschitzian property of the gradient of the objective function

$$\begin{aligned} \mathbf{p}_t^\top (\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) - \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})) &\leqslant ||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) - \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})|| \times ||\mathbf{p}_t|| \\ &\leqslant L||\boldsymbol{w}^{(t+1)} - \boldsymbol{w}^{(t)}|| \times ||\mathbf{p}_t|| \\ &\leqslant L\eta_t ||\mathbf{p}_t||^2 \end{aligned}$$

Proof of Zoutendijk's theorem

3. By combining both inequalities it comes

$$\forall t, 0 \leqslant (\beta - 1)(\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})) \leqslant L\eta_t ||\mathbf{p}_t||^2$$

4. For $\eta_t \geq \frac{\beta - 1}{L} \frac{\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})}{||\mathbf{p}_t||^2} > 0$ we get from Armijo's condition

$$\begin{aligned}
\hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) - \hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) &\geq -\alpha\eta_t \mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) \\
&\geq \alpha \frac{1-\beta}{L} \frac{(\mathbf{p}_t^\top \nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}))^2}{||\mathbf{p}_t||^2} \\
&\geq \alpha \frac{1-\beta}{L} \cos^2(\theta_t) ||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})||^2 \geq 0
\end{aligned}$$

Proof of Zoutendijk's theorem

5. The objective function is lower bounded, the sequence of general term $\hat{\mathcal{L}}(\boldsymbol{w}^{(t)}) - \hat{\mathcal{L}}(\boldsymbol{w}^{(t+1)}) > 0$ is convergent

6. Hence, the series

$$\sum_t \cos^2(\theta_t) ||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})||^2$$

is convergent.

Corollary of Zoutendijk's theorem Guarantee of convergence

- In the case where, the descente direction and the gradient are not orthogonal :

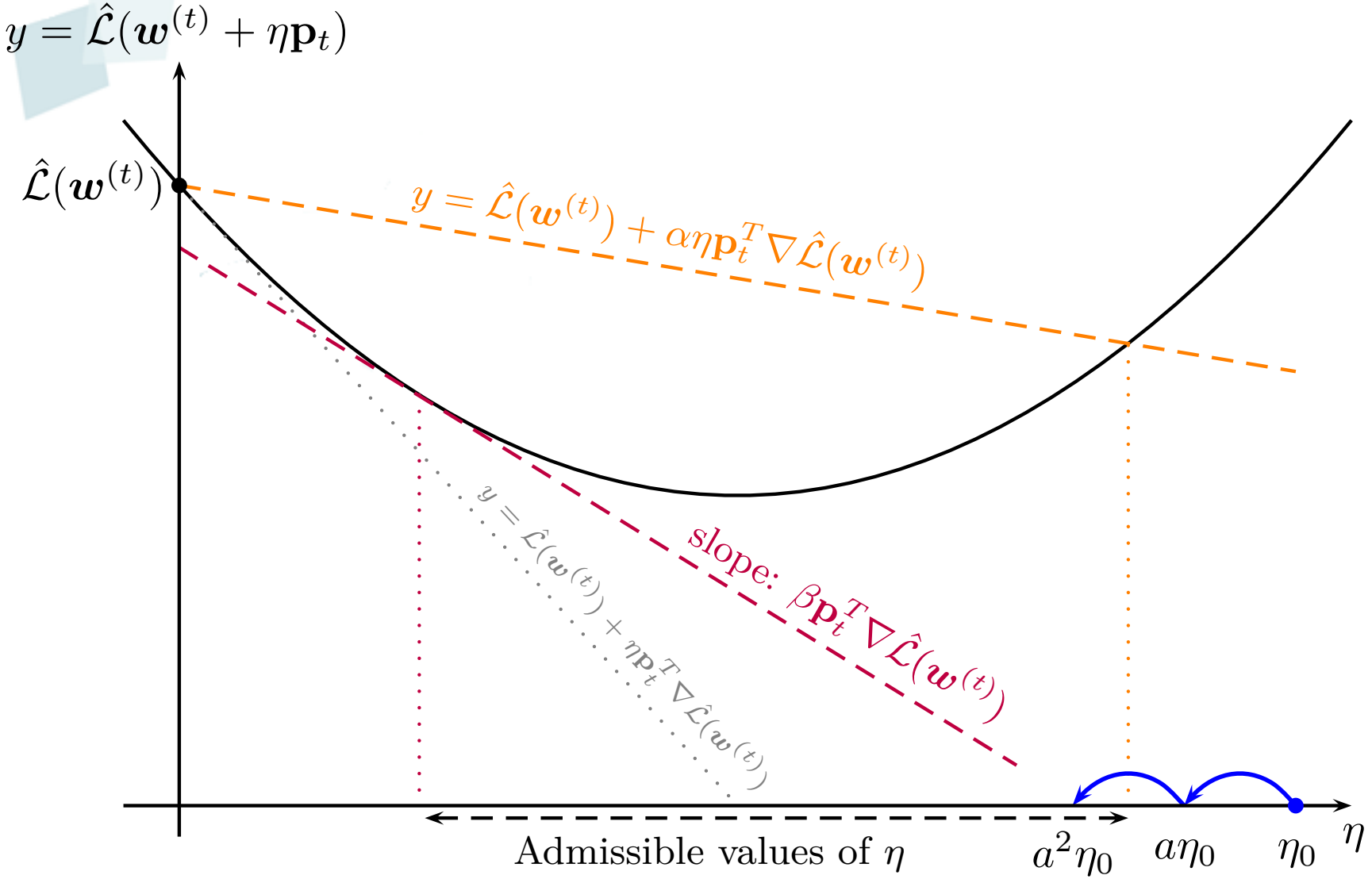
$$\exists \kappa > 0, \forall t \geq T, \cos^2(\theta_t) \geq \kappa$$

- Following Zoutendijk's theorem the series :

$$\sum_t ||\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)})||^2$$

  is convergent.
- Hence, the sequence $(\nabla \hat{\mathcal{L}}(\boldsymbol{w}^{(t)}))_t$ tends to 0 when t tends to infinity.

