Machine Learning for Time Series

Lecture 2: Feature Extraction and Selection

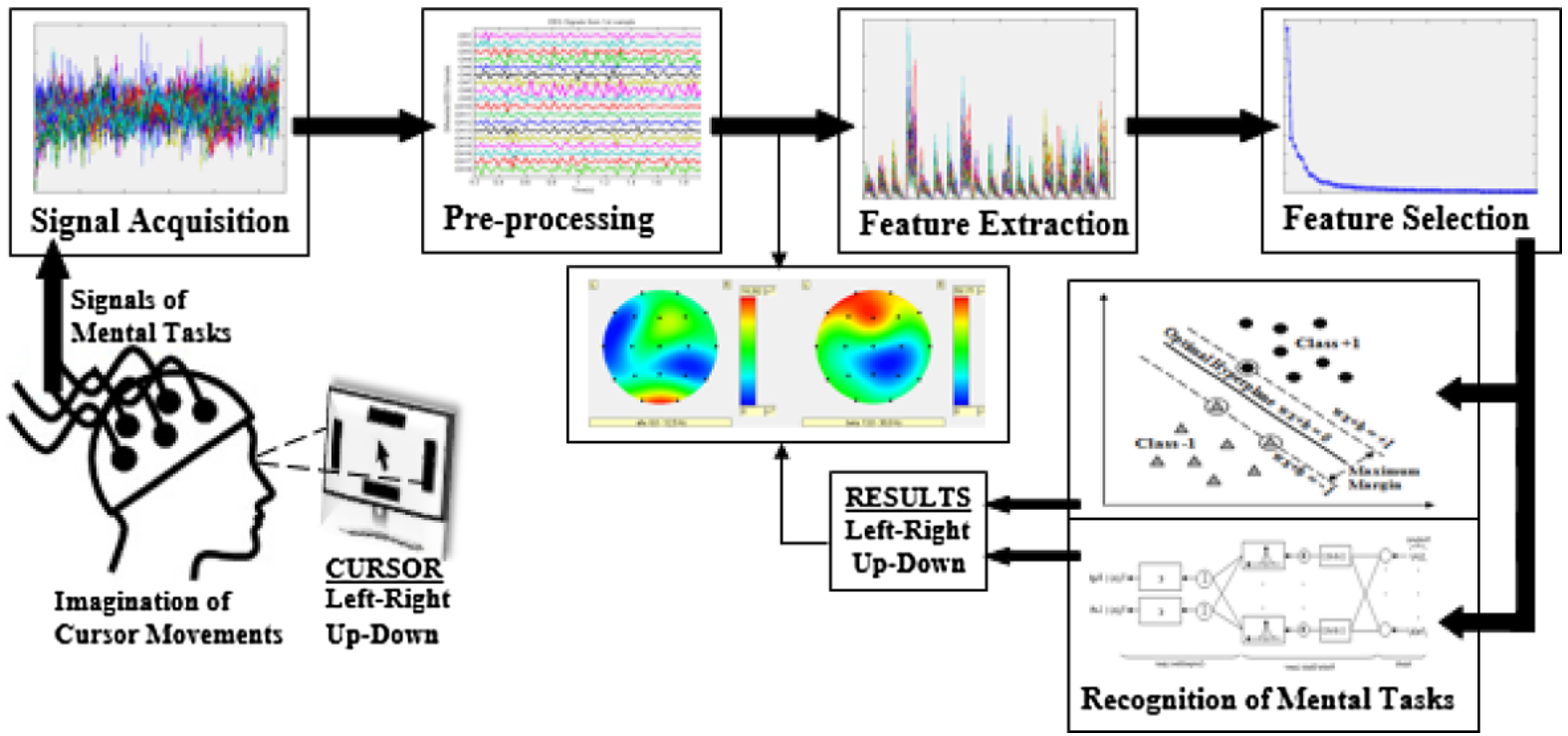
Laurent Oudre
laurent.oudre@ens-paris-saclay.fr

Master MVA
2023-2024

Contents

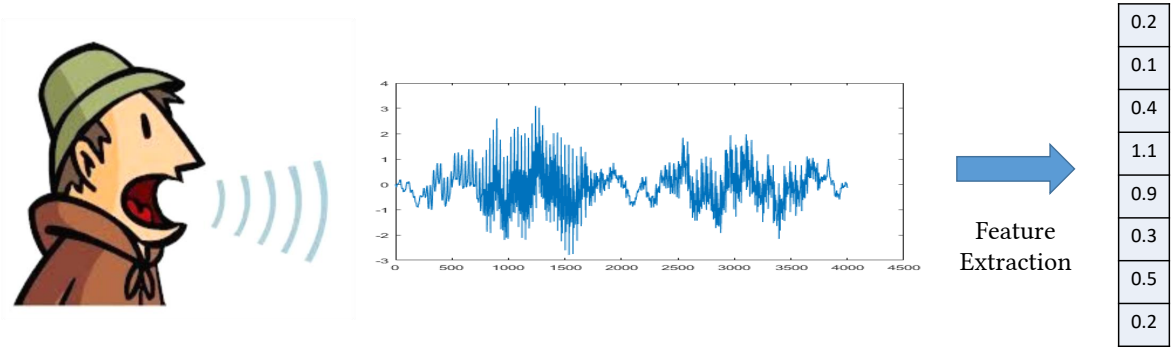
Contents

Motivation



Feature extraction/selection: fundamental step for using ML on time series

Problem 1: Feature Extraction

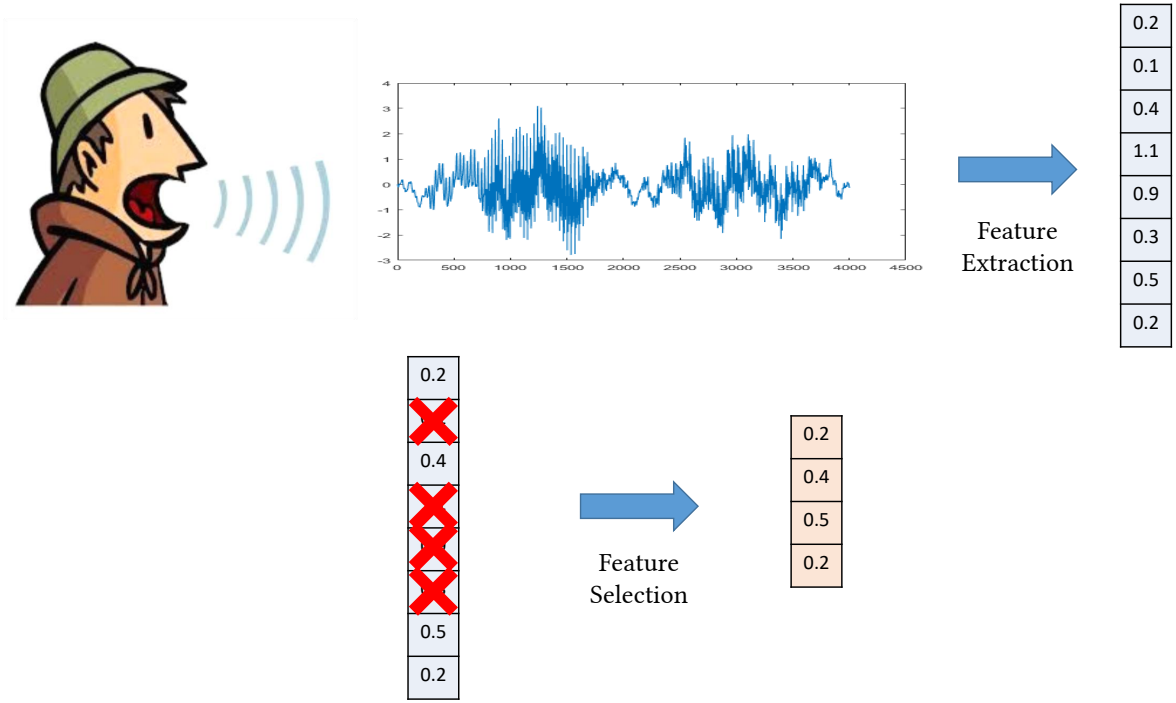


Feature Extraction

Given a time series of length N, extract a set of D features that characterizes the time series

- Useful for indexing, classification, clustering...
- Often based on expertise : feature engineering (construction of ad hoc features from domain knowledge)
- *Bag of features* representation for time series

Problem 2: Feature Selection



Feature Selection

Given a dataset of M time series represented each with D features, select the $K < D$ features that are relevant for a given task

- Can be performed in a supervised (labeled data) or unsupervised setting

Contents

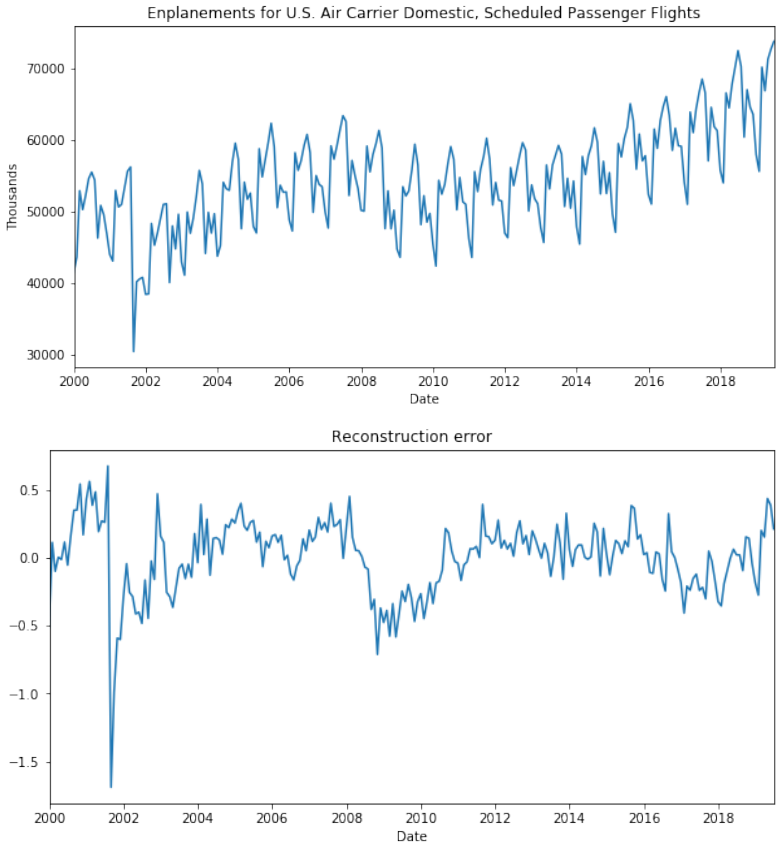
Good feature engineering

- Good feature engineering is based on domain knowledge: adapted to the data and to the underlying model (see Lecture 3 on Models and Representation Learning)
- In the general case, one solution is to use classical features that capture statistical or physical properties of the time series
- Reminder : two visions
  - **Physics :** a time series $x[1:N]$ is the result of the digitization of a physical phenomenon $x(t)$. Physical properties of this phenomenon can be retrieved and analyzed through the study of $x[1:N]$ (and vice/versa).

    Spectral features
  - **Statistics :** a time series $x[1:N]$ is a realization of a stochastic process $X[1:N]$. Statistical properties of this phenomenon can be retrieved and analyzed through the study of $x[1:N]$ (and vice/versa).

    Statistical features

The hidden assumptions behind feature extraction



Would it make sense to compute the empirical mean of these time series ?

- Some data scientists extract bags of features without thinking if there are relevant or not
- Even a simple feature such as the empirical mean can be wrong
- Before attempting to extract relevant information from a time series, one should ask the question: is it meaningful to have one unique feature to describe the whole time series?

