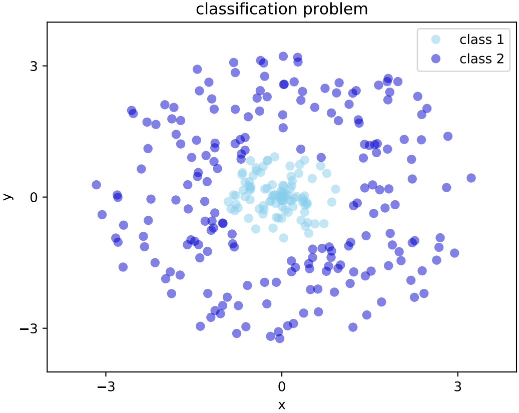
Fondamentaux théoriques du machine learning



Summary / updates

- Updates
- Kernel methods
- Support vector machines
- Decision trees

Optimization of neural networks

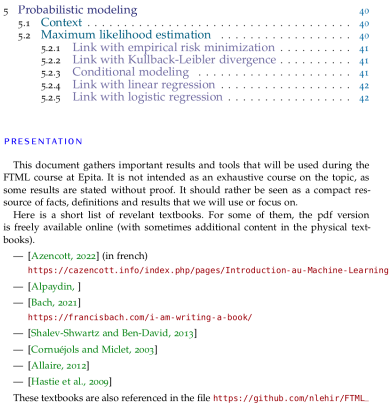
- Backpropagation
- Chain rule
- Difficulties of optimizing neural networks
- Specific methods for neural networks

Statistical learning

- Bounding the estimation error
- Interpolation regime and double descent

Updates in lecture notes.pdf

Added books and clarifications (still a work in progress, feedback welcome).



5 Probabilistic modeling 40
5.1 Context 40
5.2 Maximum likelihood estimation 40
5.2.1 Link with empirical risk minimization 41
5.2.2 Link with Kullback-Leibler divergence 41
5.2.3 Conditional modeling 41
5.2.4 Link with linear regression 42
5.2.5 Link with logistic regression 42

PRESENTATION

This document gathers important results and tools that will be used during the FTML course at Epita. It is not intended as an exhaustive course on the topic, as some results are stated without proof. It should rather be seen as a compact ressource of facts, definitions and results that we will use or focus on.

Here is a short list of revelant textbooks. For some of them, the pdf version is freely available online (with sometimes additional content in the physical textbooks).

— [Azencott, 2022] (in french)
https://cazencott.info/index.php/pages/Introduction-au-Machine-Learning
— [Alpaydin,]
— [Bach, 2021]
https://francisbach.com/i-am-writing-a-book/
— [Shalev-Shwartz and Ben-David, 2013]
— [Cornuéjols and Miclet, 2003]
— [Allaire, 2012]
— [Hastie et al., 2009]

These textbooks are also referenced in the file https://github.com/nlehir/FTML_

Project : exercice 1

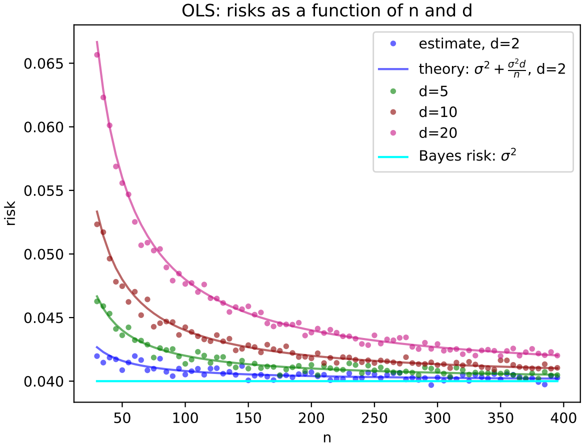
In exercice 1, you need to perform statistical estimation of the generalization error for an estimator differant than the Bayes estimator.

- generate test data
- average the errors on the test data

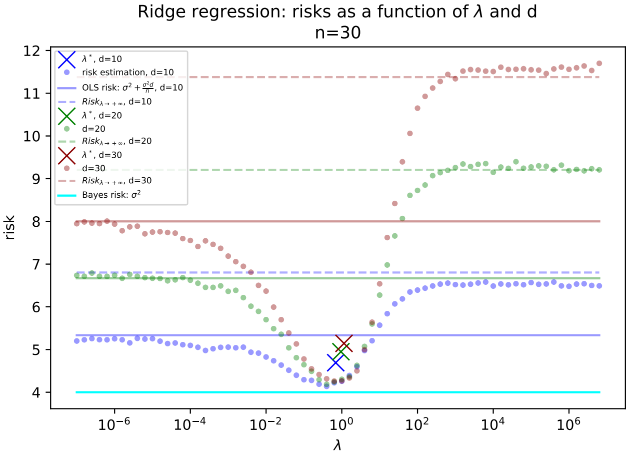
Thanks to the law of large numbers, this statistical average converges to the generalization error.
Note that depending on your setup, it might be possible to compute the generalization in **closed-form** (for any estimator). This was the case for some of the classification settings we used during exercices. Don't hesitate to do it and check consistency between the closed form expression and the simulation.

Project : exercice 1

We did this in the first practical sessions :



Project : exercice 1



Project : exercice 1

You may use a simple setting for this exercice (simple multi-class classification, simple regression, etc).

Project : exercice 3

Added clarifications on the statistical setting / definition of the **fixed design risk**.

Project

Do you have other questions on the project ?

Kernel methods

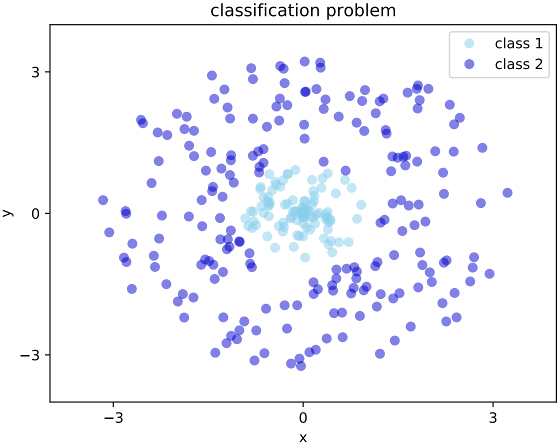
Replace inputs $x \in \mathcal{X}$ by a function $\phi(x) \in \mathcal{H}$, with $\mathcal{H}$ a $\mathbb{R}$-Hilbert space. We then perform linear predictions on $\phi(x)$. This means that estimators have the form :

$$f(x) = \langle \theta, \phi(x) \rangle_{\mathcal{H}} \tag{1}$$

Feature maps

- $\mathcal{X}$ need not be a vector space.
- $\phi(x)$ can provide more useful **representation** of the input for the considered problem (classification, regression).
- The prediction function is then allowed to depend **non-linearly** on x.