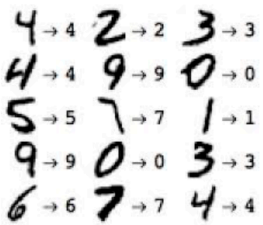
Backtracking Line Search



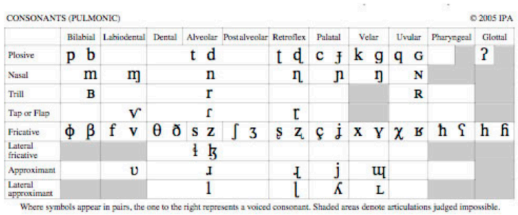
Multiclass Classification

Real-life classification applications

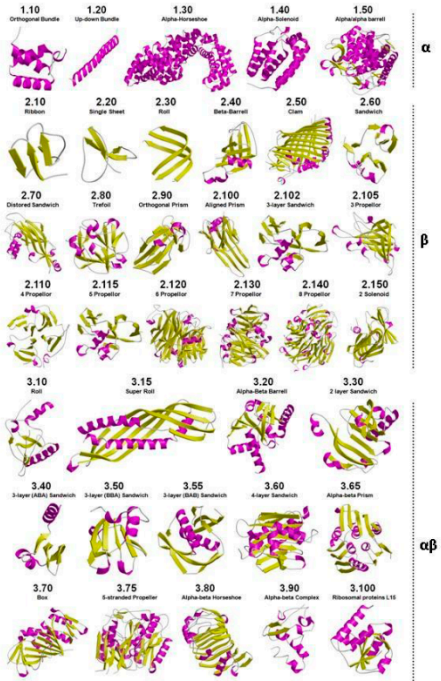
- In most real-life classification applications the number of classes is more than two.



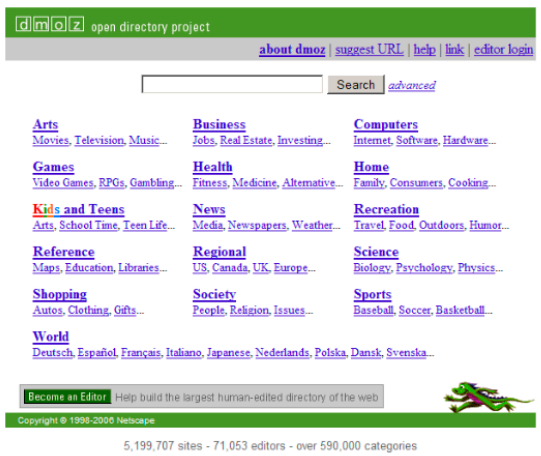
Digit recognition
$K = 10$



Phoneme recognition
$K = 50$



Protein classification
$K \in [300 - 600]$



Text classification
$K > 10^5$

Multi-class classification problem

- There are two cases to be distinguished :
  - the *mono-label case*, where each example is labeled with a single class. In this case the output space $\mathcal{Y}$ is a finite set of classes marked generally with numbers for convenience $\mathcal{Y} = \{1, \ldots, K\}$,
  - the *multi-label case*, where each example can be labeled with several classes ; $\mathcal{Y} = \{1, +1\}^K$.
- In both cases, the learning algorithm takes a labeled training set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$ in input where pairs of examples $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ are supposed i.i.d. with respect to an unknown yet fixed probability distribution $\mathcal{D}$.

Multi-class classification problem

- The aim of learning is then to find a prediction function from $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ with the lowest generalization error :

$$\mathcal{L}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(f(\mathbf{x}), y)], \tag{14}$$

  where, $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is the instantaneous classification error, and $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})) \in \mathcal{Y}$ is a predicted output vector for example $\mathbf{x}$ in the multi-label case, or the class label of $\mathbf{x}$ in the mono-label case.

- In the multi-label case, the instantaneous error is based on the Hamming distance that counts the number of different components in the predicted, $f(\mathbf{x})$, and the true class, y, labels for $\mathbf{x}$.

$$\ell(f(\mathbf{x}), y) = \frac{1}{2}\sum_{k=1}^{K}(1 - y_k f_k(\mathbf{x}))$$

Multi-class classification problem

- In the mono-label case, the instantaneous error is simply:

$$\ell(f(\mathbf{x}), y) = \mathbb{1}_{f(\mathbf{x}) \neq y}$$

- As in binary classification, the prediction function is found according to the Empirical Risk Minimization principle using a training set $S = \{(\mathbf{x}_i, y_i); i \in \{1, \ldots, m\}\} \in (\mathcal{X} \times \mathcal{Y})^m$:

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{\mathcal{L}}_m(f, S) = \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \ell(f(\mathbf{x}_i), y_i)$$

Multi-class classification problem

- In practice, the function learned $\boldsymbol{h}$ is defined as :

$$\begin{aligned} \boldsymbol{h} : \mathbb{R}^d &\to \mathbb{R}^K \\ \mathbf{x} &\mapsto (h(\mathbf{x}, 1), \ldots, h(\mathbf{x}, K)) \end{aligned}$$

  where $h \in \mathbb{R}^{\mathcal{X} \times \mathcal{Y}}$, by minimizing a convex derivative upper bound of the empirical error.

- For an example $\mathbf{x}$, the prediction is hence obtained by thresholding the outputs $h(\mathbf{x}, k), k \in \{1, \ldots, K\}$ for the multi-label case, or by taking the class index giving the highest prediction in the mono-label case:

$$\forall \mathbf{x}, f_{\boldsymbol{h}}(\mathbf{x}) = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; h(\mathbf{x}, k)$$

Approaches

- Combined approaches (on the basis of binary classification)
  - One-versus-All (OvA),
  - One-versus-One (OvO),
  - Error-correction codes (ECOC).