Time series are random processes

- To formalize this question, we shall assume that $x[1:N]$ corresponds to a realization of a stochastic process $X[1:N]$
- Each $X[n]$ can be seen as a random variable
- Our capacity to extract meaningful features depends on the statistical properties of the implicit random process $X[1:N]$

Two fundamental properties: stationarity

- **Stationarity :** The statistical properties of the time series do not change over time
  - Order 1
$$\forall n, \quad \mathbb{E}\left[X[n]\right] = \mu$$
  - Order 2
$$\forall n_1, n_2, \quad \mathbb{E}\left[X[n_1]X[n_2]\right] = \gamma_X[|n_2 - n_1|]$$
  - Order 1 + Order 2 $\rightarrow$ wide-sense stationarity (most common assumption)

Two fundamental properties: ergodicity

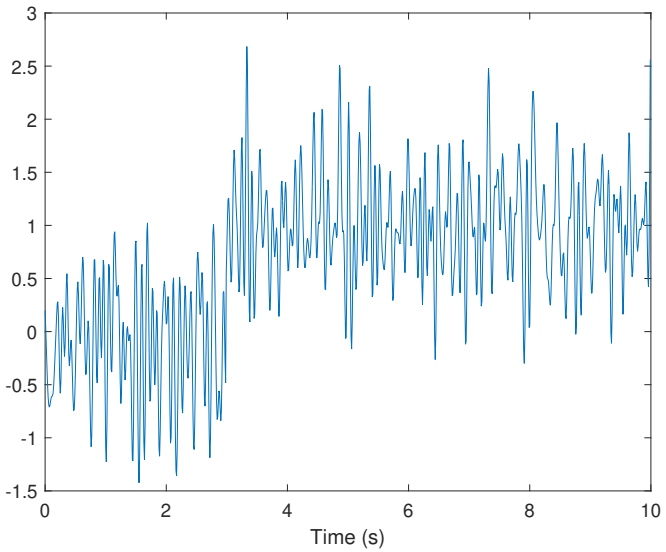
- **Ergodicity :** Statistical properties can be retrieved from temporal properties. Assuming that the time series is wide-sense stationary:
  - Order 1

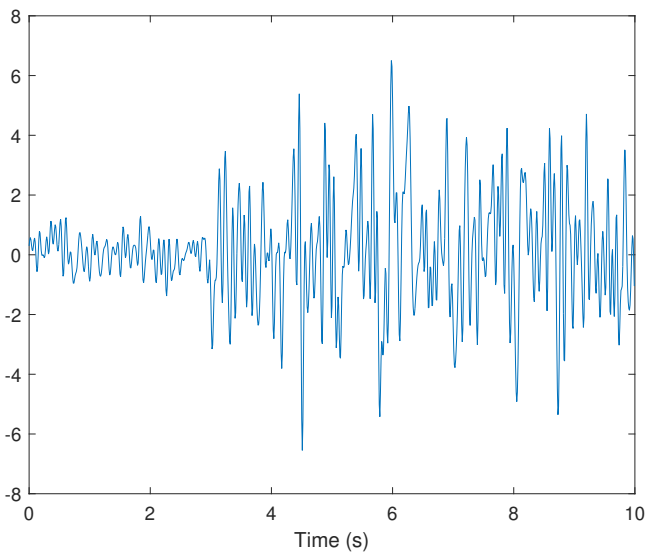
$$\forall n, \quad \mathbb{E}\left[X[n]\right] = \mu = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} x[n]$$

  - Order 2

$$\forall n_1, n_2, \quad \mathbb{E}\left[X[n]X[n+k]\right] = \gamma_X[k] = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} x[n]x[n+k]$$

Stationarity vs. non-stationarity





- None of these signals are stationary: any statistical features computed on the whole time series will be wrong
- In order to prevent this to happen, two solutions exist
  - Divide the signals into small frames where the signal is assumed to be stationary and ergodic
  - Use a change-point detection algorithm to detect these changes and work separately on each segment (see Lecture 5)

Statistical features

- A popular choice for features consists in extracting statistical properties of the time series
- In this context, we assume that $x[1:N]$ corresponds to a realization of a stochastic process $X[1:N]$, that is ergodic and wide-sense stationary
- Statistical features of $X[1:n]$ can be retrieved from estimates based on $x[1:n]$

Statistical features

Assuming that $X[1:n]$ is ergodic and wide-sense stationary, we can estimate several statistical properties from $x[1:n]$

- Mean

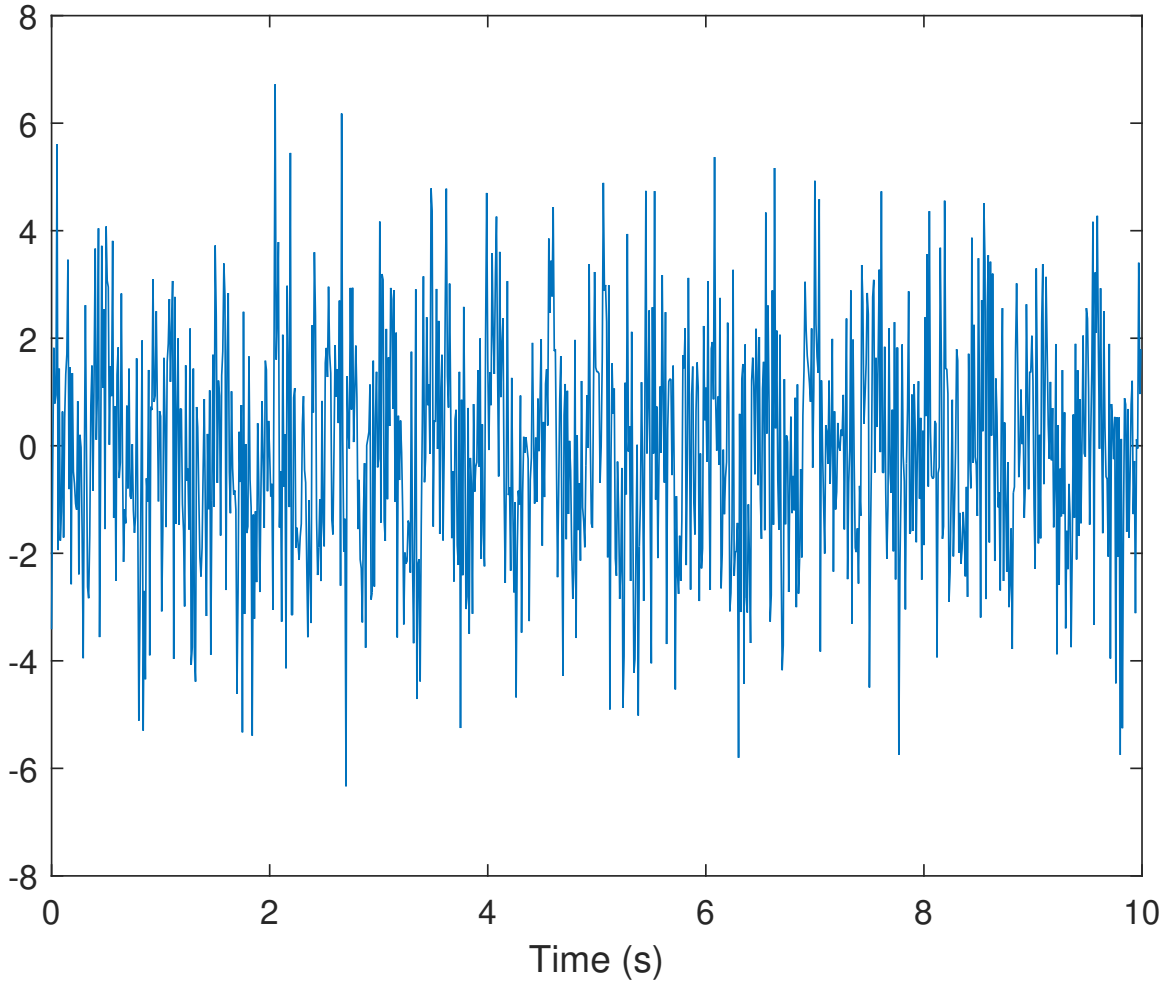
$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x[n]$$

- Autocorrelation function

$$\hat{\gamma}_x^{biased}[m] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x[n+m] \quad \text{where } x[n] = 0 \text{ for } n \neq 0 \dots N-1$$

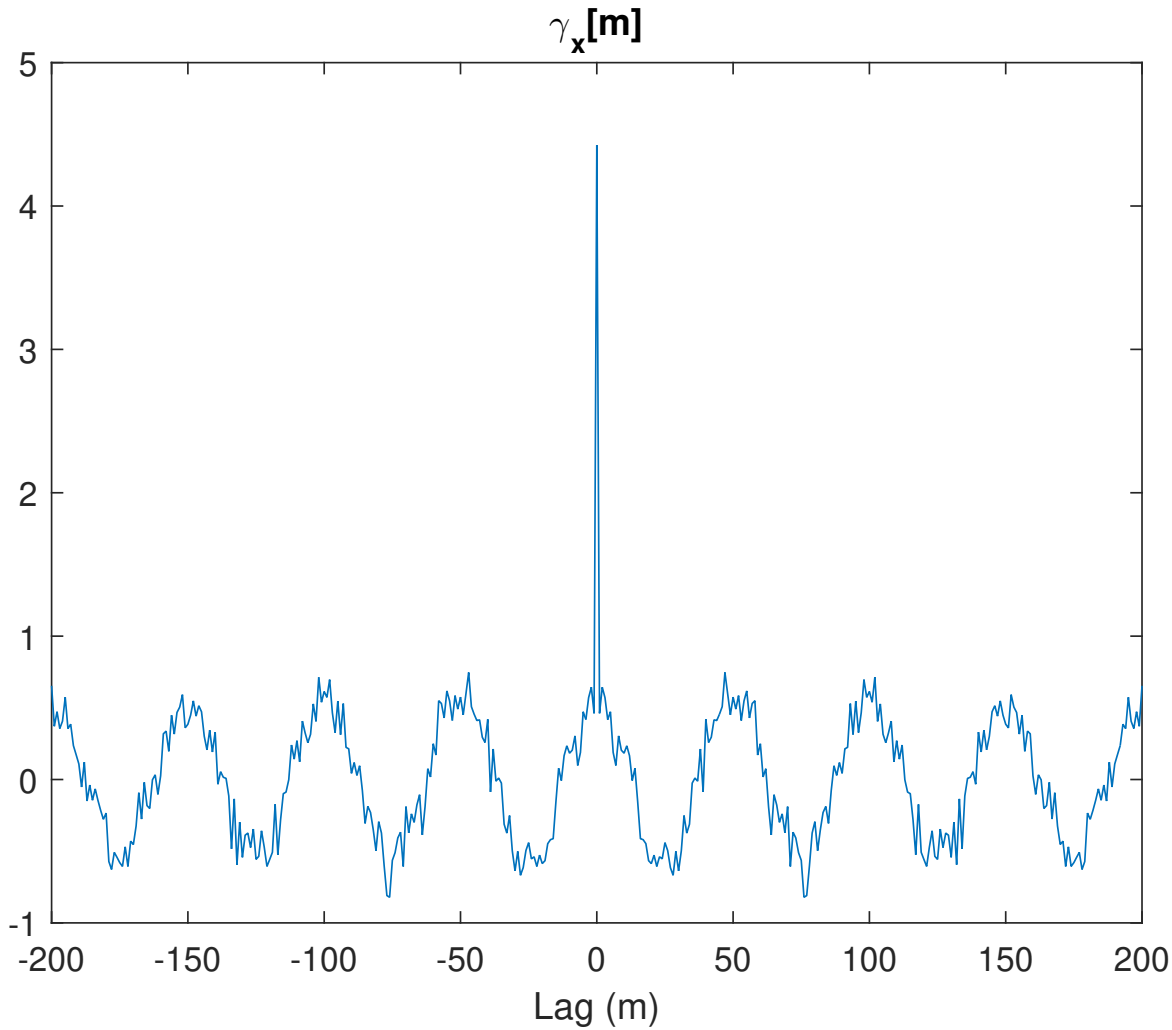
$$\hat{\gamma}_x^{unbiased}[m] = \frac{1}{N-|m|}\sum_{n=0}^{N-1} x[n]x[n+m] \quad \text{where } x[n] = 0 \text{ for } n \neq 0 \dots N-1$$

How to use the autocorrelation function



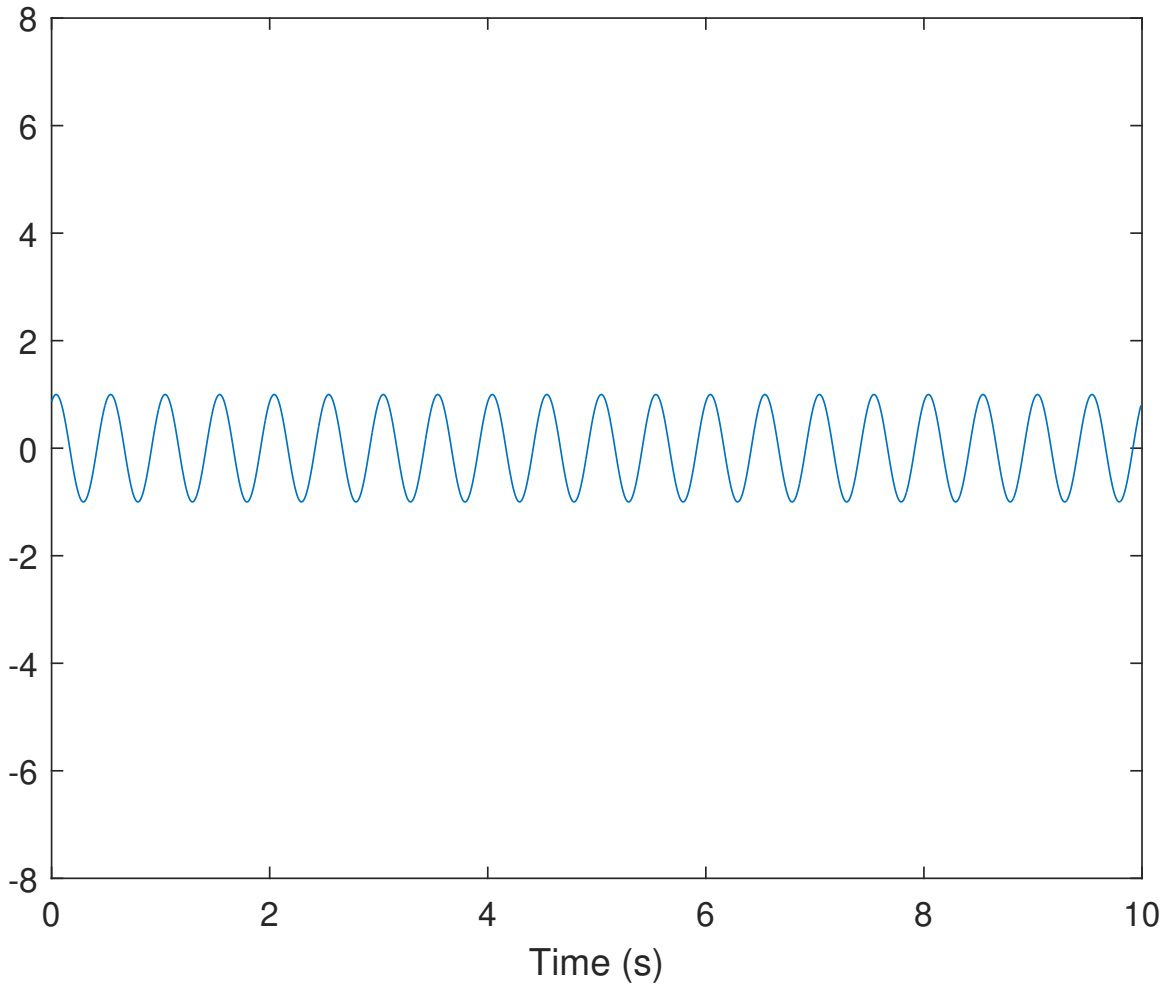
Original signal, sampling frequency 100 Hz

How to use the autocorrelation function



Autocorrelation function, peaks are visible for lags multiple of 50

How to use the autocorrelation function



A periodic signal with period $50 \times \frac{1}{100} = 0.5$ sec was hiding!