Nonlinear data separation



Representer theorem

The loss

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, \langle\theta, \phi(x_i))\rangle + \lambda||\theta||^2 \tag{2}$$

is minimized for $\hat{\theta} \in Vect(\{\phi(x_i)\}$.

$$\hat{\theta} = \sum_{i=1}^{n} \alpha_i \phi(x_i), \alpha \in \mathbb{R}^n \tag{3}$$

Alternate minimization problem

$$
\begin{aligned}
&\inf_{\theta\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} l(y_i, \langle\theta, \phi(x_i)\rangle + \lambda||\theta||^2 \\
= &\inf_{\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^{n} l(y_i, (K\alpha)_i) + \lambda\alpha^T K\alpha
\end{aligned}
\tag{4}
$$

It might be easier to optimize in $\mathbb{R}^n$ than in $\mathcal{H}$, especially if $\mathcal{H}$ is infinite dimensional.

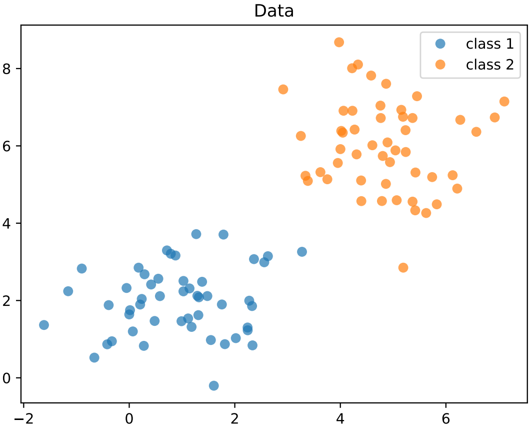


Figure – Linearly separable data

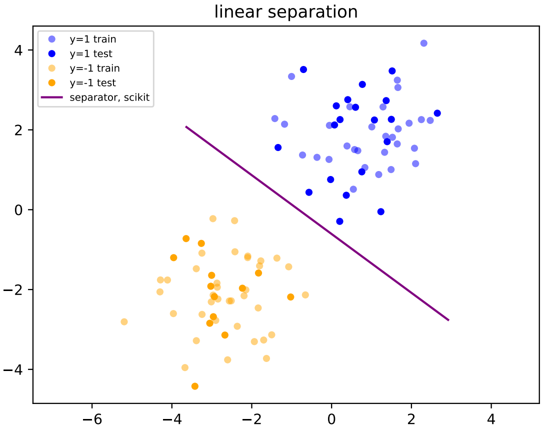


Figure – Linear separator

Linear separator

- $\mathcal{X} = \mathbb{R}^d$
- $\mathcal{Y} = \{-1, 1\}$

Equation of a linear separator

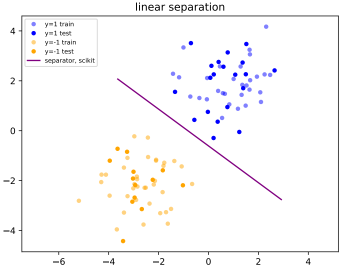
$$\langle w, x \rangle + b = 0 \tag{5}$$

- $w \in \mathbb{R}^d$
- $x \in \mathbb{R}^d$
- $b \in \mathbb{R}$

Notation :

$$h_{w,b}(x) = \langle w, x \rangle + b \tag{6}$$

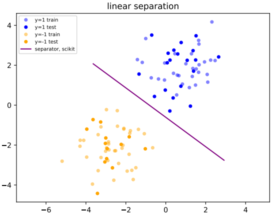
We first consider a linearly separable situation.



We note $h_{w,b}(x) = \langle w, x \rangle + b$. We look for separators that satisfies :

- $\forall x_i$ such that $y_i = 1$, $h_{w,b}(x) \geq 0$
- $\forall x_i$ such that $y_i = -1$, $h_{w,b}(x) \leq 0$

Support vectors



The **support vectors** are the vectors such that $|h_{w,b}(x)|$ is minimal among the dataset.

- the margin M is the distance from H to these vectors.
- if H is the optimal separator, there has to be a vector x_- and x_+ on each side, such that

$$M = d(x_-, H) = d(x_+, H) \tag{7}$$

Optimization problem (separable case)

The margin is proportional to $\frac{1}{||w||}$.
Optimization problem :

$$\arg\min_{w,b} \frac{1}{2}\langle w, w\rangle \tag{8}$$

subject to :

$$\forall i \in [1, n], y_i(\langle w, x_i\rangle + b) \geq 1 \tag{9}$$

Optimization problem (general case)

In the general case, the optimization problem is :

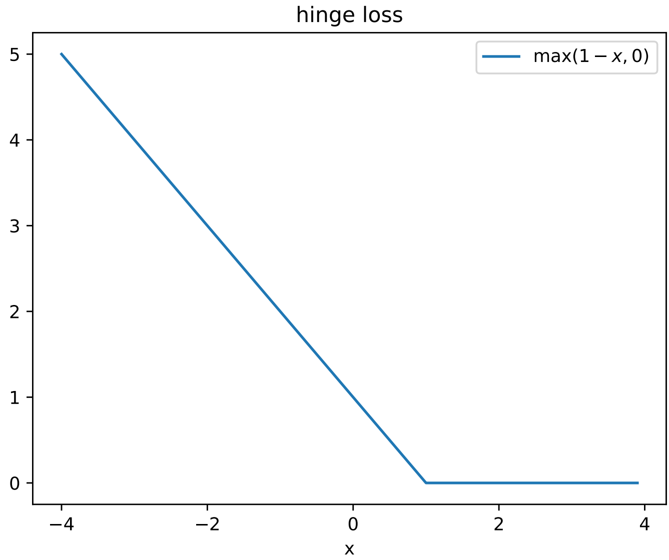
$$\underset{w,b,\xi}{\arg\min}\ \frac{1}{2}\langle w, w\rangle + C\sum_{i=1}^{n}\xi_i \tag{10}$$

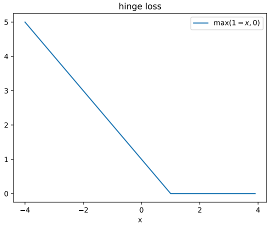
subject to :

$$\forall i \in [1, n], y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i \tag{11}$$

and

$$\forall i \in [1, n], \xi_i \geq 0 \tag{12}$$


hinge loss
max(1 − x, 0)
5
4
3
2
1
0
−4
−2
0
2
4
x



- estimation : $h(x) = \langle w, x \rangle + b$
- label : $y \in \{-1, 1\}$

Hinge loss :

$$L_{\text{hinge}}(h(x), y) = \max(0, 1 - yh(x)) \tag{13}$$

Problem reformulation

$$\arg\min_{w,b} \frac{1}{2}\langle w, w\rangle + C\sum_{i=1}^{n} max\left(0, 1 - y_i(\langle w, x_i\rangle + b)\right) \tag{14}$$

or equivalently

$$\arg\min_{w,b} \frac{1}{2}\langle w, w\rangle + C\sum_{i=1}^{n} L_{\text{hinge}}(h(x_i)), y_i) \tag{15}$$

Margin maximisation is an ERM problem with a $L2$ regularization and hinge loss.