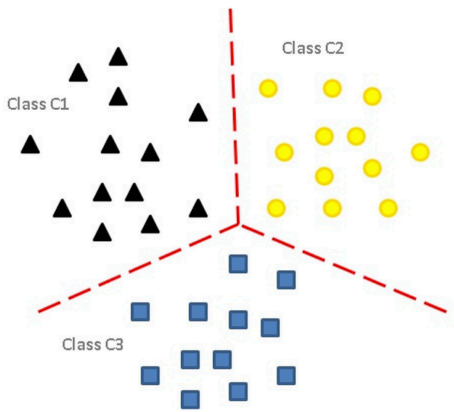
- Uncombined approaches
  - K-Nearest Neighbour (K-NN),
  - Generative models,
  - Discriminative models (MLP, M-SVM, M-AdaBoost, etc.)

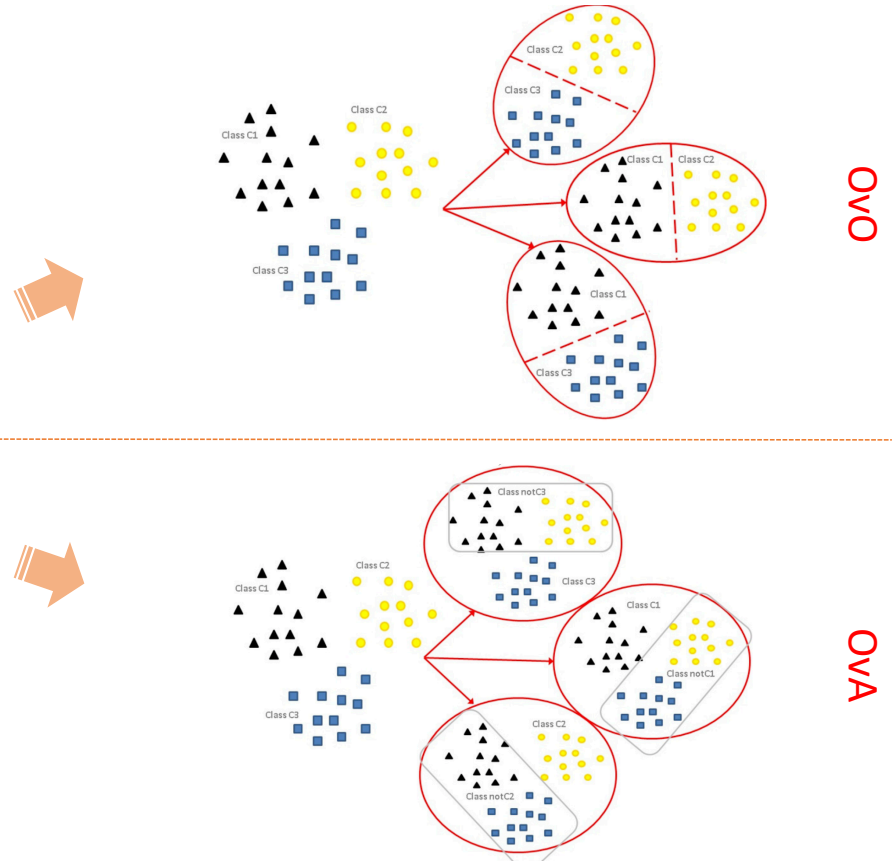
Combined approach - OvA

Algorithm 9 The OVA approach

Training set $S = \{(x_i, y_i) \mid i \in \{1, \ldots, m\}\}$
for each $k = 1 \ldots K$ **do**
    $\tilde{S} \leftarrow \emptyset$
    for each $i = 1 \ldots m$ **do**
        if $y_i == k$ **then**
            $\tilde{S} \leftarrow (\mathbf{x}_i, +1)$
        else
            $\tilde{S} \leftarrow (\mathbf{x}_i, -1)$
        end if
    end for each
    Learn a classifier $h_k : \mathcal{X} \to \mathbb{R}$ on $\tilde{S}$;
end for each
The final classifier: $\forall \mathbf{x} \in \mathcal{X}, f(\mathbf{x}) = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; h_k(\mathbf{x})$