Features from autocorrelation function

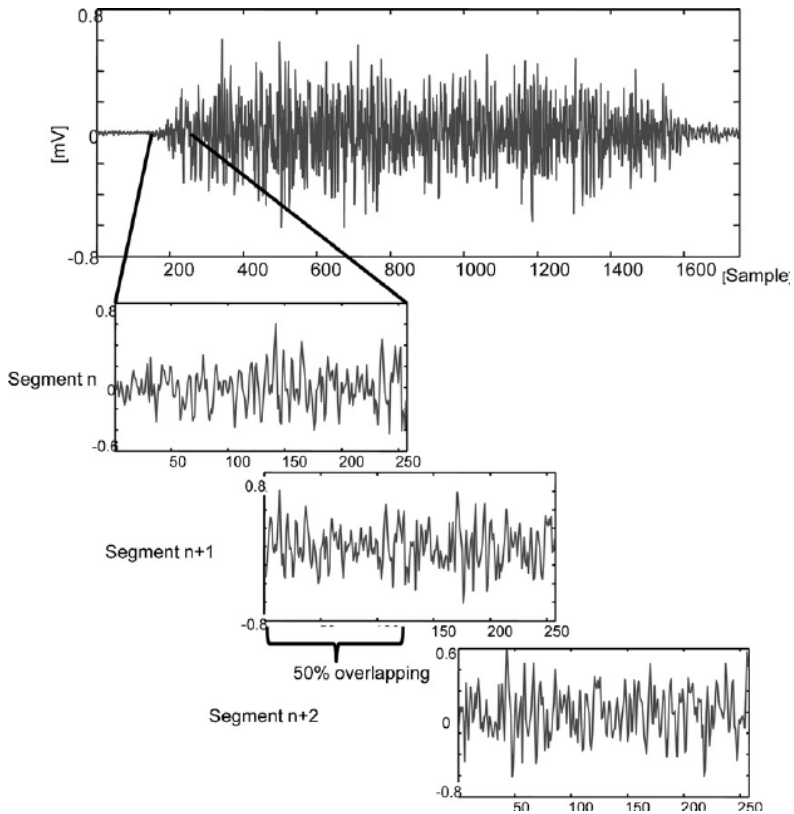
- Given a lag m, one useful feature is the renormalized autocorrelation coefficient

$$\gamma'_X[m] = \frac{\gamma_X[m]}{\gamma_X[0]}$$

  that quantifies the correlation between two samples spaced by m in the time series
- These features are very useful when the signal has a seasonality or periodic component (see Lecture 3): in this case, we store $\gamma'_X[m_0]$ where m_0 corresponds to the index of the first peak of the autocorrelation function ($m_0 > 0$)

Other statistics

- Other quantities can be estimated such as minimum and maximum values, or robust statistics such as median or percentiles (5%, 25%, etc.)
- Most of the time, these purely statistical features remove the time information...
- If necessary, those quantities can be extracted on sliding windows, increasing the number of features while preserving the time information



Spectral features

- Several tools derived from Fourier analysis and signal processing can be used to extract relevant features
- In this context, we assume that $\mathbf{x}$ corresponds to the discrete measurement of a continuous signal $x(t)$
- **Sampling theory:** uniform sampling period T_s and sampling frequency $F_s = \frac{1}{T_s}$

$$x[n] = x(nT_s)$$

Discrete Fourier Transform (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi \frac{kn}{N}} \text{ pour } 0 \leq k \leq N-1$$

where N is the number of samples

- The space between two observable frequencies is called **frequency resolution**

$$\Delta f = \frac{F_s}{N}$$

- $X[k]$ corresponds to the DFT for the physical frequency

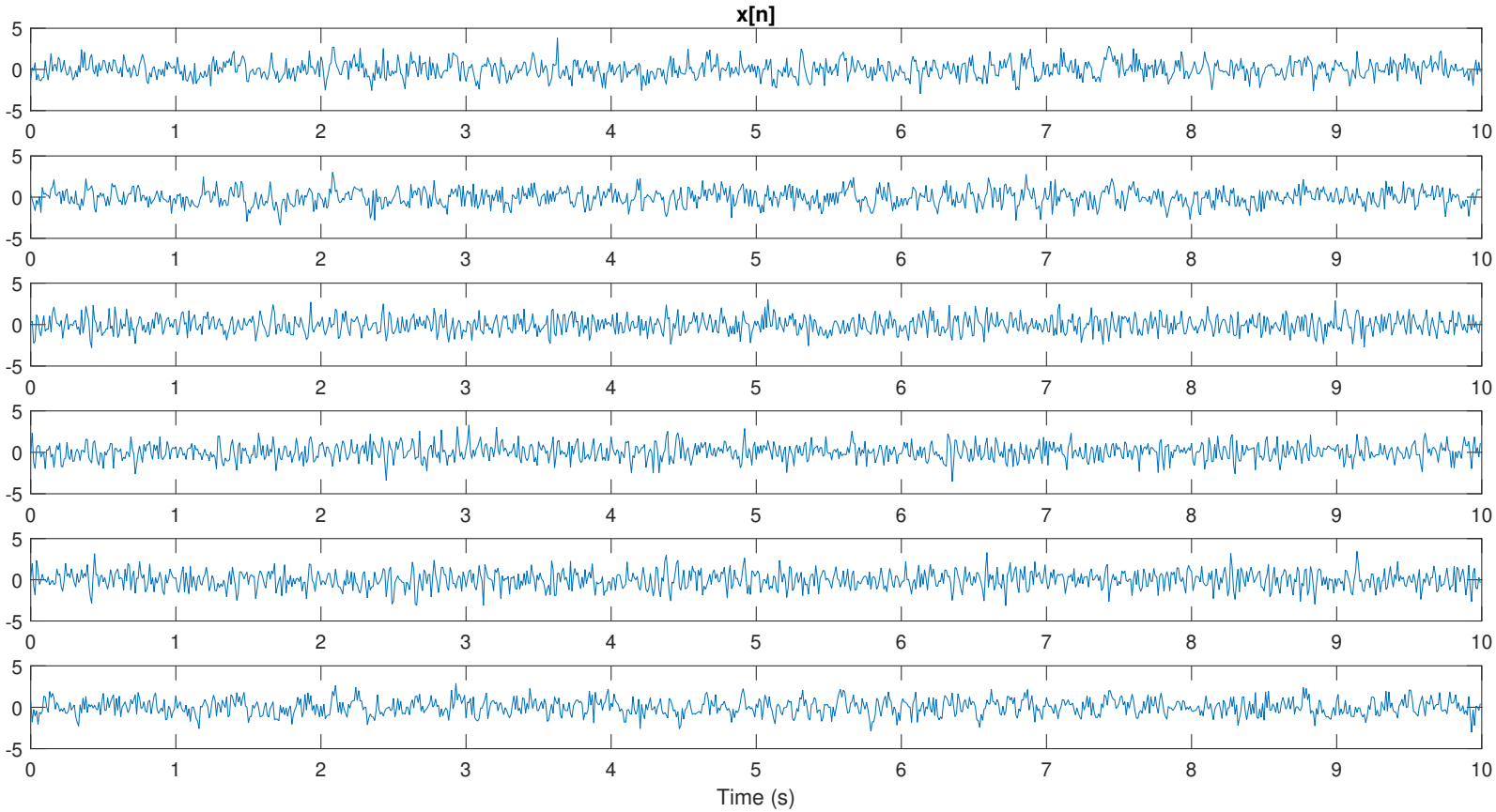
$$f[k] = k\frac{F_s}{N} \text{ for } 0 \leq k \leq N-1$$

- No physical frequency greater than $\frac{F_s}{2}$ can be observed (Nyquist theorem).

Spectral analysis

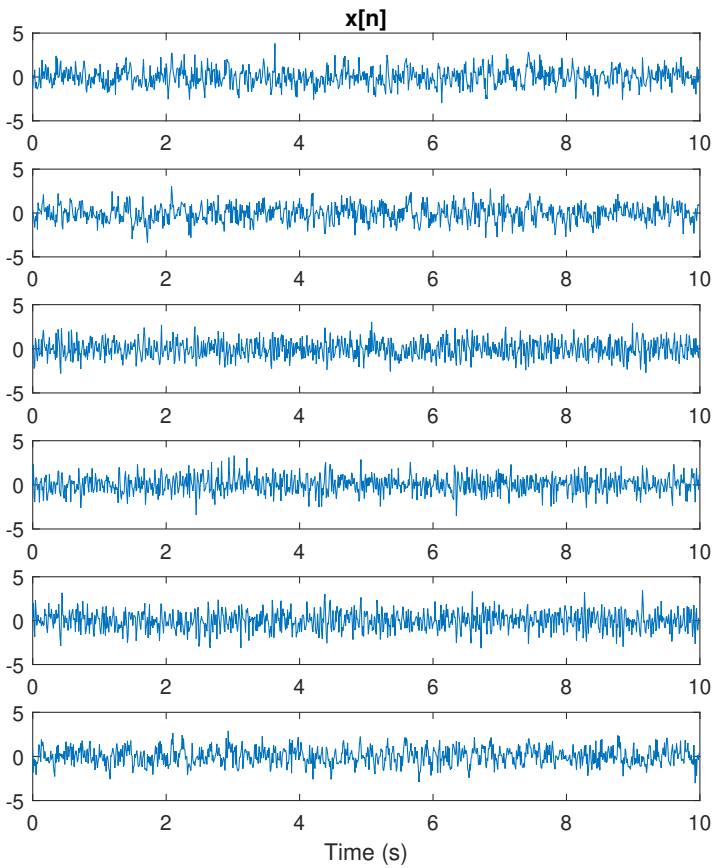
- $X[k]$ is a complex quantity: most of the time, we use the squared absolute values $|X[k]|^2$ instead
- The analysis of the quantity $|X[k]|^2$ can allow to discover interesting properties of the time series
- $|X[k]|^2$ with low frequencies $f[k]$ correspond to phenomena with smooth variations
- $|X[k]|^2$ with large frequencies $f[k]$ correspond to phenomena with fast variations
- One very useful plot consists in plotting $|X[k]|^2$ as a function of $f[k]$: such plot is often referred to as **spectrum** (hence spectral analysis)

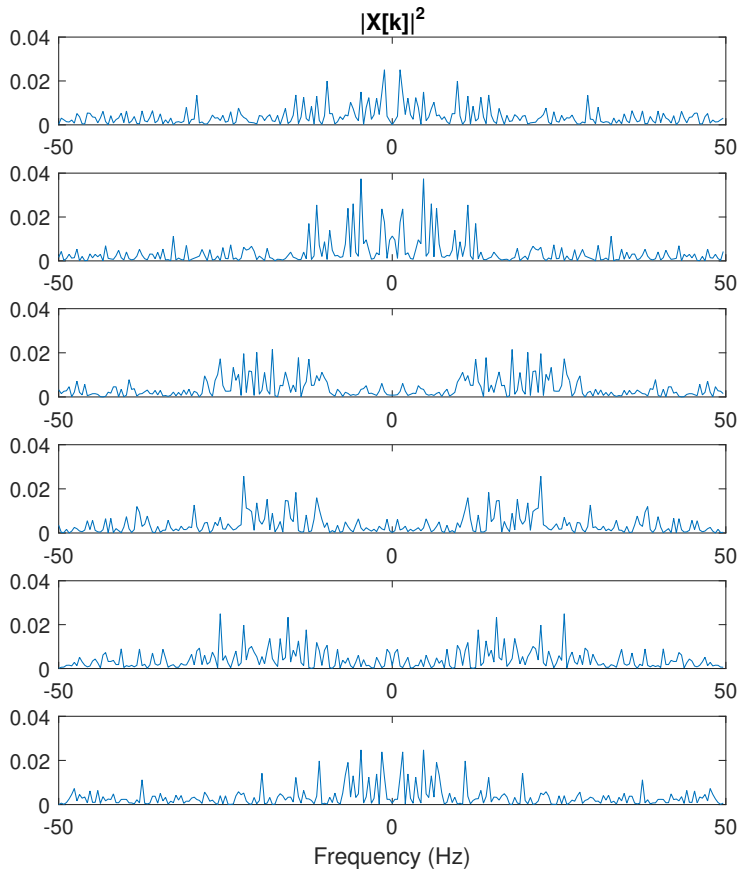
Example



Two classes of signals?