Duality

An consequence of the lagrangian resolution of the optimization problem is that there exists α such that

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{16}$$

Hence

$$h_{w,b}(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + b \tag{17}$$

Only the inner products $\langle x_i, x_j \rangle$ are involved in the estimator. Similarly, the dual problem formulation also only uses the inner products. This motivates the use of kernels.

Gaussian kernel (RBF)

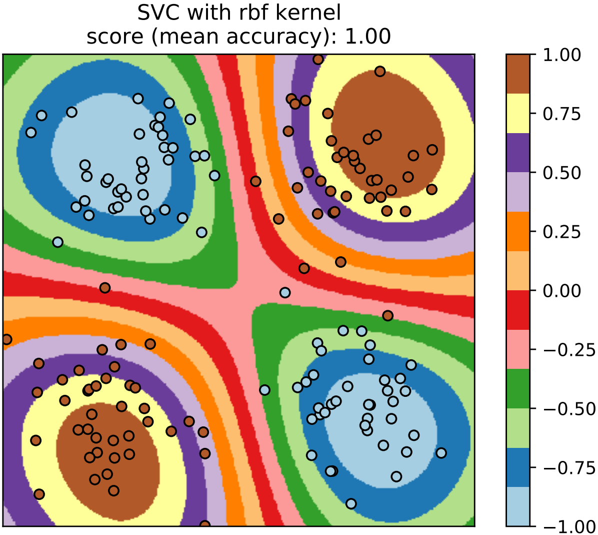
$$k(x, x') = \exp\Big(- \gamma ||x - x'||^2 \Big) \tag{18}$$

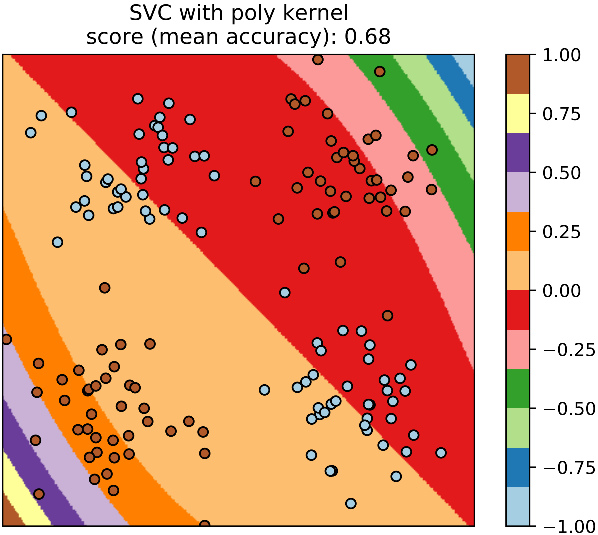
With $k(x, x') = \langle \phi(x), \phi(x') \rangle$, $\phi(x) \in \mathcal{H}$

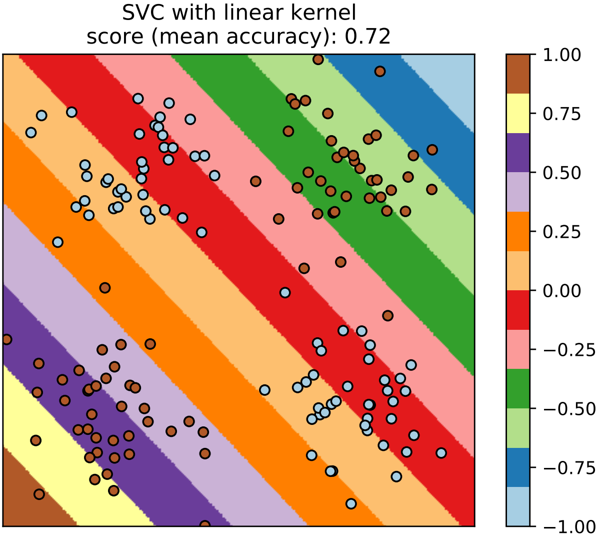
- $\mathcal{H}$ is of infinite dimension.
- γ is a parameter that should be carefully tuned.

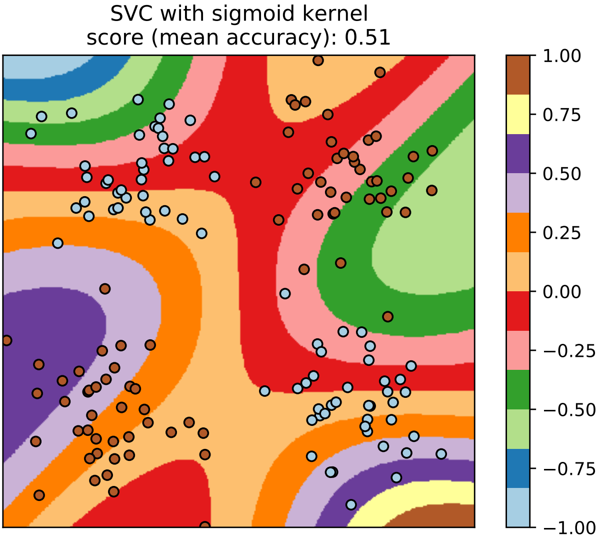
Linear kernel

$$k(x, x') = \langle x, x' \rangle \tag{19}$$

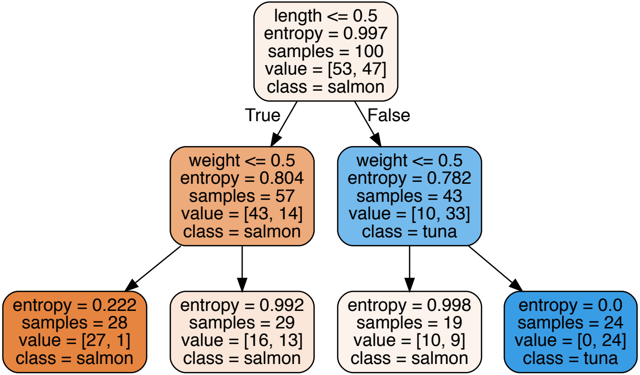

SVC with rbf kernel
score (mean accuracy): 1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


SVC with poly kernel
score (mean accuracy): 0.68
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


SVC with linear kernel
score (mean accuracy): 0.72
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


SVC with sigmoid kernel
score (mean accuracy): 0.51
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Decision tree



Homogeneity criterion for regression

In the case of regression, $\mathcal{Y} = \mathbb{R}$ and the heterogeneity $H(n)$ of node n is its empirical variance :

$$H(n) = \frac{1}{|n|} \sum_{i \in n} (y_i - \bar{y_n})^2 \qquad (20)$$

Homogeneity criterion for classification : entropy (information gain)

- We use the convention that $0 \log(0) = 0$
- p_n^l is the proportion of class l in node n.