Combined approach - OvA





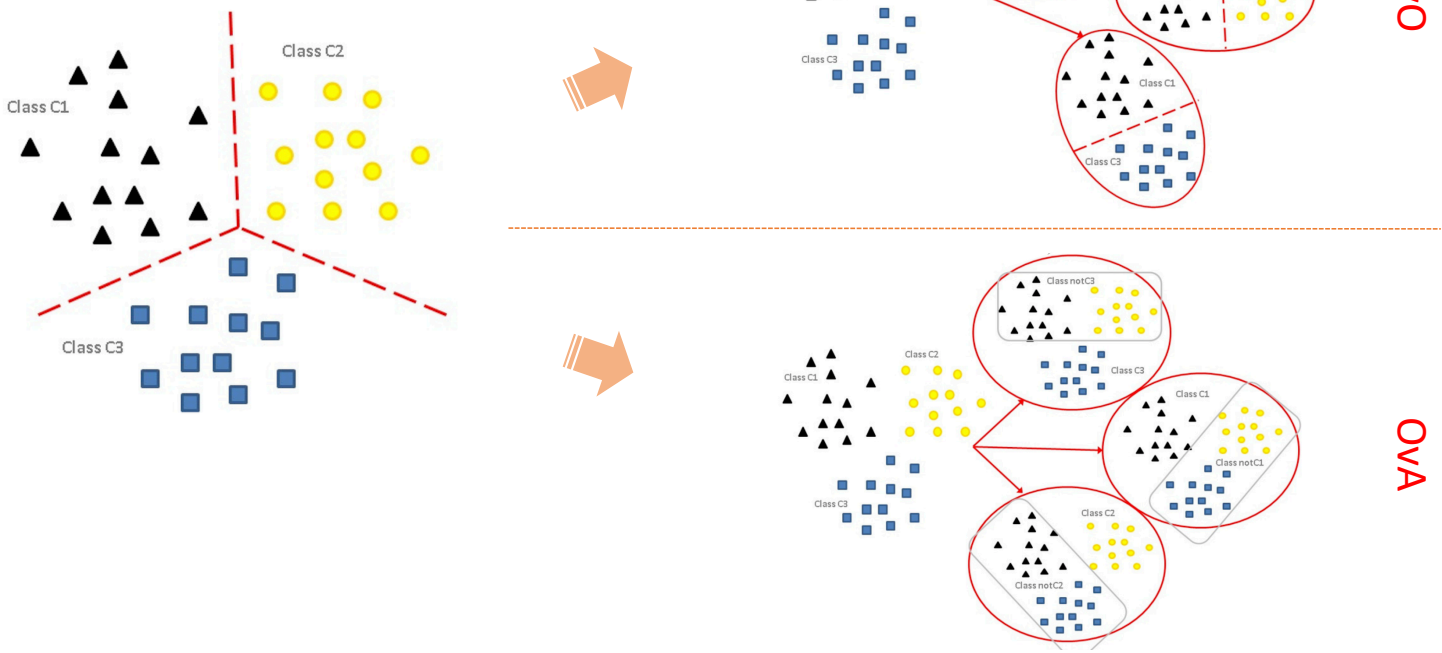
Combined approach - OvO

Algorithm 10 The OVO approach

Training set $S = \{(x_i, y_i) \mid i \in \{1, \dots, m\}\}$
for each $k = 1 \dots K-1$ **do**
  for each $\ell = k+1 \dots K$ **do**
    $\tilde{S} \leftarrow \emptyset$
    for each $i = 1 \dots m$ **do**
      if $y_i == k$ **then**
        $\tilde{S} \leftarrow (\mathbf{x}_i, +1)$
      else if $y_i == \ell$ **then**
        $\tilde{S} \leftarrow (\mathbf{x}_i, -1)$
      end if
    end for each
    Learn a classifier $h_{k\ell} : \mathcal{X} \to \mathbb{R}$ on $\tilde{S}$;
  end for each
end for each
The final classifier: $\forall \mathbf{x} \in \mathcal{X}, f(\mathbf{x}) = \underset{y' \in \mathcal{Y}, y' \neq y}{\operatorname{argmax}} \left| \{y \mid f_{yy'}(\mathbf{x}) = +1\} \right|$

where, $\forall \mathbf{x} \in \mathcal{X}, \forall (y, y') \in \mathcal{Y}^2, y \neq y', f_{yy'}(\mathbf{x}) = \begin{cases} \operatorname{sgn}\left(h_{yy'}(\mathbf{x})\right), & \text{if } y < y' \\ -f_{y'y}(\mathbf{x}), & \text{if } y' < y \end{cases}$

Combined approach - OvO



Combined approach - ECOC

This technique is composed of three steps :

- Each class $k \in \{1, \ldots, K\}$ is first coded (or represented) by a *code word* which is generally a binary vector of length n, $\mathfrak{D}_k \in \{-1, +1\}^n$,
- With the resulting matrix of codes $\mathfrak{D} \in \{-1, +1\}^{K \times n}$, n binary classifiers $(f_j)_{j=1}^n$ are learned after creating n training sets $\tilde{S}_j$ from the initial training set S :

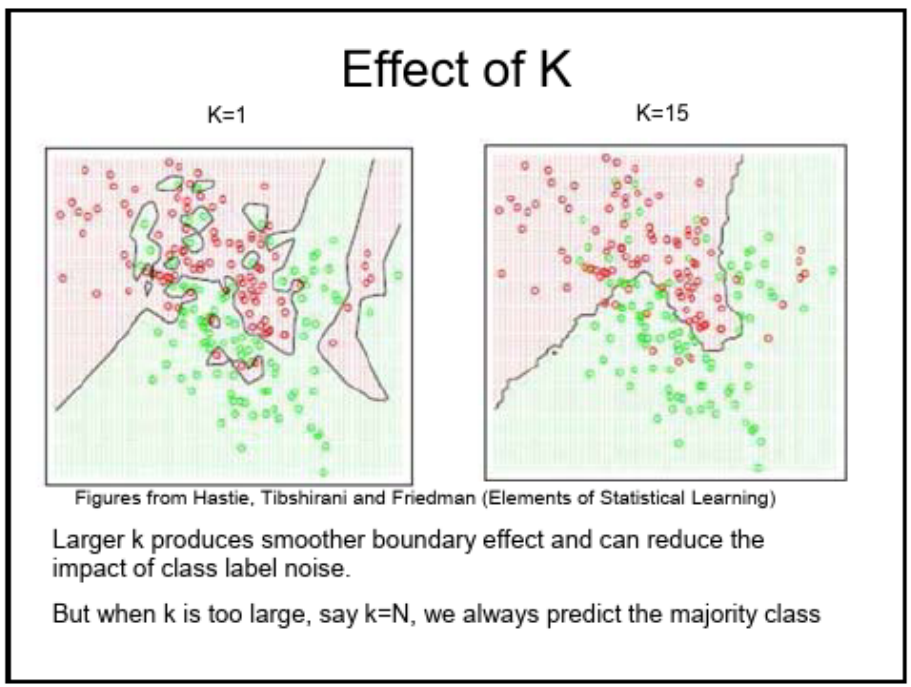
  $\forall (\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}, \forall j \in \{1, \ldots, n\},$ the associated code is $(\mathbf{x}, \mathfrak{D}_y(j))$

- To predict the class of an example $\mathbf{x}$, let $\mathbf{f}(\mathbf{x})$ denote the vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$, the associated class is the one having the lowest Hamming distance with the line vectors of $\mathfrak{D}$:

$$\forall \mathbf{x} \in \mathcal{X}, y^* = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmin}} \frac{1}{2} \sum_{j=1}^{n} (1 - \operatorname{sgn}(\mathfrak{D}_k(j) f_j(\mathbf{x})))$$

Uncombined approach - K-NN

- The K-Nearest Neighbors algorithm is a non-parametric method used for classification,
- The input consists of the K closest training examples in the characteristics space.
- For each observation $\mathbf{x} \in \mathcal{X}$ the class membership is decided by a majority vote of its neighbours.