Example





Can be distinguished based on their DFT coefficients

DFT features

- $|X[k]|^2$ coefficients can be useful features
- In most cases, we can merge them on a frequency band of interest $[f_1, f_2]$, with $0 \leq f_1 < f_2 \leq \frac{F_s}{2}$
- The computed quantity is called relative energy, and is often renormalized by the total energy of the signal

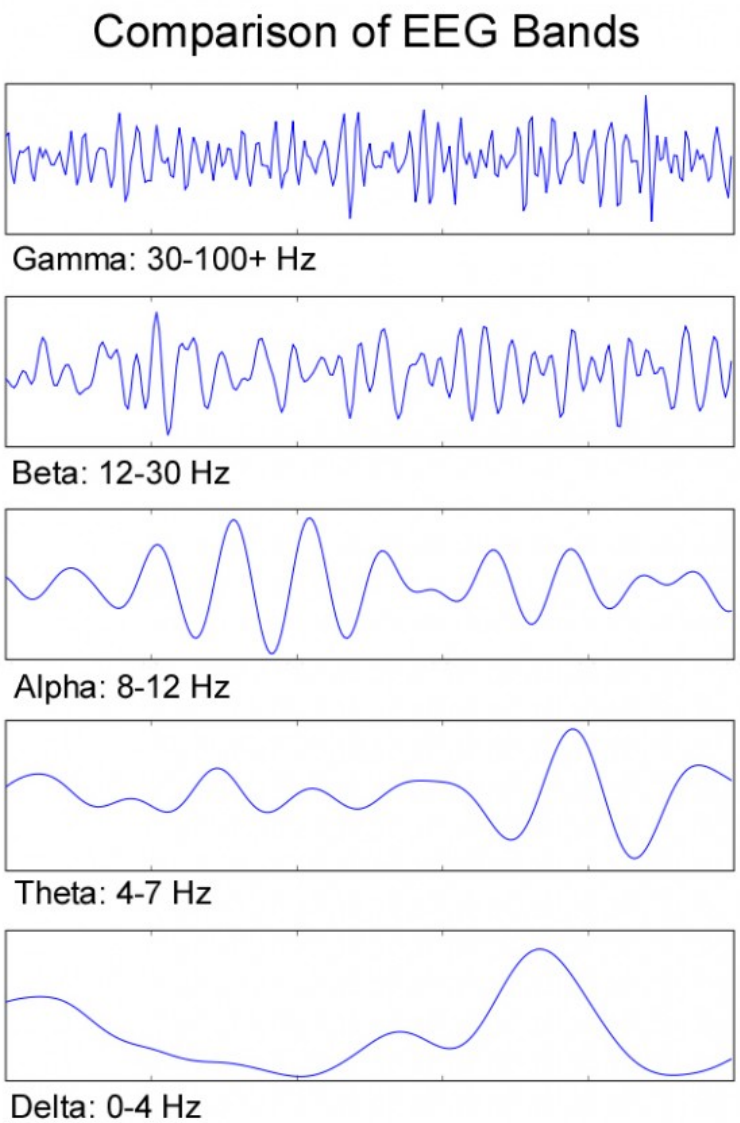
$$E_{[f_1,f_2]} = \frac{\sum\limits_{k, f[k] \in [f_1, f_2]} |X[k]|^2}{\sum\limits_{k, f[k] \in [0, \frac{F_s}{2}]} |X[k]|^2}$$

Example: EEG

Electroencephalography (EEG):

- Measures the electrical activity of the brain with a sensor network
- Used to study several neurological diseases (epilepsy, stroke...) and sleep
- Each sleep phase corresponds to the emergence of typical frequencies in the brain

Practical example: EEG



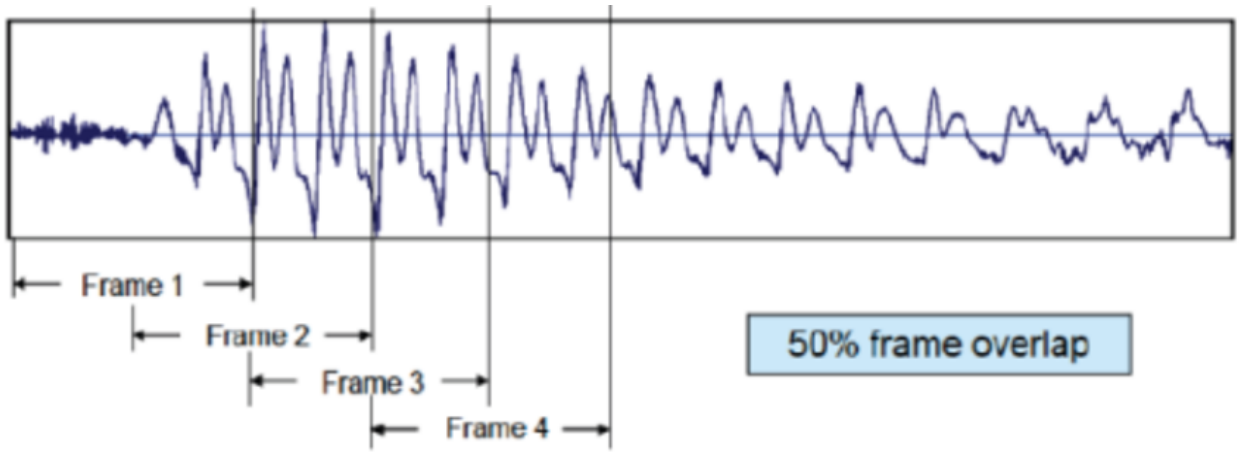
Each sleep phase corresponds to the emergence of typical frequencies in the brain:

- Delta waves: deep sleep without dreams
- Theta waves: deep relaxation
- Alpha waves: light relaxation
- Beta waves : awake, active information processing

Spectral features allow to figure out in which sleep phase a subject is

Spectrogram

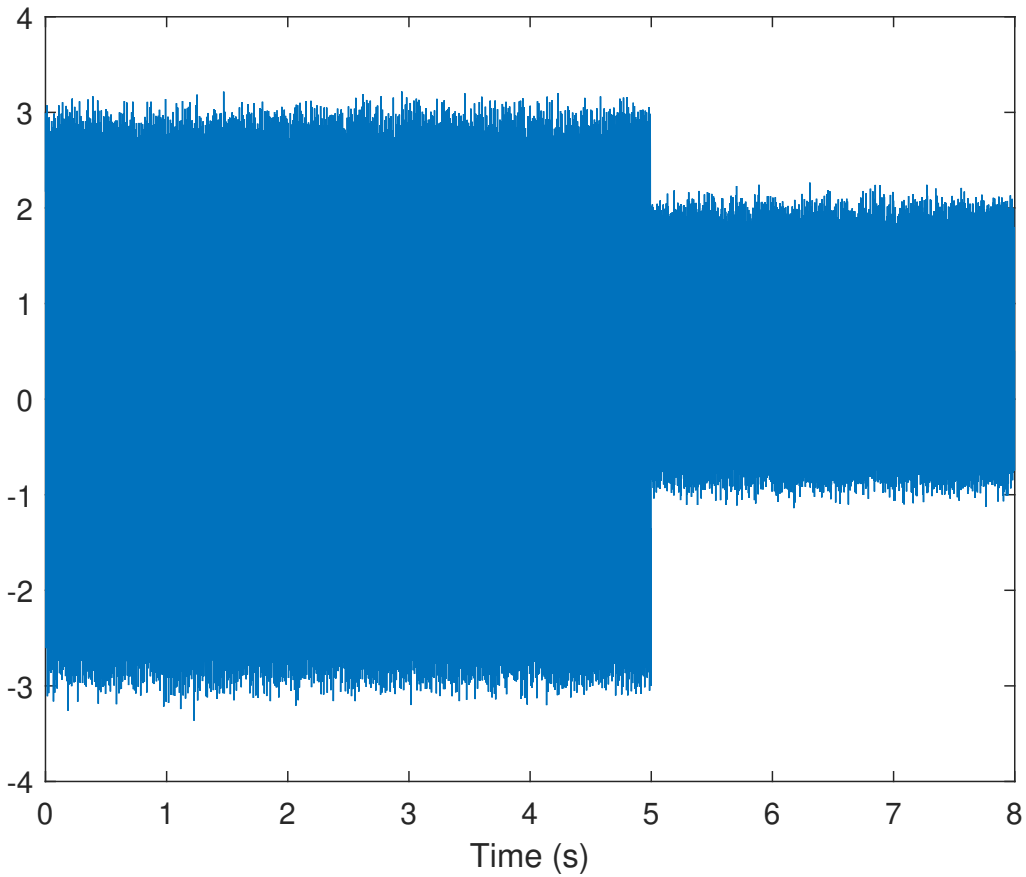
- When the properties of the time series tend to change with time, it is more careful to compute the DFT on sliding windows
- By sliding the window along the signal, we recover a time-frequency representation called **spectrogram**



- Matrix representation: each column corresponds to the DFT on the window of interest.

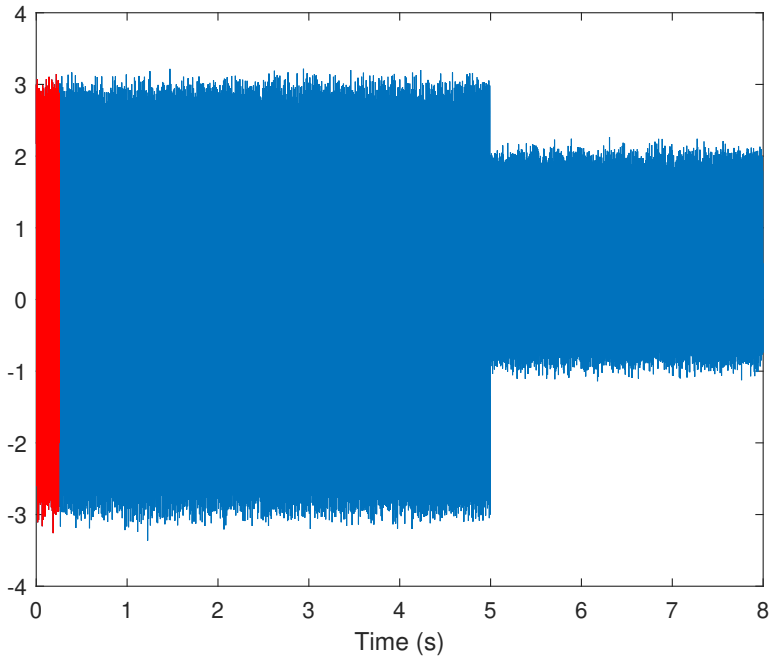
X-axis: frame number, Y-axis: frequency bin