The **entropy** writes :

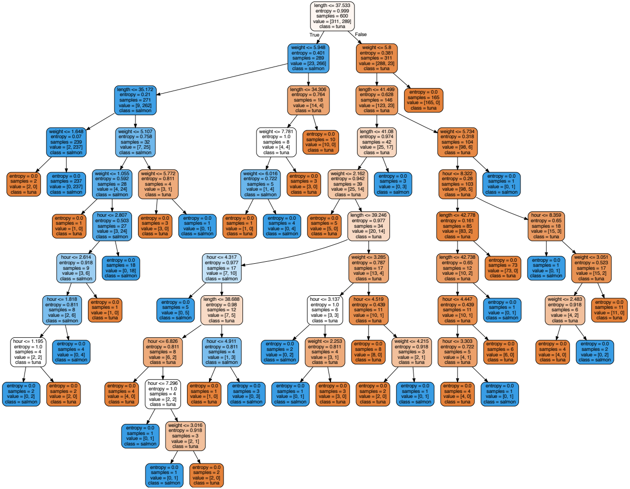
$$H(n) = -\sum_{l=1}^{L} p_n^l \log(p_n^l) \tag{21}$$

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

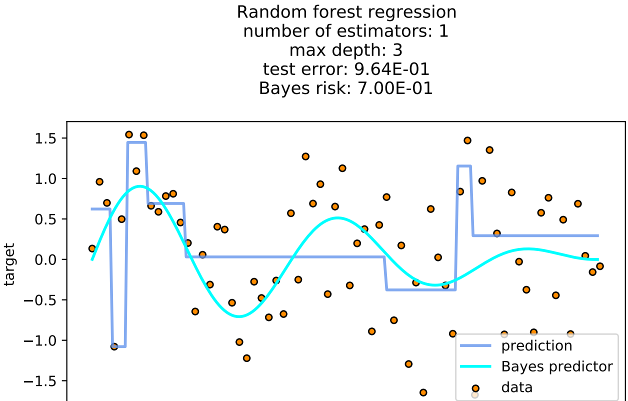
$$H(n) = \sum_{l=1}^{L} p_n^l (1 - p_n^l) \tag{22}$$

Overfitting


length <= 37.533 entropy = 0.999 samples = 600 value = [311, 289] class = tuna
True
False
weight <= 5.948 entropy = 0.401 samples = 289 value = [23, 266] class = salmon
weight <= 5.8 entropy = 0.381 samples = 311 value = [288, 23] class = tuna
length <= 35.172 entropy = 0.21 samples = 271 value = [9, 262] class = salmon
length <= 34.306 entropy = 0.764 samples = 18 value = [14, 4] class = tuna
length <= 41.499 entropy = 0.628 samples = 146 value = [123, 23] class = tuna
entropy = 0.0 samples = 165 value = [165, 0] class = tuna
weight <= 1.648 entropy = 0.07 samples = 239 value = [2, 237] class = salmon
weight <= 5.107 entropy = 0.758 samples = 32 value = [7, 25] class = salmon
weight <= 7.781 entropy = 1.0 samples = 8 value = [4, 4] class = tuna
entropy = 0.0 samples = 10 value = [10, 0] class = tuna
length <= 41.08 entropy = 0.974 samples = 42 value = [25, 17] class = tuna
weight <= 5.734 entropy = 0.318 samples = 104 value = [98, 6] class = tuna
entropy = 0.0 samples = 2 value = [2, 0] class = tuna
entropy = 0.0 samples = 237 value = [0, 237] class = salmon
weight <= 1.055 entropy = 0.592 samples = 28 value = [4, 24] class = salmon
weight <= 5.772 entropy = 0.811 samples = 4 value = [3, 1] class = tuna
weight <= 6.016 entropy = 0.722 samples = 5 value = [1, 4] class = salmon
entropy = 0.0 samples = 3 value = [3, 0] class = tuna
weight <= 2.162 entropy = 0.942 samples = 39 value = [25, 14] class = tuna
entropy = 0.0 samples = 3 value = [0, 3] class = salmon
hour <= 8.322 entropy = 0.28 samples = 103 value = [98, 5] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
entropy = 0.0 samples = 1 value = [1, 0] class = tuna
hour <= 2.807 entropy = 0.503 samples = 27 value = [3, 24] class = salmon
entropy = 0.0 samples = 3 value = [3, 0] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
entropy = 0.0 samples = 1 value = [1, 0] class = tuna
entropy = 0.0 samples = 4 value = [0, 4] class = salmon
entropy = 0.0 samples = 5 value = [5, 0] class = tuna
length <= 39.246 entropy = 0.977 samples = 34 value = [20, 14] class = tuna
length <= 42.778 entropy = 0.161 samples = 85 value = [83, 2] class = tuna
hour <= 8.359 entropy = 0.65 samples = 18 value = [15, 3] class = tuna
hour <= 2.614 entropy = 0.918 samples = 9 value = [3, 6] class = salmon
entropy = 0.0 samples = 18 value = [0, 18] class = salmon
hour <= 4.317 entropy = 0.977 samples = 17 value = [7, 10] class = salmon
weight <= 3.285 entropy = 0.787 samples = 17 value = [13, 4] class = tuna
length <= 42.738 entropy = 0.65 samples = 12 value = [10, 2] class = tuna
entropy = 0.0 samples = 73 value = [73, 0] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
weight <= 3.051 entropy = 0.523 samples = 17 value = [15, 2] class = tuna
hour <= 1.818 entropy = 0.811 samples = 8 value = [2, 6] class = salmon
entropy = 0.0 samples = 1 value = [1, 0] class = tuna
entropy = 0.0 samples = 5 value = [0, 5] class = salmon
length <= 38.688 entropy = 0.98 samples = 12 value = [7, 5] class = tuna
hour <= 3.137 entropy = 1.0 samples = 6 value = [3, 3] class = tuna
hour <= 4.519 entropy = 0.439 samples = 11 value = [10, 1] class = tuna
hour <= 4.447 entropy = 0.439 samples = 11 value = [10, 1] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
weight <= 2.483 entropy = 0.918 samples = 6 value = [4, 2] class = tuna
entropy = 0.0 samples = 11 value = [11, 0] class = tuna
hour <= 1.195 entropy = 1.0 samples = 4 value = [2, 2] class = tuna
entropy = 0.0 samples = 4 value = [0, 4] class = salmon
hour <= 6.826 entropy = 0.811 samples = 8 value = [6, 2] class = tuna
hour <= 4.911 entropy = 0.811 samples = 4 value = [1, 3] class = salmon
entropy = 0.0 samples = 2 value = [0, 2] class = salmon
weight <= 2.253 entropy = 0.811 samples = 4 value = [3, 1] class = tuna
entropy = 0.0 samples = 8 value = [8, 0] class = tuna
weight <= 4.215 entropy = 0.918 samples = 3 value = [2, 1] class = tuna
hour <= 3.303 entropy = 0.722 samples = 5 value = [4, 1] class = tuna
entropy = 0.0 samples = 6 value = [6, 0] class = tuna
entropy = 0.0 samples = 4 value = [4, 0] class = tuna
entropy = 0.0 samples = 2 value = [0, 2] class = salmon
entropy = 0.0 samples = 2 value = [0, 2] class = salmon
entropy = 0.0 samples = 2 value = [2, 0] class = tuna
entropy = 0.0 samples = 4 value = [4, 0] class = tuna
hour <= 7.296 entropy = 1.0 samples = 4 value = [2, 2] class = tuna
entropy = 0.0 samples = 1 value = [1, 0] class = tuna
entropy = 0.0 samples = 3 value = [0, 3] class = salmon
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
entropy = 0.0 samples = 3 value = [3, 0] class = tuna
entropy = 0.0 samples = 2 value = [2, 0] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
entropy = 0.0 samples = 4 value = [4, 0] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
weight <= 3.016 entropy = 0.918 samples = 3 value = [2, 1] class = tuna
entropy = 0.0 samples = 1 value = [0, 1] class = salmon
entropy = 0.0 samples = 2 value = [2, 0] class = tuna