Uncombined approach - Generative models

- Estimate $p(y)$ and $p(\mathbf{x} \mid y)$ by maximizing the complete log-likelihood,
- For prediction, use the Bayes rule $p(y \mid \mathbf{x}) \propto p(y) \times p(\mathbf{x} \mid y)$
- Affect an observation $\mathbf{x}$ to $y^* = \text{argmax}_y \, p(y \mid \mathbf{x})$

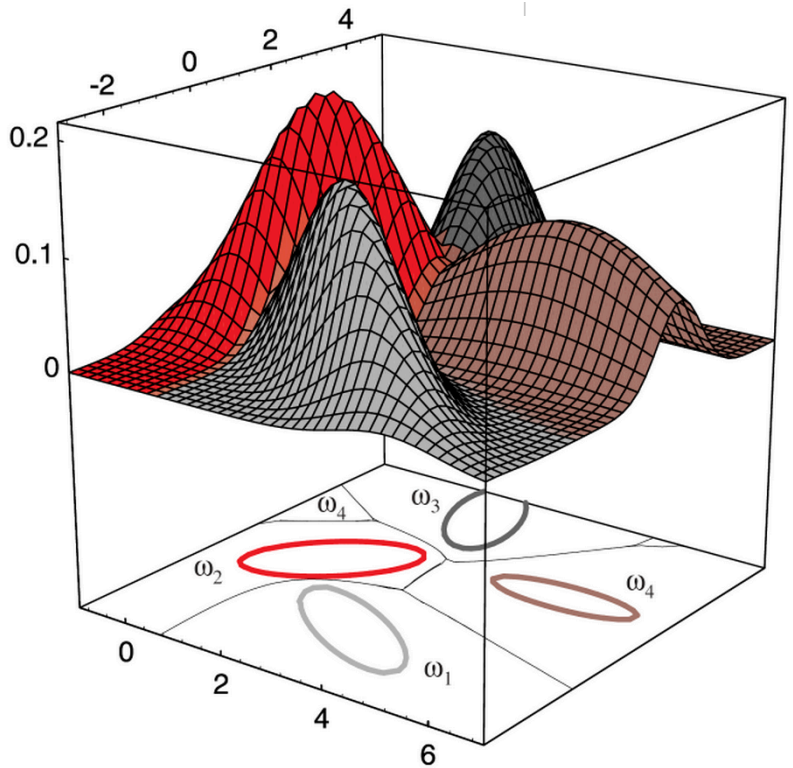
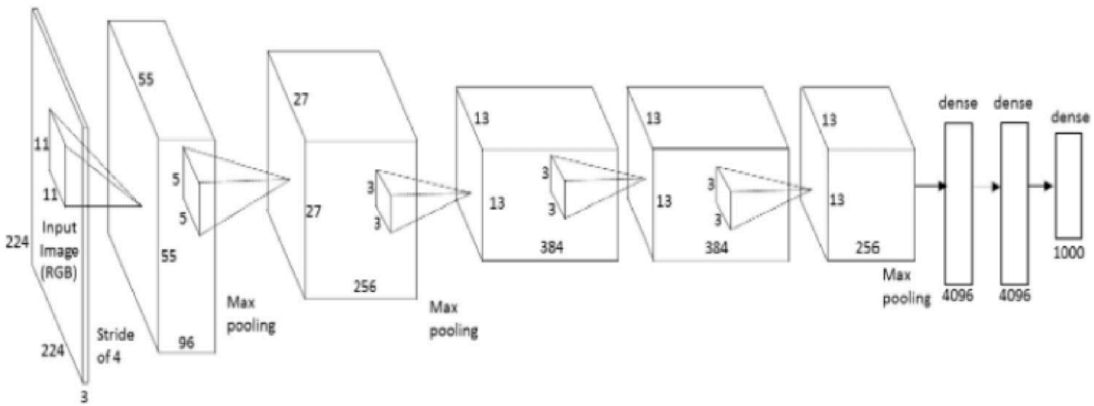


Figure from Duda, Hart and Stork (Pattern Classification)

Rebirth of Neural Networks

Deep architecture for ImageNet [Krizhevsky, Sutskever, Hinton, 2012]