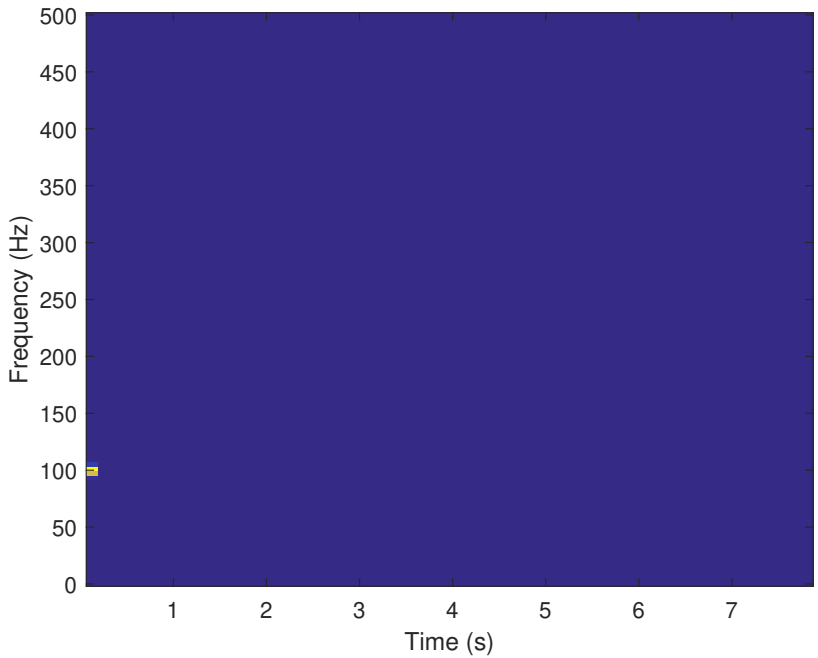
Example



Original signal, F_s = 1000 Hz

Example

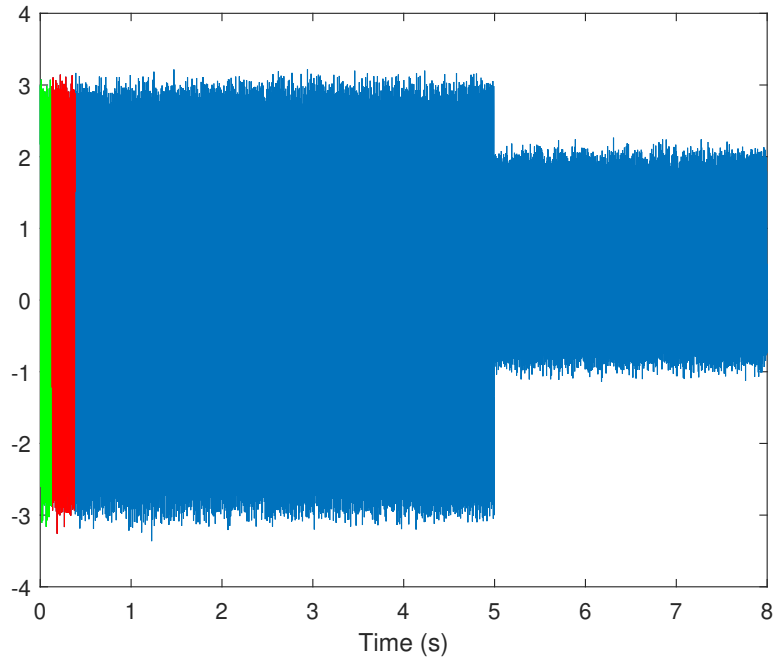


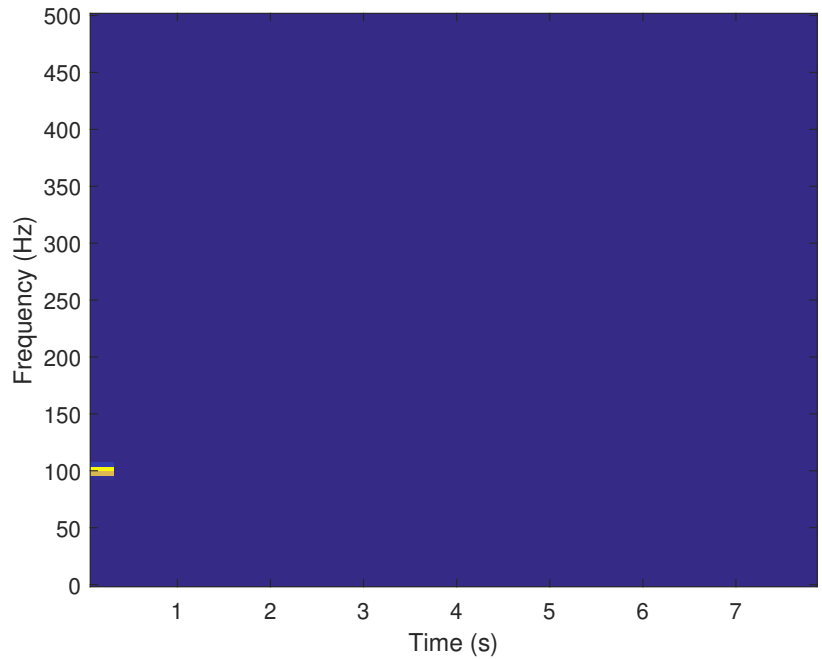


Window length $N_w = 256$

Computation of the DFT on the first frame and storage in the spectrogram matrix...

Example

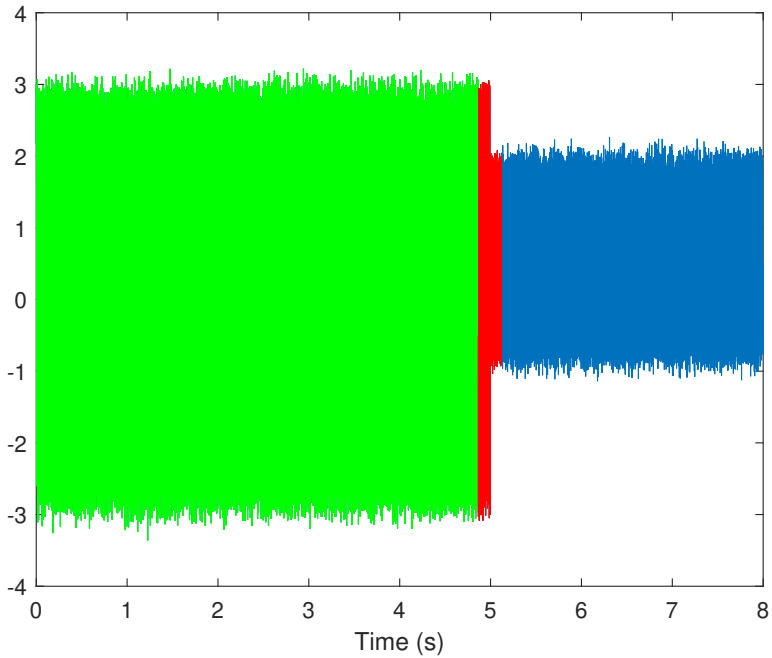


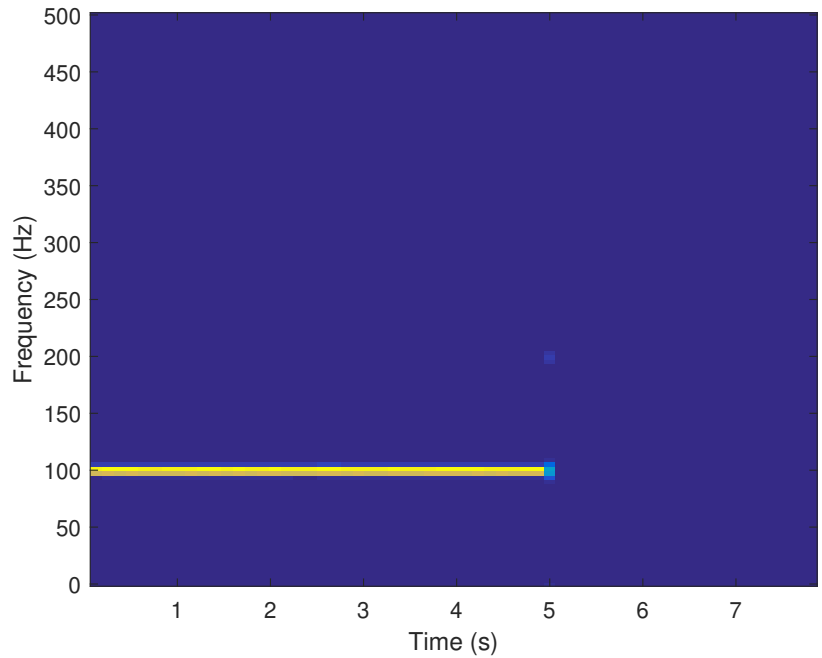


Window length $N_w = 256$

Computation of the DFT on the second frame and storage in the spectrogram matrix...

Example

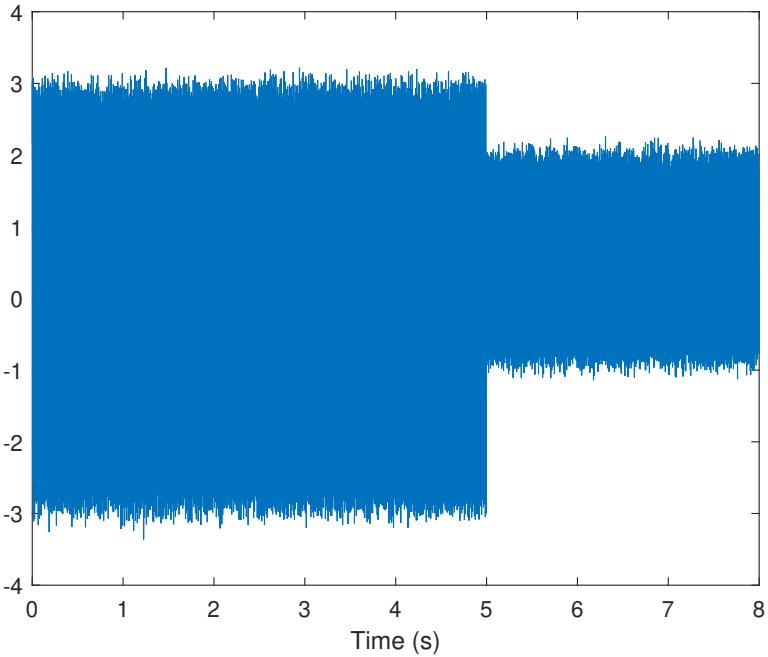


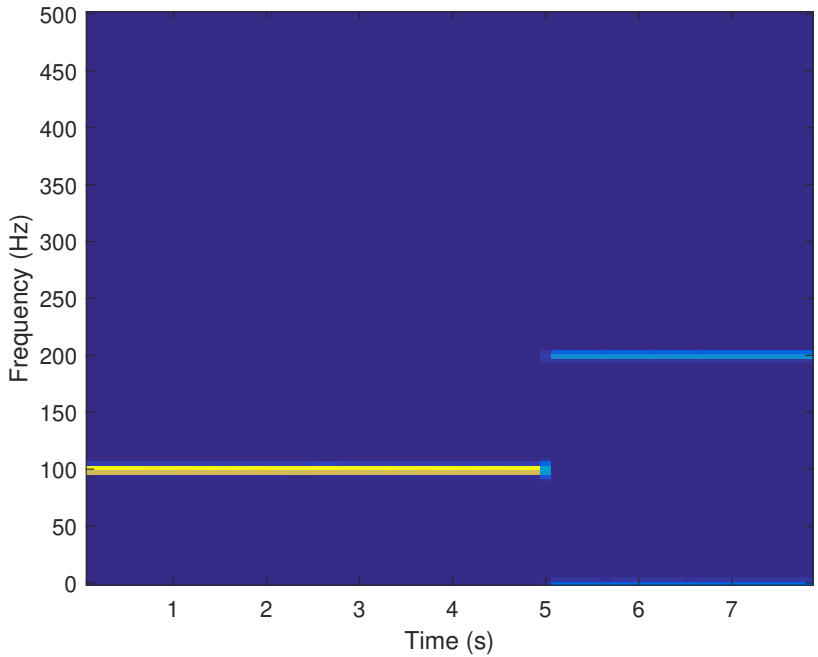


Window length $N_w = 256$

Same process...