Ensemble learning

- Bagging (bootstrapping + aggregating), Random forest (parallel learners)
- Boosting, gradient tree boosting (sequential learners)



Summary / updates

- Updates
- Kernel methods
- Support vector machines
- Decision trees

Optimization of neural networks

- Backpropagation
- Chain rule
- Difficulties of optimizing neural networks
- Specific methods for neural networks

Statistical learning

- Bounding the estimation error
- Interpolation regime and double descent

Learning representations / features

- $\mathcal{X} = \mathbb{R}^d$.
- $\mathcal{Y} = \mathbb{R}$.

A neural network learns a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^m$ **and** a linear regressor or classifier, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{23}$$

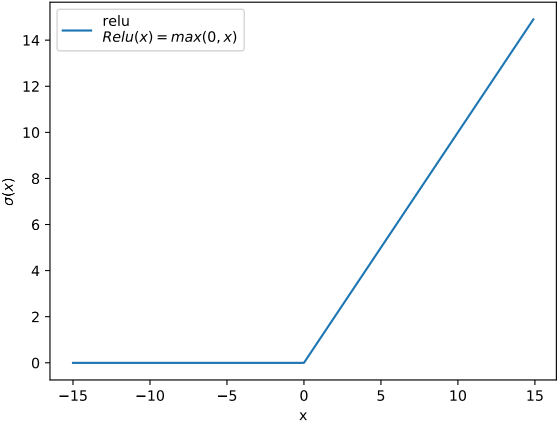
We can add a bias by adding a dimension to θ and adding a component with a 1 to each $\phi(x)$.
Remark : kernel methods use hardcoded features ϕ, that can be **implicit** (kernel trick).

Single layer neural network

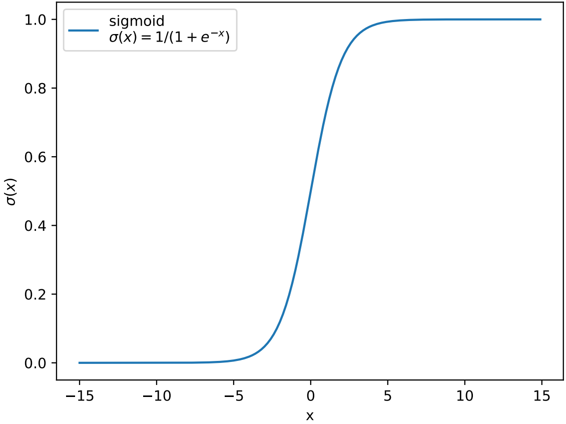
$$f(x) = \sum_{j=1}^{m} \theta_j \sigma(w_j^T x + b_j) \tag{24}$$

σ is an activation function (sigmoid, tanh, ReLu, etc).

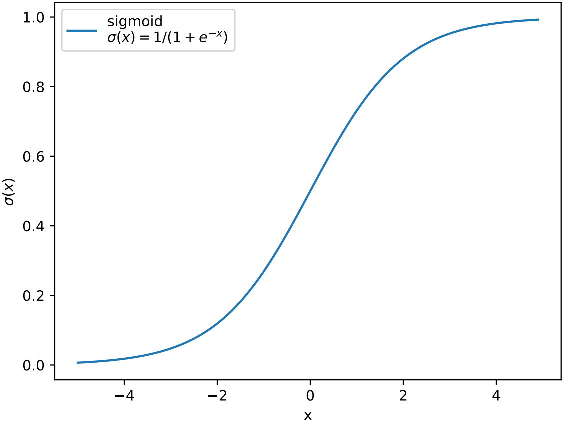
ReLU



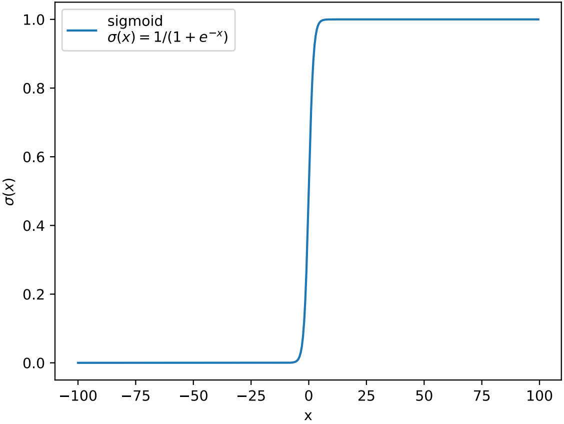
Sigmoid



Sigmoid



Sigmoid


sigmoid
$\sigma(x) = 1/(1+e^{-x})$
1.0
0.8
0.6
0.4
0.2
0.0
$\sigma(x)$
−100
−75
−50
−25
0
25
50
75
100
x

Chain rule

Backpropagation is the general optimization framework for neural networks. SGD and variants are used.
To understand backpropagation, the main tool is the **chain rule.**

Chain rule

- Simple neural network : $d = 1,\ m = 1,\ n = 1.$
- $w \in \mathbb{R},\ \theta \in \mathbb{R},\ b.$

$$f(x) = \theta\sigma(wx + b) \tag{25}$$

Loss function

$$L(\theta, w, b) = (f(x) - y)^2 \tag{26}$$

We can note $\hat{y} = f(x)$ and $h = \sigma(wx + b)$, then $L(\theta, w, b) = (\hat{y} - y)^2$.
Exercice 1 : Compute and express in terms of y, $\hat{y}$ and h :

- $\frac{\partial h}{\partial w}, \frac{\partial h}{\partial b}$
- $\frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}$