- 60 million parameters, using 2 GPU – 6 days

A new fashion in image processing

2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

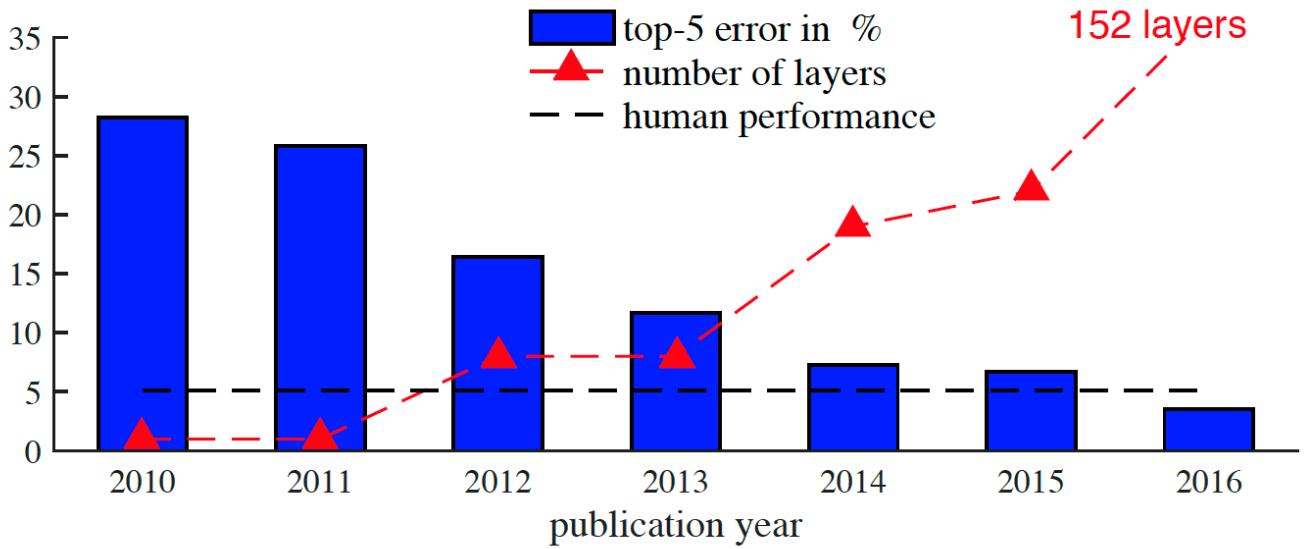
2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

shallow approaches

deep learning

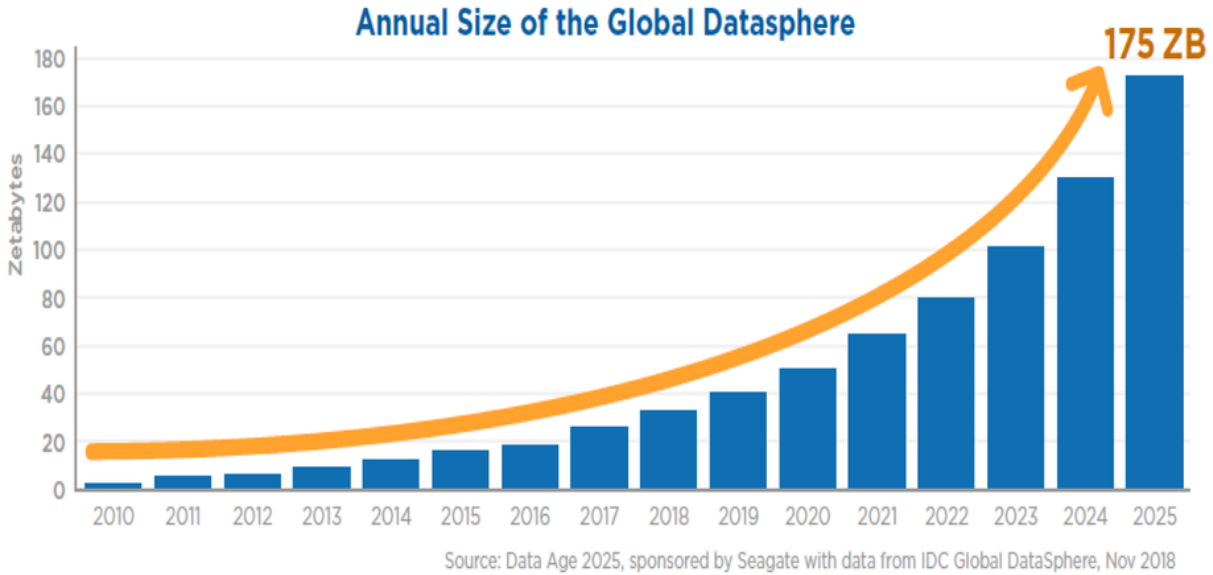
ImageNet Results



- 2012 AlexNet
- 2013 ZFNet
- 2014 VGG
- 2015 GoogLeNet / Inception
- 2016 Residual Network

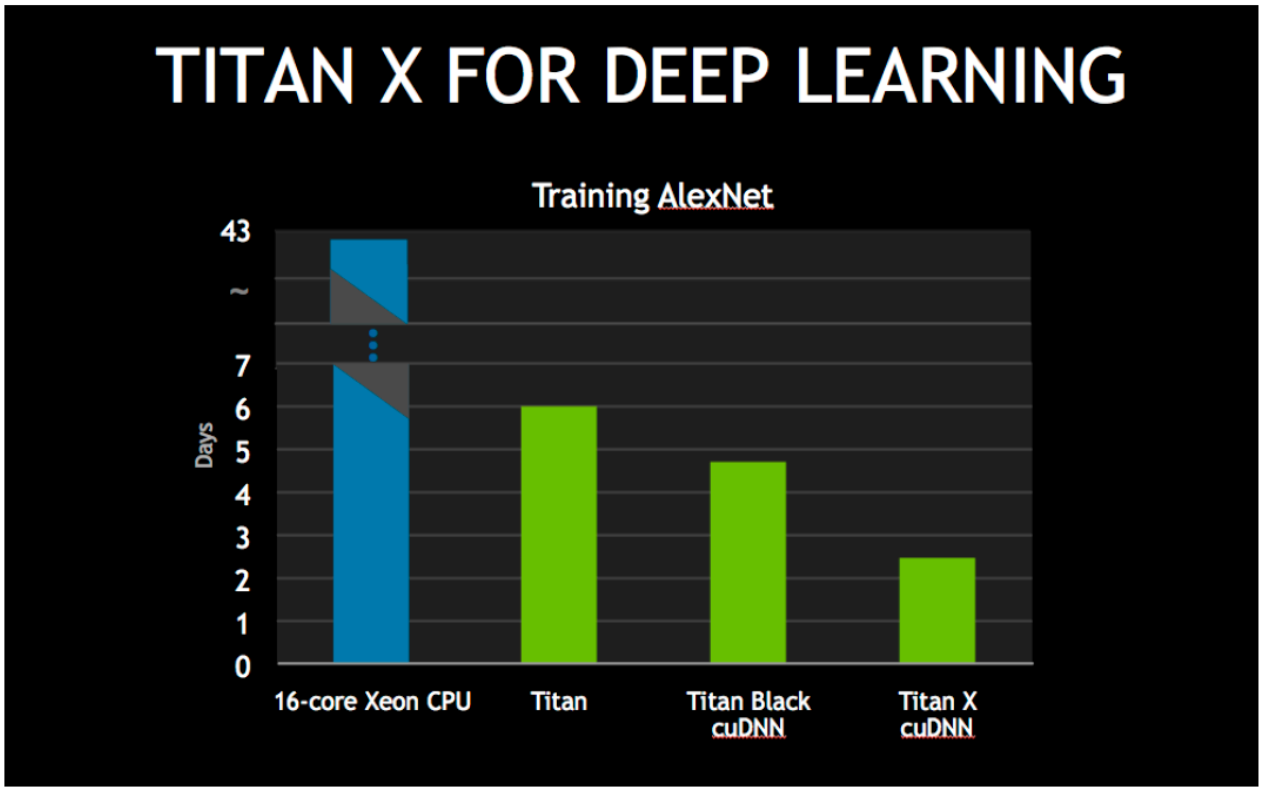
What's new in Deep Learning?