Example





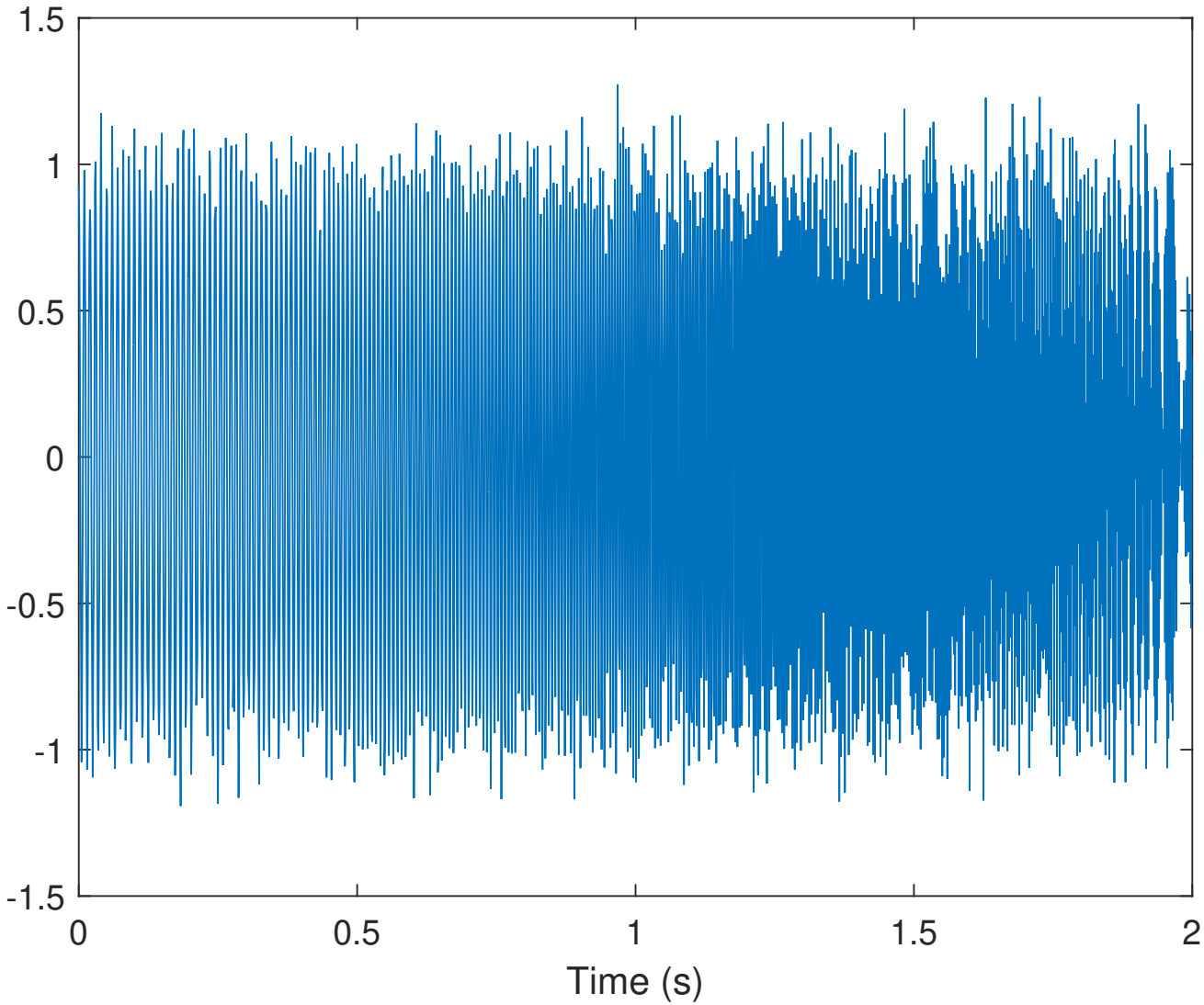
Window length $N_w = 256$

Final result

DFT vs. Spectrogram

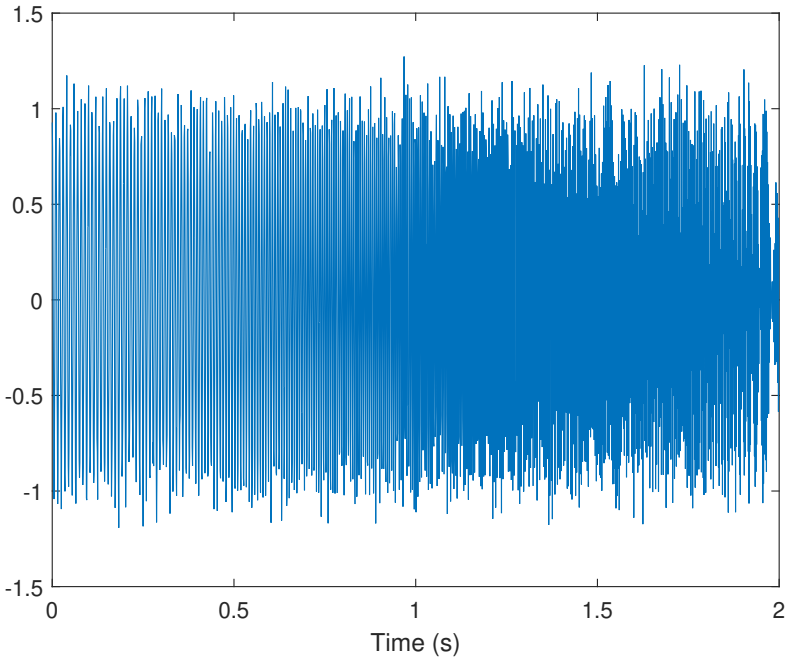
- Only use DFT when you are sure that there is no abrupt changes in the time series
- Note that using DFT will tend to average the frequency content on the whole time series, which can be tricky in some application contexts
- For safety, always first visualize the spectrogram to make sure that no significant changes occur

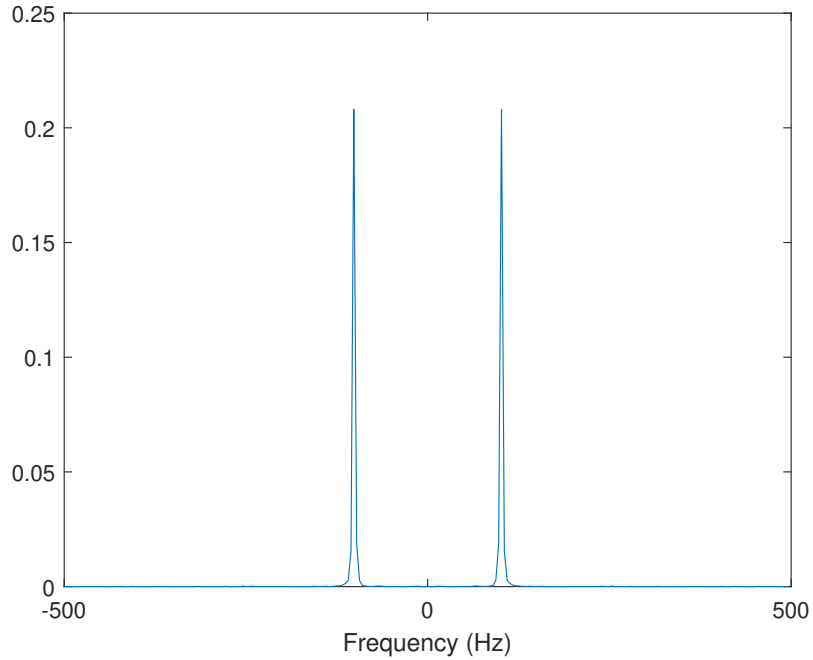
DFT vs. Spectrogram



Periodic phenomenon ? (sinusoid ?)

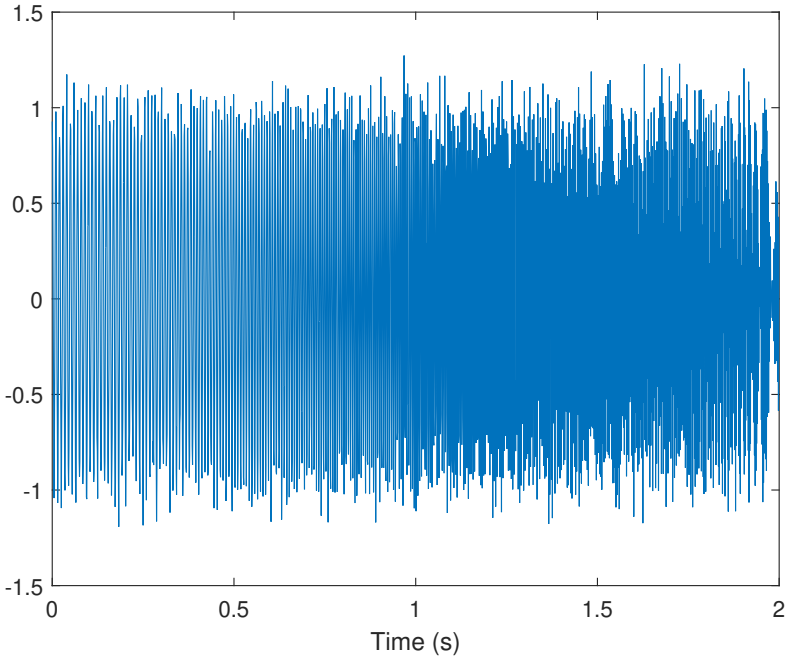
DFT vs. Spectrogram

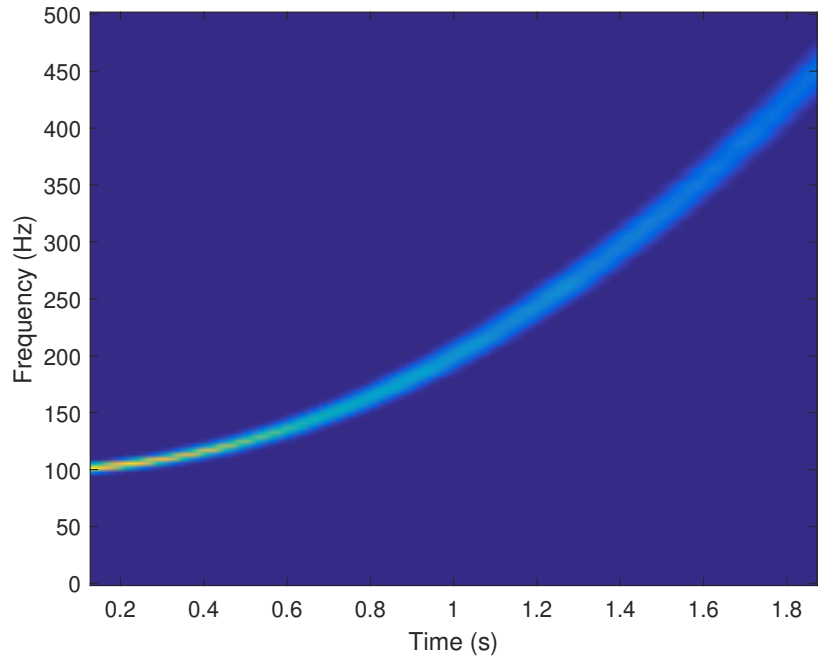




DFT suggests a sinusoidal phenomenon around frequency 100 Hz

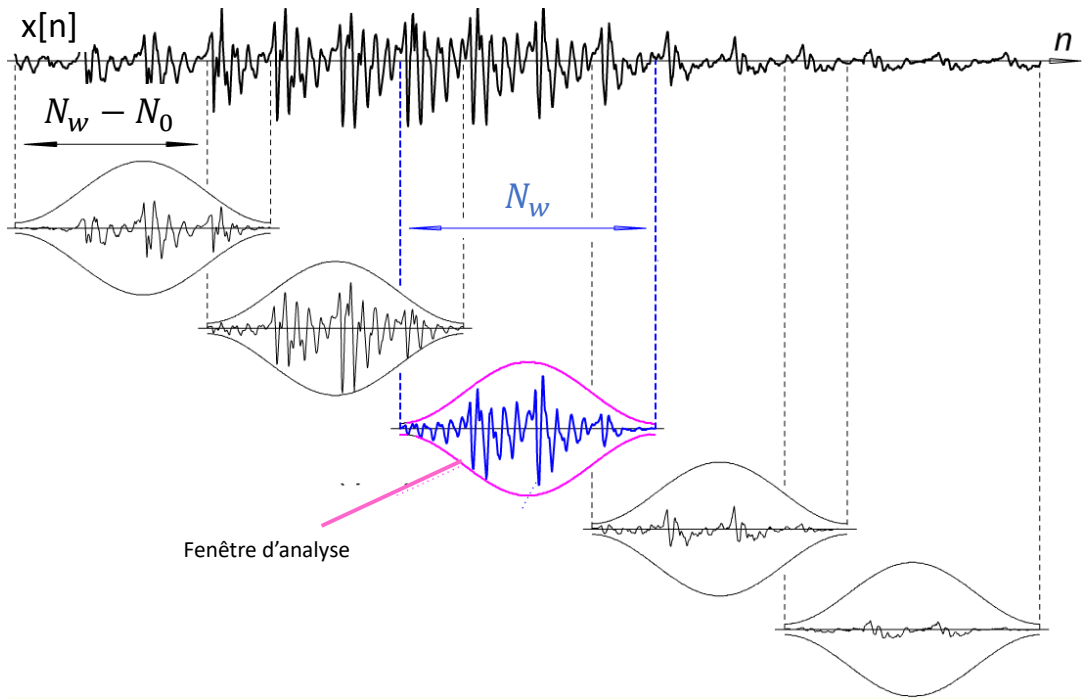
DFT vs. Spectrogram





In fact, chirp signal between 100 and 500 Hz !!