Chain rule

$$\frac{\partial h}{\partial w} = \sigma'(wx + b)x \tag{27}$$

$$\frac{\partial h}{\partial b} = \sigma'(wx + b) \tag{28}$$

$$\frac{\partial L}{\partial \theta} = 2(\hat{y} - y)\sigma(wx + b) \tag{29}$$

$$\frac{\partial L}{\partial w} = 2(\hat{y} - y)\theta\sigma'(wx + b)x \tag{30}$$

$$\frac{\partial L}{\partial b} = 2(\hat{y} - y)\theta\sigma'(wx + b) \tag{31}$$

Chain rule

$$\frac{\partial h}{\partial w} = \sigma'(wx + b)x \tag{32}$$

$$\frac{\partial h}{\partial b} = \sigma'(wx + b) \tag{33}$$

$$\frac{\partial L}{\partial \theta} = 2(\hat{y} - y)\sigma(wx + b) \tag{34}$$

$$\frac{\partial L}{\partial w} = 2(\hat{y} - y)\theta\frac{\partial h}{\partial w} \tag{35}$$

$$\frac{\partial L}{\partial b} = 2(\hat{y} - y)\theta\frac{\partial h}{\partial b} \tag{36}$$

- Simple neural network : $d = 2$, $m = 2$, $n = 1$.
- $w = (w_1, w_2) \in \mathbb{R}^{2,2}$, $\theta \in \mathbb{R}^2$, $b \in \mathbb{R}^2$.

$$\begin{aligned} f(x) &= \theta_1\sigma(w_1^T x + b_1) + \theta_2\sigma(w_2^T x + b_2) \\ &= \langle \theta, h \rangle \\ &= \hat{y} \end{aligned} \tag{37}$$

Loss function

$$\begin{aligned} L(\theta, w, b) &= (f(x) - y)^2 \\ &= (\hat{y} - y)^2 \end{aligned} \tag{38}$$

Exercice 2 : Compute :

- $\nabla_\theta f(w, \theta, b)$
- $\nabla_{w_1} f(w, \theta, b)$, $\nabla_{w_2} f(w, \theta, b)$
- $\frac{\partial f}{\partial b_1}(w, \theta, b)$, $\frac{\partial f}{\partial b_2}(w, \theta, b)$

Gradients

$$\nabla_\theta f(w, \theta, b) = h \tag{39}$$

$$\begin{aligned} \nabla_\theta L(w, \theta, b) &= 2(\hat{y} - y)\nabla_\theta(\theta \mapsto \langle \theta, h \rangle)(w, \theta, b) \\ &= 2(\hat{y} - y)h \end{aligned} \tag{40}$$

Gradients

$$\nabla_{w_1} f(w, \theta, b) = \theta_1 \sigma'(w_1^T x + b_1) x \tag{41}$$

$$\nabla_{w_2} f(w, \theta, b) = \theta_2 \sigma'(w_2^T x + b_2) x \tag{42}$$

Gradients

$$\frac{\partial f}{\partial b_1}(w, \theta, b) = \theta_1 \sigma'(w_1^T x + b_1) \tag{43}$$

$$\frac{\partial f}{\partial b_2}(w, \theta, b) = \theta_2 \sigma'(w_2^T x + b_2) \tag{44}$$

General one hidden layer network

- $x \in \mathbb{R}^d$
- $h \in \mathbb{R}^m$
- $y \in \mathbb{R}$

- Weights : $W \in \mathbb{R}^{d,m}$.
- Bias : $B \in \mathbb{R}^m$.
- Output classifier : $\theta \in \mathbb{R}^m$

- $h = \sigma(W^T x + B)$
- $y = \sigma(\langle \theta, h \rangle)$

General one hidden layer network

- $x \in \mathbb{R}^d$, $h \in \mathbb{R}^m$, $y \in \mathbb{R}$
- Weights : $W \in \mathbb{R}^{d,m}$.
- Bias : $B \in \mathbb{R}^m$.
- Output classifier : $\theta \in \mathbb{R}^m$
- $h = \sigma(W^T x + B)$
- $y = \sigma(\langle \theta, h \rangle)$

Exercice 3 : Compute the gradient

$$\frac{\partial h_i}{\partial W_i} \tag{45}$$

$$\frac{\partial h_i}{\partial W_i} = \sigma'({W_i}^T x + B_i)x \tag{46}$$

General one hidden layer network

- $x \in \mathbb{R}^d$, $h \in \mathbb{R}^m$, $y \in \mathbb{R}$
- Weights : $W \in \mathbb{R}^{d,m}$.
- Bias : $B \in \mathbb{R}^m$.
- Output classifier : $\theta \in \mathbb{R}^m$
- $h = \sigma(W^T x + B)$
- $y = \sigma(\langle \theta, h \rangle)$

Exercice 4 : Compute

$$\frac{\partial h_i}{\partial B_i} \tag{47}$$

$$\frac{\partial h_i}{\partial B_i} = \sigma'({W_i}^T x + B_i) \tag{48}$$

General one hidden layer network

- $x \in \mathbb{R}^d$, $h \in \mathbb{R}^m$, $y \in \mathbb{R}$

- Weights : $W \in \mathbb{R}^{d,m}$.
- Bias : $B \in \mathbb{R}^m$.
- Output classifier : $\theta \in \mathbb{R}^m$

- $h = \sigma(W^T x + B)$
- $y = \sigma(\langle \theta, h \rangle)$

Exercice 5 : Compute the gradient

$$\frac{\partial y}{\partial \theta} \tag{49}$$

$$\frac{\partial y}{\partial \theta} = \sigma'(\langle \theta, h \rangle) h \tag{50}$$

General one hidden layer network

- $x \in \mathbb{R}^d$, $h \in \mathbb{R}^m$, $y \in \mathbb{R}$

- Weights : $W \in \mathbb{R}^{d,m}$.
- Bias : $B \in \mathbb{R}^m$.
- Output classifier : $\theta \in \mathbb{R}^m$

- $h = \sigma(W^T x + B)$
- $y = \sigma(\langle \theta, h \rangle)$

Exercice 6 : Compute the gradient

$$\frac{\partial y}{\partial W_i} \tag{51}$$

$$\begin{aligned}\frac{\partial y}{\partial W_i} &= \sigma'(\langle\theta, h\rangle)\theta_i \frac{\partial h_i}{\partial W_i} \\ &= \sigma'(\langle\theta, h\rangle)\theta_i \sigma'({W_i}^T x + B_i)x\end{aligned} \tag{52}$$

General one hidden layer network

- $x \in \mathbb{R}^d$, $h \in \mathbb{R}^m$, $y \in \mathbb{R}$
- Weights : $W \in \mathbb{R}^{d,m}$.
- Bias : $B \in \mathbb{R}^m$.
- Output classifier : $\theta \in \mathbb{R}^m$
- $h = \sigma(W^T x + B)$
- $y = \sigma(\langle \theta, h \rangle)$

Exercice 7 : Compute

$$\frac{\partial y}{\partial B_i} \tag{53}$$

$$\begin{aligned}\frac{\partial y}{\partial B_i} &= \sigma'(\langle \theta, h \rangle)\theta_i \frac{\partial h_i}{\partial B_i} \\ &= \sigma'(\langle \theta, h \rangle)\theta_i \sigma'(B_i^T x + B_i)\end{aligned} \tag{54}$$

Automatic differentiation

When working with neural networks, most used libraries implement automatic differentiation.