1. ***A lot of data (Big Data phenomena),***



What's new in Deep Learning?

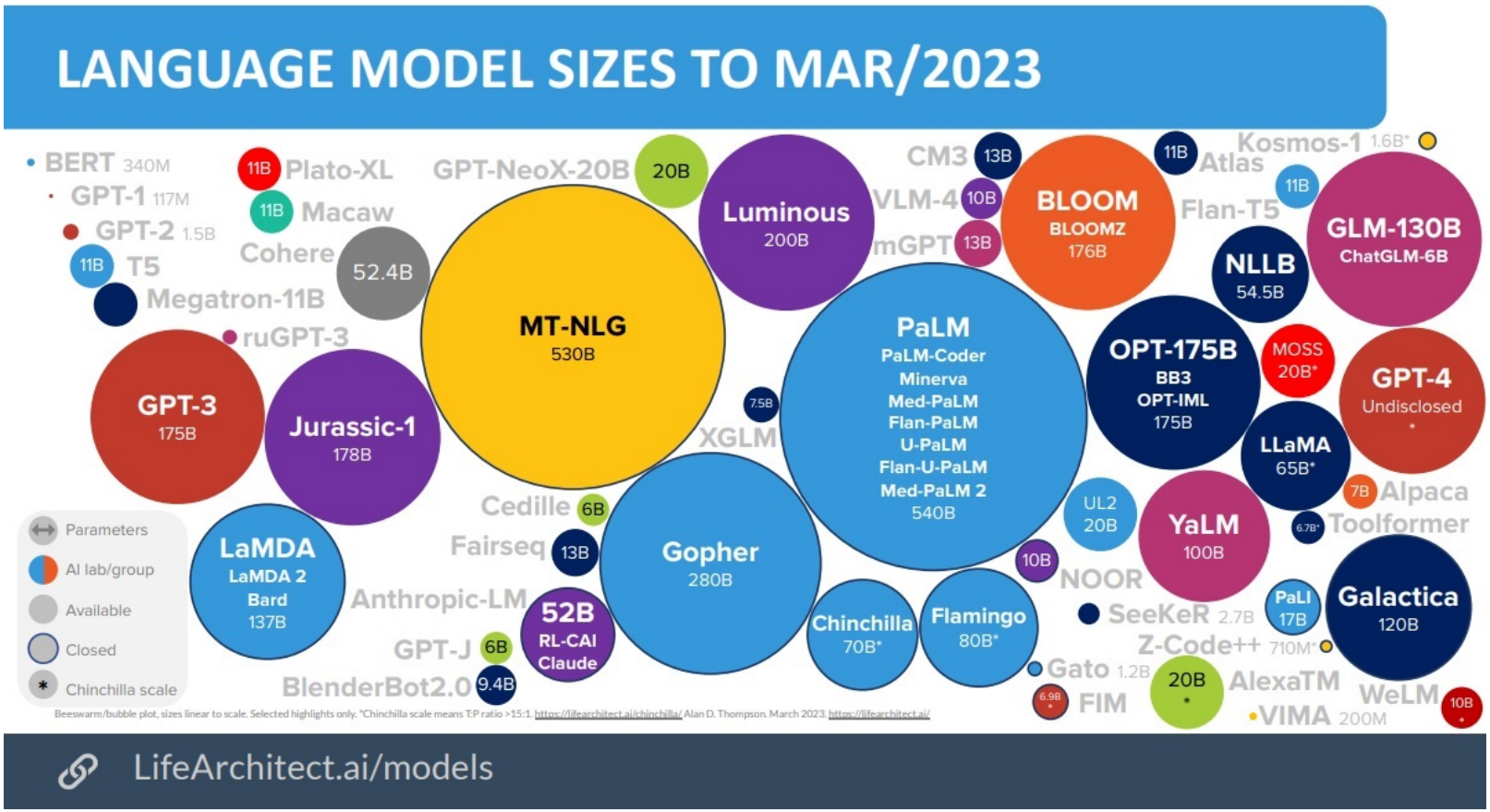
2. *Big computers: GPU needed*



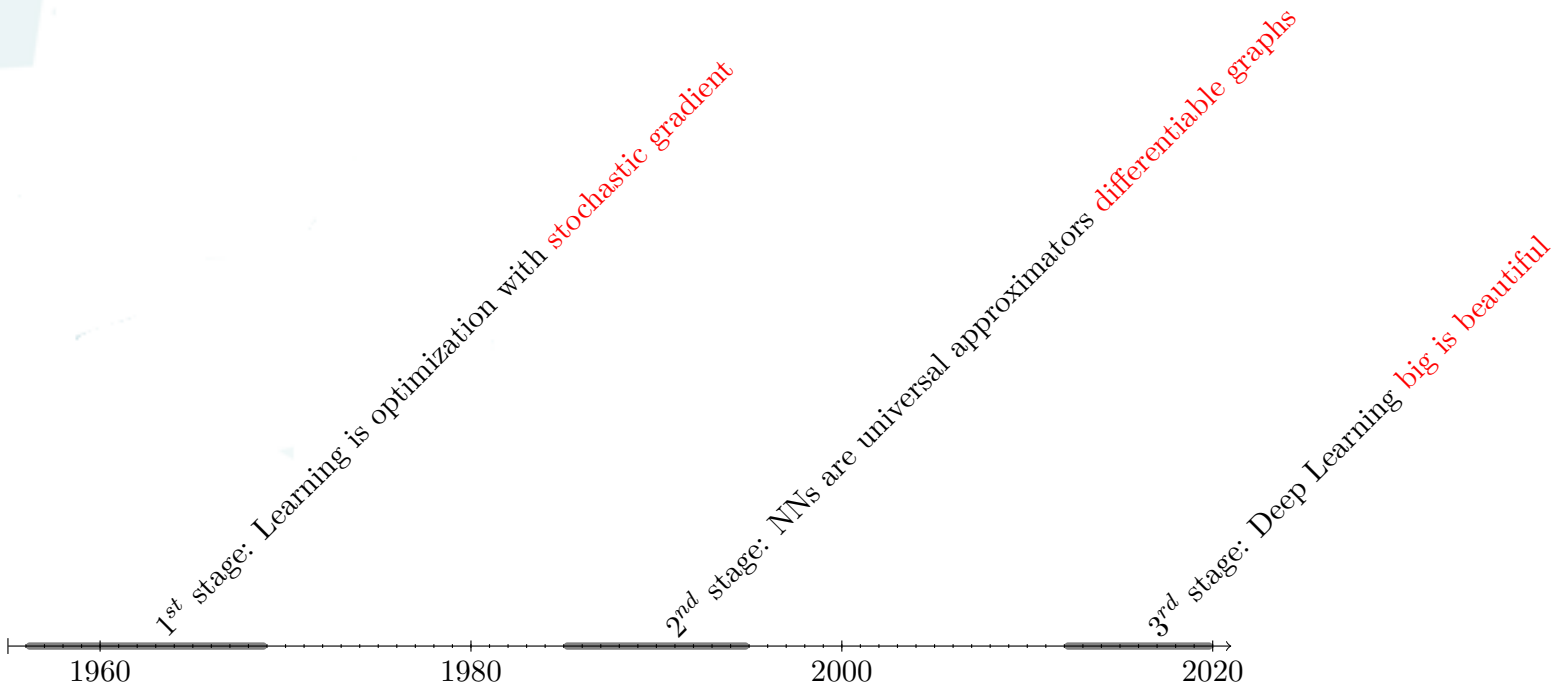
Now 2 hours with Nvidia DGX-1, and enough Memory

What's new in Deep Learning?

3. *Big architectures*



The deep learning time line



- Open issues
  - Fairness and explainability
  - Do more with less: green learning
  - Neural Architecture Search
  - Theory needed

References

Massih-Reza Amini
Machine Learning, de la théorie à la pratique
éditions Eyrolles, 2^{nd} edition, 2020.
Translated in Chinese :
机器学习：理论、实践与提高
iTuring edition, 2018

B.E. Boser, I.M. Guyon, V.N. Vapnik
A Training Algorithm for Optimal Margin Classifiers
Proceedings of the 5^{th} annual workshop on Computational Learning Theory
1992

D.R. Cox
The regression analysis of binary sequences (with discussion)".
Journal of Royal Statistics Soc B. 20 (2): 215–242.
1958

R.O. Duda, P.E. Hart, D.G. Stork
Pattern Classification
2000.

W. Hoeffding
Probability inequalities for sums of bounded random variables
Journal of the American Statistical Association, 58:13–30,
1963.

References

C. McDiarmid
On the method of bounded differences
Surveys in combinatorics, 141:148–188,
1989.

V. Koltchinskii
Rademacher penalties and structural risk minimization
IEEE Transactions on Information Theory, 47(5):1902–1914,
2001.

Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalker
Foundations of Machine Learning
2012.

A.B. Novikoff
On convergence proofs on perceptrons.
Symposium on the Mathematical Theory of Automata, 12: 615–622.
1962

F. Rosenblatt
The perceptron: A probabilistic model for information storage and organization in the brain.
Psychological Review, 65: 386–408.
1958

References

D. E. Rumelhart, G. E. Hinton and R. Williams
Learning internal representations by error propagation.
Parallel Distributed Processing: Explorations in the Microstructure of Cognition,
1986

S. Shalev-Shwartz, Y. Singer, N. Srebro and A. Cotter.
Primal Estimated sub-Gradient SOlver for SVM (Pegasos).
Mathematical Programming, 127(1):3–30, 2011

R.E. Schapire
Theoretical views of boosting and applications.
In Proceedings of the 10th International Conference on Algorithmic Learning Theory, pages 13–25.
1999

G. Widrow and M. Hoff
Adaptive switching circuits.
Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, 4: 96–104, 1960.

V. Vapnik.
The nature of statistical learning theory.
Springer, Verlag, 1998.