Hyperparameters for spectrogram

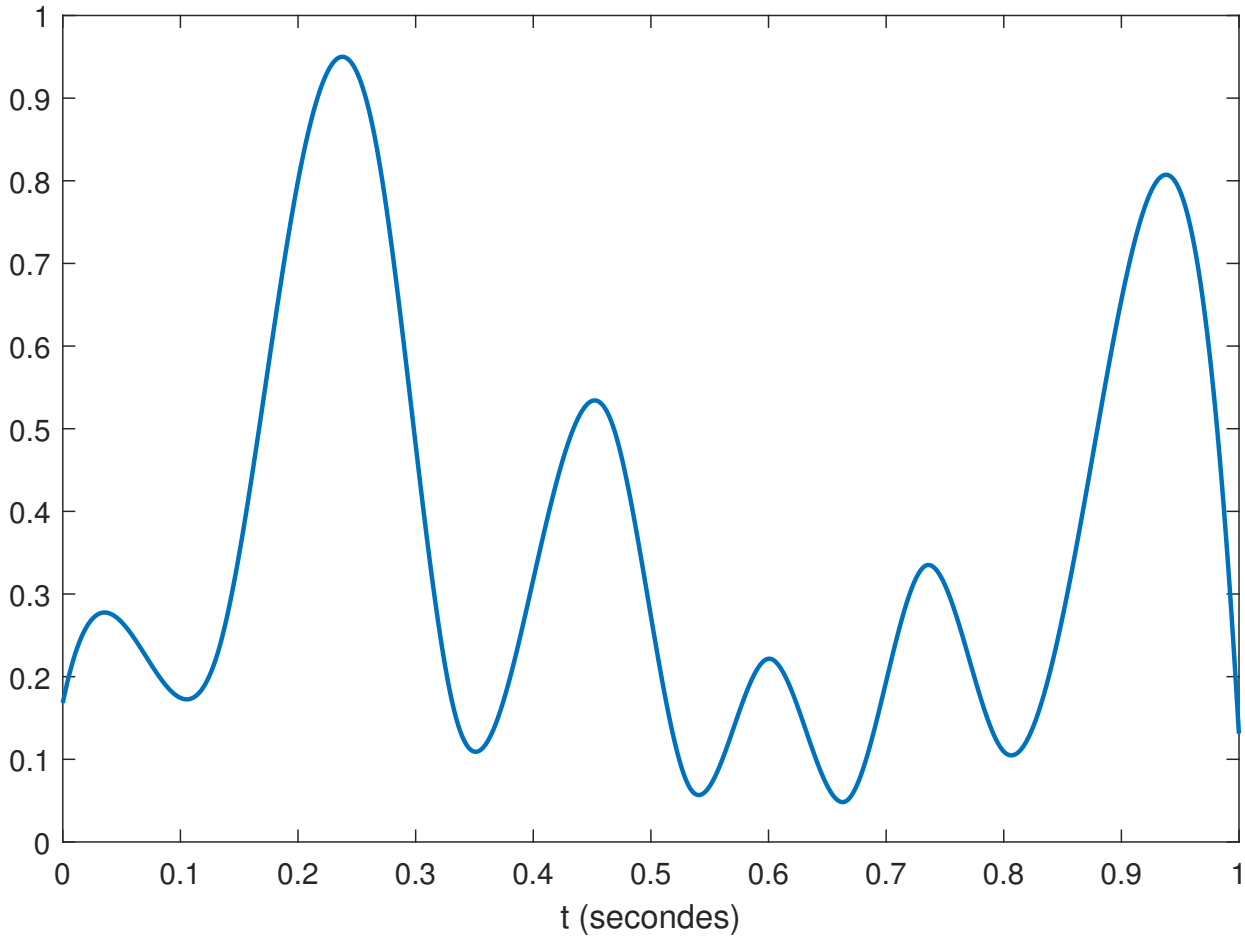


- N_w : window length (in samples)
  Often taken as a power of 2 (for FFT) and linked to the desired frequency resolution.
- N_o : overlap between two successive frames (in samples)
  Often taken as 50% or 75% of the window length and characterizes the time resolution (optimal when $N_o = N_w - 1$)
- w : analysis window (Hann, Hamming, Blackman...)
  Traditionally, in order to limit side effects, the signal frame is multiplied by an analysis window

Spectral analysis

- In reality, spectral estimation is a much more complicated subject than it seems... which is also linked to the statistical hypothesis on the signal
- When we compute the spectrum $\frac{|X[k]|^2}{N}$, we assume that the signal is wide-sense stationary, and we estimate the so-called Power Spectral Density (PSD) of the random process
- When it is the case and that the quantity $\frac{|X[k]|^2}{N}$ is too noisy, a solution consists in computing a spectrogram and then averaging it over time. This procedure is called the Welch's PSD estimator and if often cleaner and prettier.

Local symbolic features



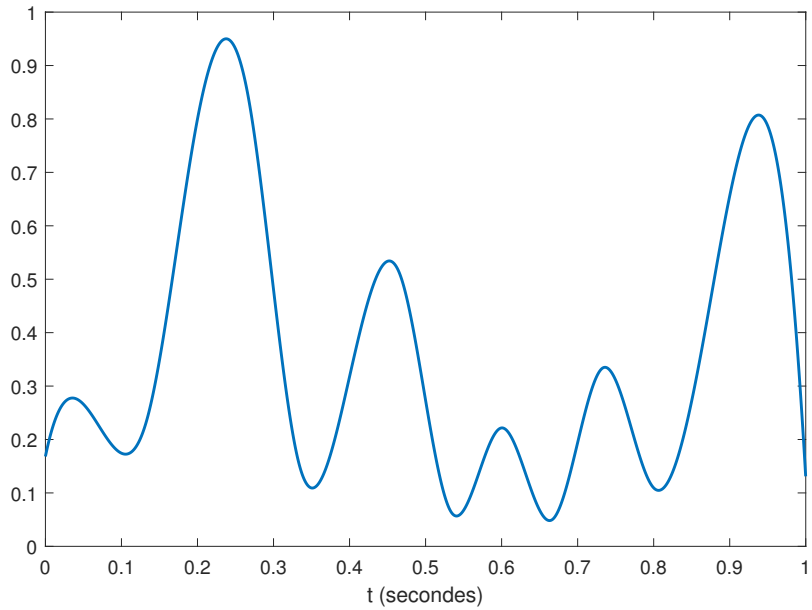
How would you describe such as signal?

small up, small down, big up, big down, up, down...

Local symbolic features

- By describing the whole time series as a succession of local behaviors, it is possible to fully encode the time series
- This encoding process is refered to as **symbolization**: it takes a time series as input and outputs a sequence of discrete symbols
- Each symbol represents a local behavior (amplitude, slope...) and statistical features can be extracted from that (number of occurrence of each symbol, number of transitions, etc...)
- These local symbolic features are the basis of several state-of-the-art time series classification techniques such as BOSS and variants [Schäfer et al., 2015]

Quantization/symbolization

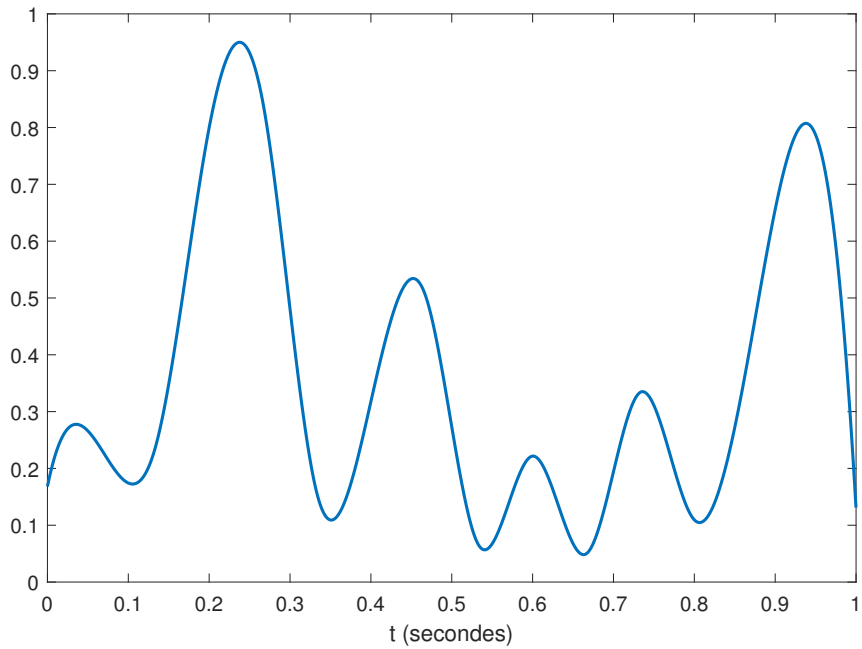


CCCFJHCBEEBBBACCBEHF

Quantization/symbolization

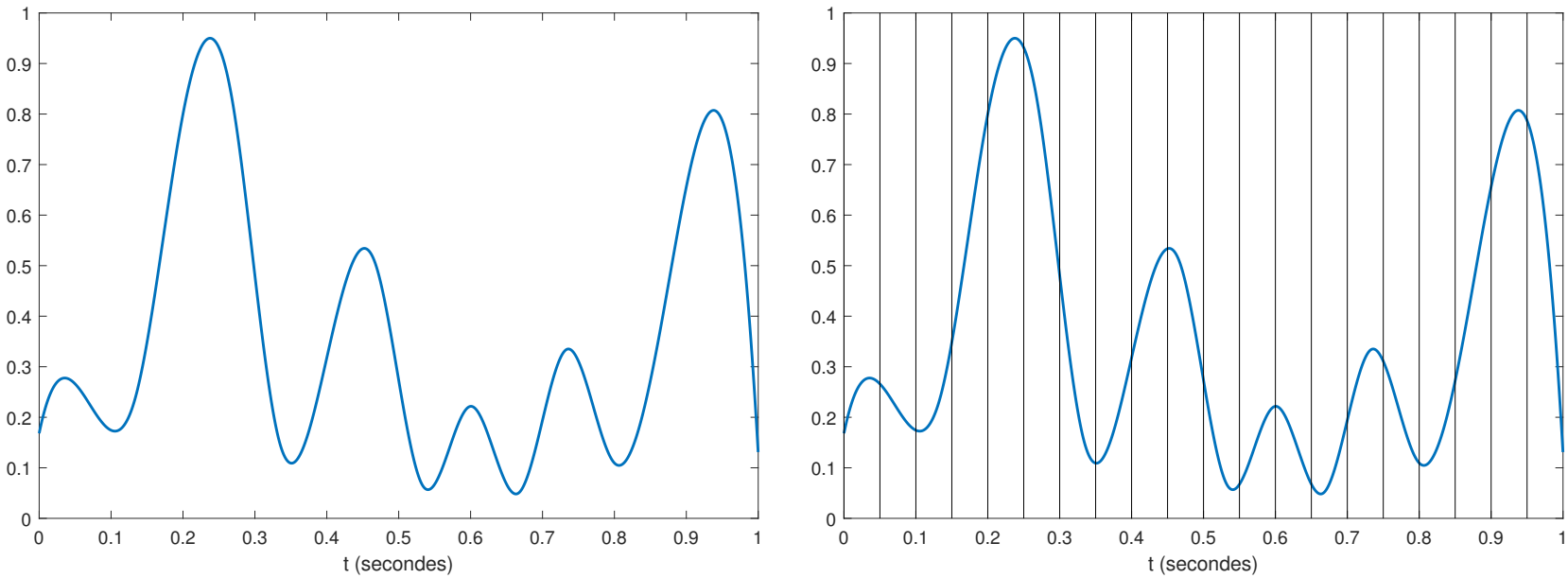
Given a real-valued signal x of length N, transform it into a discrete-valued time series y of smaller length.

Example



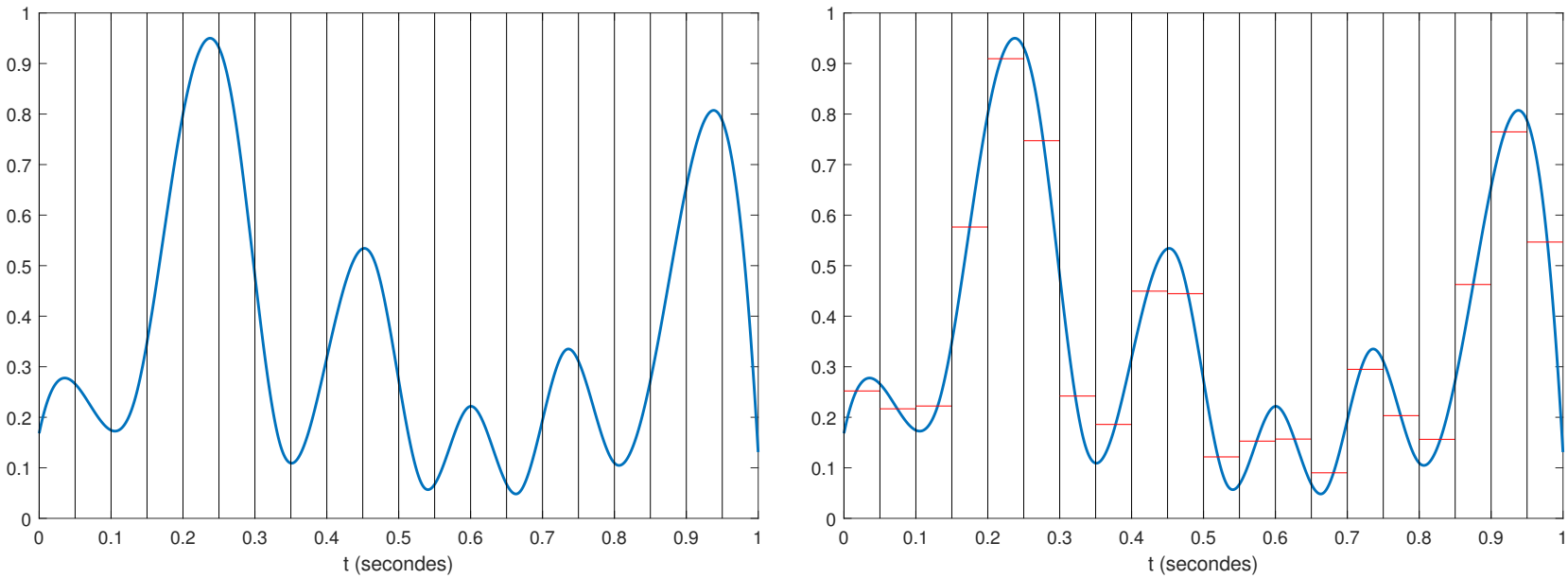
Original signal, sampled at $F_s = 10000$ Hz