- tensorflow
- pytorch (autograd)

Optimizing neural networks

Optimizing neural networks comes with specific difficulties.

Optimizing neural networks

Optimizing neural networks comes with specific difficulties.

- the problem is non-convex
- there is often a large number of parameters (optimization in a high dimensional space)
- specific problems due to depth (vanishing gradients, see below)

Depth of neural networks

Deep neural networks (several tenth or even hundreds of hidden layers) achieve state-of-the-art performance in several tasks :

- Natural language processing (NLP)
- Computer vision
- See first class for other examples.

Non-convexity

We now that

- If f is increasing and convex and g is convex, then $f \circ g$ is convex.
- Is f in convex and g is linear, then $f \circ g$ is convex.

With neural networks, we are in neither of these cases, as the activations σ are non linear.

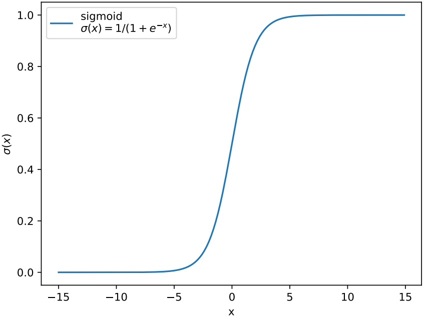
Non-convexity

We now that

- If f is increasing and convex and g is convex, then $f \circ g$ is convex.
- Is f in convex and g is linear, then $f \circ g$ is convex.

With neural networks, we are in neither of these cases, as the activations σ are non linear.
Hence **the objective function is non-convex**, and it remains difficult to understand why gradient based methods often perform well in practice

Vanishing gradients



Exercice 8 :

What is the maximum value of $|\sigma'(z)|$?

Exponentially decreasing gradients

- At each layer, the gradients are multiplied by a term of the form $\sigma'(u)$. Using a large number of layers leads to gradient norms that decrease rapidly when we move away from the output layer.
- This slows training down and caused deep learning to plateau for some years.
- Several initializations were necessary in order to obtain convergence, the result was unstable.

ReLU

The usage of ReLU solved this problem.
Other activation functions :
https://dashee87.github.io/deep%20learning/visualising-activation-functions-in-neural-networks/

SGD variants for neural networks

Several specific variations of SGD are commonly used for deep learning.
`https://pytorch.org/docs/stable/optim.html`

Specific methods for deep learning

Architectures :

- Convotutional networks
- Residual neural network (ResNet)

Optimization / regularization :

- dropout
- batch normalisation

Probably studied during practical sessions.

Statistical learning

- We come back to the statistical analysis of supervised learning.
- More precisely, to that of empifical risk minimization.

Reminder on risks

Let l be a loss. Generalization error :

$$R(f) = E_{(X,Y) \sim \rho}[l(Y, f(X))] \tag{55}$$

The **empirical risk (ER)** of an estimator f writes

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, f(x_i)) \tag{56}$$

We emphasize that the risks depends on the loss l.

Risk decomposition

- f^* : Bayes predictor
- F : Hypothesis space
- $\tilde{f}_n$: estimated predictor (hence in F).

$$E\left[R(\tilde{f}_n)\right] - R^* = \left(E\left[R(\tilde{f}_n)\right] - \inf_{f \in F} R(f)\right) + \left(\inf_{f \in F} R(f) - R^*\right) \quad (57)$$

Risk decomposition

- f^* : Bayes predictor
- F : Hypothesis space
- $\tilde{f}_n$: estimated predictor.

When doing empirical risk minimization, $\tilde{f}_n$ is obtained by minimization of the empirical risk.

However :

- we have seen that in many cases, finding the exact minimizer of the empirical risk might be computationnaly hard.
- also, a natural question is whether it is sufficient to have an **approximate minimizer** of the empirical risk, as the empirical risk is an **approximation** of the generalization error.

Optimization error

We introduce the **optimization error**. Let f_n be the exact minimizer of R_n, which we assume exists. The optimization error is defined as ([Bottou and Bousquet, 2009]) :

$$\epsilon_{opt} = E\left[R(\tilde{f}_n) - R(f_n)\right] \tag{58}$$

Risk decomposition

Now, we can write another risk decomposition

$$\begin{aligned} E\Big[R(\tilde{f}_n)\Big] - R^* = &\Big(E\Big[R(\tilde{f}_n)\Big] - E\Big[R(f_n)\Big]\Big. \\ &+ \Big(E\Big[R(f_n)\Big] - \inf_{f \in F} R(f)\Big) \\ &+ \Big(\inf_{f \in F} R(f) - R^*\Big) \end{aligned} \tag{59}$$

Risk decomposition

Optimization error : depends on D_n, F, $\tilde{f}_n$.

$$E\Big[R(\tilde{f}_n) - R(f_n)\Big] \tag{60}$$

Estimation error (variance term, fluctuation error, stochastic error) : depends on D_n, F, $\tilde{f}_n$.

$$E\Big[R(f_n)\Big] - \inf_{f \in F} R(f) \geq 0$$

Approximation error (bias term) : depends on f^* and F, not on $\tilde{f}_n$, D_n.

$$\inf_{f \in F} R(f) - R^* \geq 0$$

Bound on the estimation error

We will now focus on the estimation error

$$E\Big[R(f_n)\Big] - \inf_{f \in F} R(f) \geq 0$$

f_n is the empirical risk minimizer
We consider the best estimator in hypothesis space

$$f_a = \arg\min_{h \in F} R(h)$$

Let us show that

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{61}$$

Deterministic bound on the estimation error

$$f_a = \arg\min_{h \in F} R(h) \tag{62}$$

$$f_n = \arg\min_{h \in F} R_n(h) \tag{63}$$

$$\begin{aligned} R(f_n) - R(f_a) = &\big(R(f_n) - R_n(f_n)\big) \\ &+ \big(R_n(f_n) - R_n(f_a)\big) \\ &+ \big(R_n(f_a) - R(f_a)\big) \end{aligned} \tag{64}$$

Deterministic bound on the estimation error

$$f_a = \arg\min_{h \in F} R(h)$$

$$f_n = \arg\min_{h \in F} R_n(h) \tag{65}$$

$$\begin{aligned} R(f_n) - R(f_a) = &\big(R(f_n) - R_n(f_n)\big) \\ &+ \big(R_n(f_n) - R_n(f_a)\big) \\ &+ \big(R_n(f_a) - R(f_a)\big) \end{aligned} \tag{66}$$

But by definition f_n minimizes R_n, so $\big(R_n(f_n) - R_n(f_a)\big) \leq 0$.

Deterministic bound on the estimation error

$$f_a = \arg\min_{h \in F} R(h)$$

$$f_n = \arg\min_{h \in F} R_n(h) \tag{67}$$

$$\begin{aligned} R(f_n) - R(f_a) = {} & \big(R(f_n) - R_n(f_n)\big) \\ & + \big(R_n(f_n) - R_n(f_a)\big) \\ & + \big(R_n(f_a) - R(f_a)\big) \end{aligned} \tag{68}$$

But by definition f_n minimizes R_n, so $\big(R_n(f_n) - R_n(f_a)\big) \leq 0$.
Finally :

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{69}$$

Bound on the estimation error

If we are able to bound $\sup_{h \in F} |R(h) - R_n(h)|$, then we have a bound on the estimation error.

Bound on the estimation error

Theorem

Weak law of large numbers
Let $(X_i)_{i\in\mathbb{N}}$ be a sequence of i.i.d. variables that have a moment of order 2. We note m their expected value. Then

$$\forall \epsilon > 0, \lim_{n\to+\infty} P\Big(|\frac{1}{n}\sum_{i=1}^{n} X_i - m| \geq \epsilon\Big) = 0 \tag{70}$$

We say that we have ***convergence in probability****.*

Bound on the estimation error

Theorem

Weak law of large numbers
Let $(X_i)_{i\in\mathbb{N}}$ be a sequence of i.i.d. variables that have a moment of order 2. We note m their expected value. Then

$$\forall\epsilon > 0, \lim_{n\to+\infty} P\Big(|\frac{1}{n}\sum_{i=1}^{n} X_i - m| \geq \epsilon\Big) = 0 \tag{71}$$

*We say that we have **convergence in probability**.*

However, this is only an asymptotical result : it is a limit for $n \to +\infty$.

Bound on the estimation error

If we are able to bound $\sup_{h \in F} |R(h) - R_n(h)|$, then we have a bound on the estimation error.
To do this, we will use some other mathematical results :

- Boole's inequality
- Hoeffding's inequality (non-asymptotical probabilistic bound)

Boole's inequality

Proposition

Let $A_1, A_2, \dots,$ be acountable set of events of a probability space $\{\Omega, \mathcal{F}, P)$.
Then,

$$P\Big(\cup_{i \geq 1} A_i \Big) \leq \sum_{i \geq 1} P(A_i) \tag{72}$$

This set might be infinite.

Boole's inequality

Proposition

Let $A_1, A_2, \dots,$ *be acountable set of events of a probability space* $\{\Omega, \mathcal{F}, P)$*.*
Then,

$$P\Big(\cup_{i \geq 1} A_i \Big) \leq \sum_{i \geq 1} P(A_i) \tag{73}$$

This set might be infinite.
Exercice 9 : Proove the proposition.

Hoeffding's inequality

Theorem

Hoeffding's inequality
Let $(X_i)_{1 \leq i \leq n}$ *be n i.i.d real random variables such that* $\forall i \in [1, n]$, $X_i \in [a, b]$ *and* $E(X_i) = \mu \in \mathbb{R}$. *Let* $\bar{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$.
Then $\forall \epsilon > 0$,

$$P\Big(|\bar{\mu} - \mu| \geq \epsilon\Big) \leq 2 \exp\left(-\frac{2n\epsilon^2}{(b-a)^2}\right)$$

We admit this theorem.

Setting

- Supervised learning.
- Finite space of estimator F.
- The loss l is uniformly bounded : $l(\hat{y}, y) \in [a, b]$ with a and b real numbers.

Step 1

We have seen that

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{74}$$

As a consequence, for all $t \geq 0$:

$$P\Big(R(f_n) - R(f_a) \geq t\Big) \leq P\Big(2 \sup_{h \in F} |R(h) - R_n(h)| \geq t\Big) \tag{75}$$

Step 2

The fact that

$$2 \sup_{h \in F} |R(h) - R_n(h)| \geq t \tag{76}$$

is equivalent to :

$$\cup_{h \in F} \Big(2|R(h) - R_n(h)| \geq t \Big) \tag{77}$$

Step 3

Boole's inequality shows that :

$$P\Big(\cup_{h \in F} \left(2|R(h) - R_n(h)| \geq t\right)\Big) \leq \sum_{h \in F} P\left(2|R(h) - R_n(h)| \geq t\right) \tag{78}$$

Step 4

For each $h \in F$, we need to bound

$$P\Big(2|R(h) - R_n(h)| \geq t\Big) \tag{79}$$

Exercice 10 : What bound does Hoeffding's inequality give ?

Step 4

For each $h \in F$, we need to bound

$$P\left(2|R(h) - R_n(h)| \geq t\right) \tag{80}$$

With Hoeffding's inequality we get

$$P\left(2|R(h) - R_n(h)| \geq t\right) \leq 2\exp\left(-\frac{nt^2}{2(b-a)^2}\right) \tag{81}$$

Step 5

With Hoeffding's inequality we get

$$P\big(2|R(h) - R_n(h)| \geq t\big) \leq 2\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \tag{82}$$

Finally, putting everything together :

$$\begin{aligned} P\Big(R(f_n) - R(f_a) \geq t\Big) &\leq \sum_{h\in F} P\big(2|R(h) - R_n(h)| \geq t\big) \\ &\leq \sum_{h\in F} 2\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \qquad (83) \\ &= 2|F|\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \end{aligned}$$

Conclusion

$$P\Big(R(f_n) - R(f_a) \geq t\Big) \leq 2|F| \exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \tag{84}$$

Conclusion

We write

$$\delta = 2|F| \exp\left(-\frac{nt^2}{2(b-a)^2}\right) \tag{85}$$

We assume that $b-a=1$. Then, with probability $1-\delta$, we can compute and show that

$$R(\tilde{f}) \leq R(f_a) + 2\sqrt{\frac{\log(|F|) + \log(\frac{2}{\delta})}{2n}} \tag{86}$$

Generalization

It is possible to generalize to infinite sets :

- by sampling F
- by using Rademacher complexity and Vapnik Vapnik-Chervonenkis theory.

Interpolation regime

- If the number of parameters is sufficient, it is possible to have $R_n(f_n) = 0$.
- However, the statistical error does not seem to explode for some deep networks.

For instance, for Wide Resnet, $\frac{p}{n} = 179$ with

- p : number of parameters in the netwok
- n : number of samples

Double descent

[Belkin et al., 2019]

References I

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019).
Reconciling modern machine-learning practice and the classical bias–variance trade-off.
Proceedings of the National Academy of Sciences of the United States of America, 116(32) :15849–15854.

Bottou, L. and Bousquet, O. (2009).
The tradeoffs of large scale learning.
Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference, (January 2007).