Example



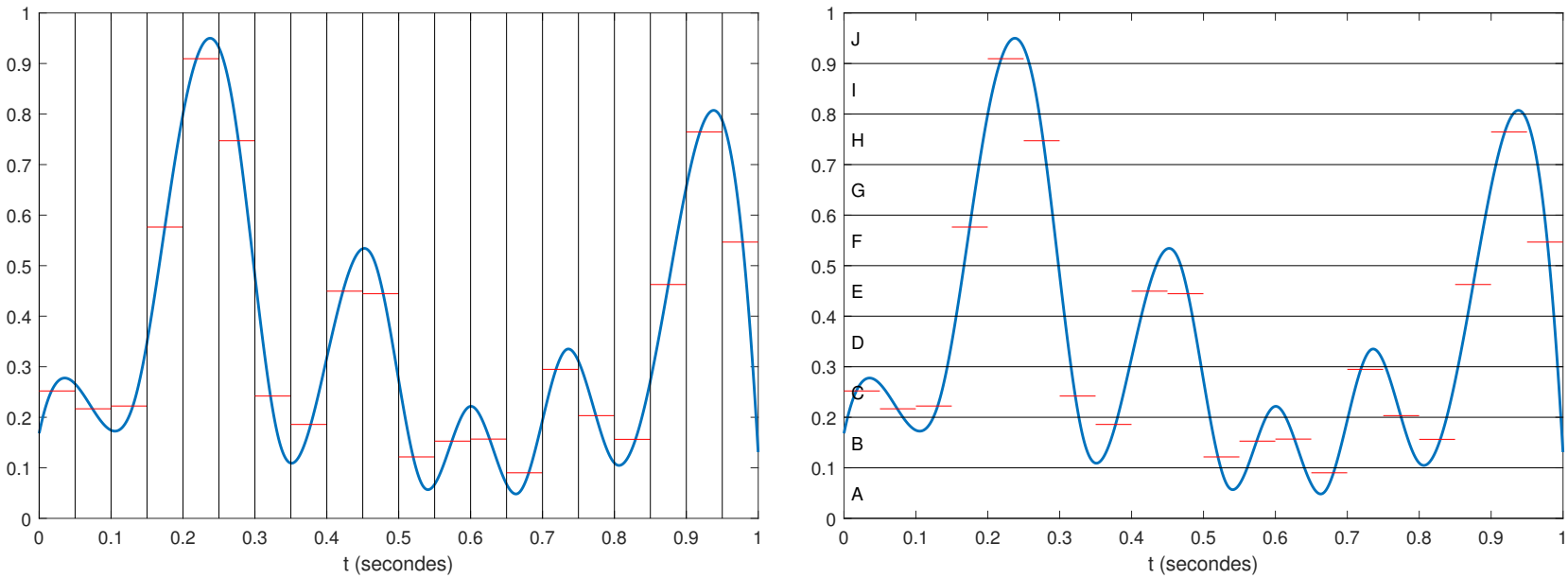
Division into non-overlapping frames of 500 samples

Example



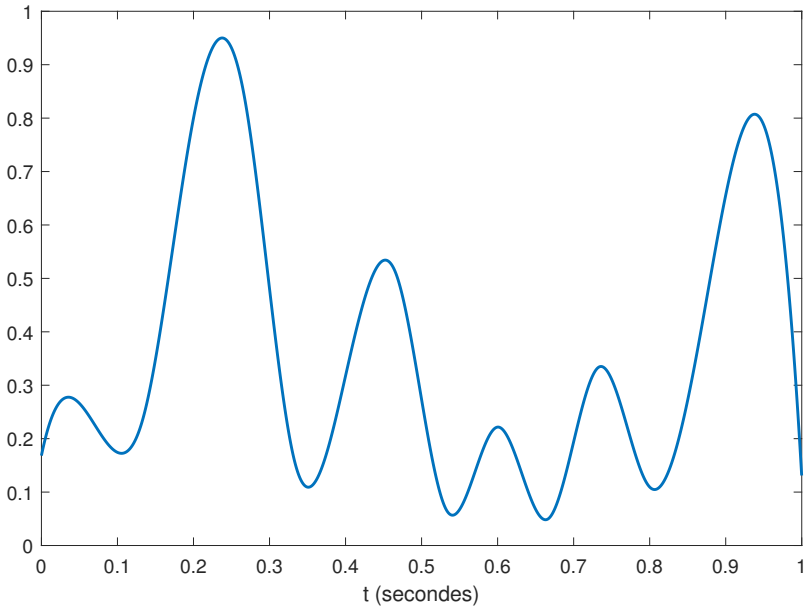
Average value on each segment

Example



Quantization of the amplitudes

Example



CCCFJHCBEEBBBACCBEHF

Symbolization of the time series
640 kbits vs. 0.1 kbit !

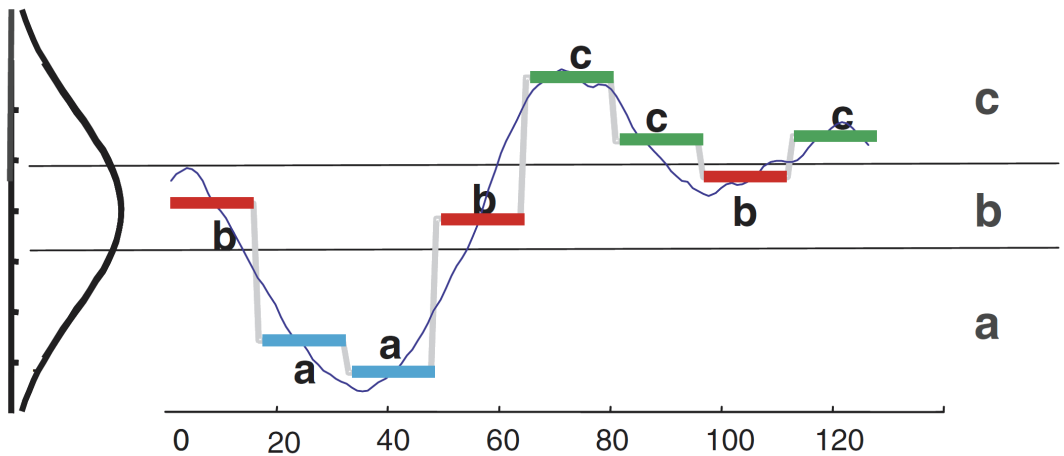
The SAX representation

- SAX, for Symbolic Aggregate approXimation [Lin et al., 2007], has been introduced as a strategy for computing such symbolization
- Input parameters
  - w : desired size of the output representation
  - A : number of symbols in the alphabet
- Each time series is encoded on $w \times \log_2(A)$ bits
- Authors have also developed a distance, called MINDIST, such that, given two time series $\mathbf{x}$ and $\mathbf{y}$:

$$\text{MINDIST}\left(\text{SAX}(\mathbf{x}), \text{SAX}(\mathbf{y})\right) \leq d_{EUC}(\mathbf{x}, \mathbf{y})$$

which allows to preserve some classification properties on the symbolic space

The SAX representation



1. Normalize the time series (zero mean, unit variance)
2. Divide the time series into w segments and compute the mean
3. Affect one symbol per segment. The A quantization intervals are determined so that all symbols have the same probability according to a $\mathcal{N}(0, 1)$ distribution.

How to construct features

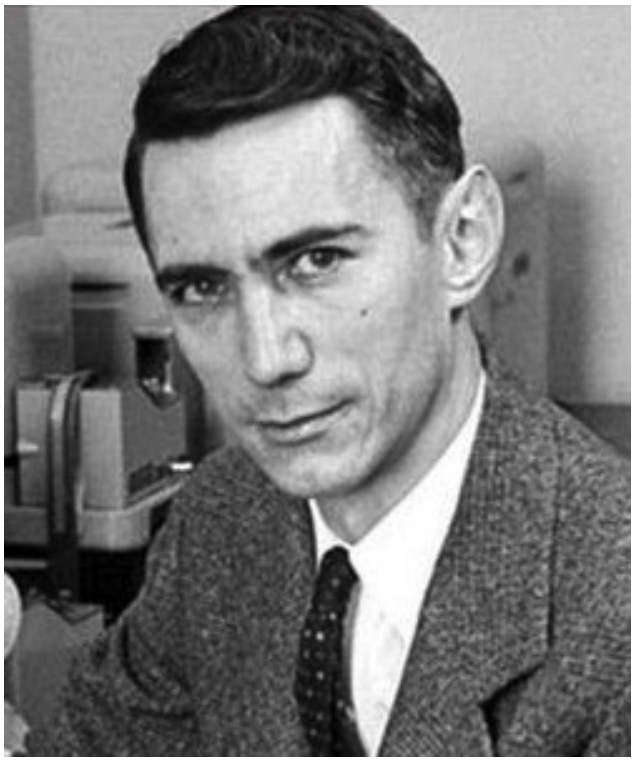
- Statistical features can be extracted from the discrete sequence of symbols (mainly histograms)
- The symbolization can be applied on sliding windows so as to form super-symbols that can be aggregated
- Multiscale approaches with different window sizes can also be used, and the histograms merged so as to provide a better representation
- Dozens of different symbolization procedures exist in the litterature (encoding the slopes, the local Fourier coefficients, the min/max values etc...) - see mini-projects

Quantization/symbolization

Quantization/symbolization can be used for several other tasks

- It can reduce time complexity and storage size
- It allows full enumeration of some phenomena of interest (patterns, anomalies...)
- It allows to use for time series several techniques from NLP or bioinformatics: pattern recognition, anomaly detection, etc... for discrete sequences

Information theory



- Shannon's information theory has emerged in the 1950s as an unified framework for representing digital information
- One of the main breakthroughs that enabled the design of digital communications
- Data can be represented in binary form, can be compressed, sent with low error probability, etc : data compression, data encryption
- Each message contains a certain quantity of information, that is linked to its probability of apparition

Entropy

- Given a random variable X in a finite alphabet $\mathcal{X}$, the **entropy** is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p_X(x) \log_2 \left(p_X(x) \right)$$

- It represents the average *quantity of information* contained in the variable
- Maximum entropy is achieved when X follows an uniform law : maximum uncertainty on sample values. In this case

$$H(X) = \log_2 \left(|\mathcal{X}| \right)$$

Entropy of two variables

Given two random variables X and Y, we can define:

- The joint entropy $H(X, Y)$: average information provided by the couple (X, Y)

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log_2 \left(p_{XY}(x, y) \right)$$

- The conditional entropy $H(X|Y)$: uncertainty on X given Y

$$H(X|Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log_2 \left(p_{X|Y}(x|y) \right)$$

- The mutual information $I(X; Y)$: quantity of information shared by X and Y

$$I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log_2 \left(\frac{p_{XY}(x, y)}{p_X(x) p_Y(y)} \right)$$

How to apply this to time series?

- First, time series samples are not necessarily discrete : in most case, we have $x[n] \in \mathbb{R}$
  - Solution 1: Use quantization or symbolization techniques to transform the time series into a discrete sequence
  - Solution 2: Use an adapted distance to redefine the notion $\mathbb{P}(X = x)$ as $\mathbb{P}(d(X, x) \leq \epsilon)$
- Second, when computing entropy, time information is lost

$$\mathbf{x} = (1.6, 0.2, 1.6, 0.2, 1.6, 0.2, 1.6, 0.2)$$

$$\mathbf{y} = (1.6, 1.6, 1.6, 1.6, 0.2, 0.2, 0.2, 0.2)$$

have the same entropy!

Extension of a source

- Given a random variable (often referred to as a *source* in information theory) X on a discrete alphabet $\mathcal{X} = \{x_1, \ldots, x_M\}$, we can define the **extension of degree** m of the source, denoted $X^{[m]}$, the source sending groups of m consecutive symbols $x_{i_1} x_{i_2} \ldots x_{i_m}$
- Exemple : if X is a source on alphabet $\{A, B\}$, $X^{[2]}$ corresponds to the source sending symbols AA, AB, BA and BB.
- The source $X^{[m]}$ has an alphabet of size $|\mathcal{X}|^m$
- Its probability distribution can be estimated by computing the probability of apparition of each subsequence of length m in the time series

Approximate entropy

The Approximate entropy [Pincus., 1991] of order m can be computed as follows:

1. Estimate the probability distribution for $X^{[m]}$ and $X^{[m+1]}$.
   - If the time series values are discrete this can be done by simple enumeration
   - If the time series values are real, use an alternate definition for the event $(X[n], X[n+1], \ldots, X[n+m-1]) = (x[n], x[n+1], \ldots, x[n+m-1])$

     $$\max_{0 \leq i \leq m} |X[n+i] - x[n+i]| \leq \epsilon$$

     where ϵ is an additional parameter
2. Compute $H\left(X^{[m]}\right)$ and $H\left(X^{[m+1]}\right)$
3. The Approximate Entropy (ApEn) is computed as

$$\text{ApEn} = H\left(X^{[m+1]}\right) - H\left(X^{[m]}\right)$$

Intuition behind ApEn

Two limit cases

- When the samples are independent and identically distributed, it is easy to prove that $H\left(X^{[m]}\right) = mH(X)$ and the Approximate Entropy is maximal:

$$\text{ApEn} = H(X)$$

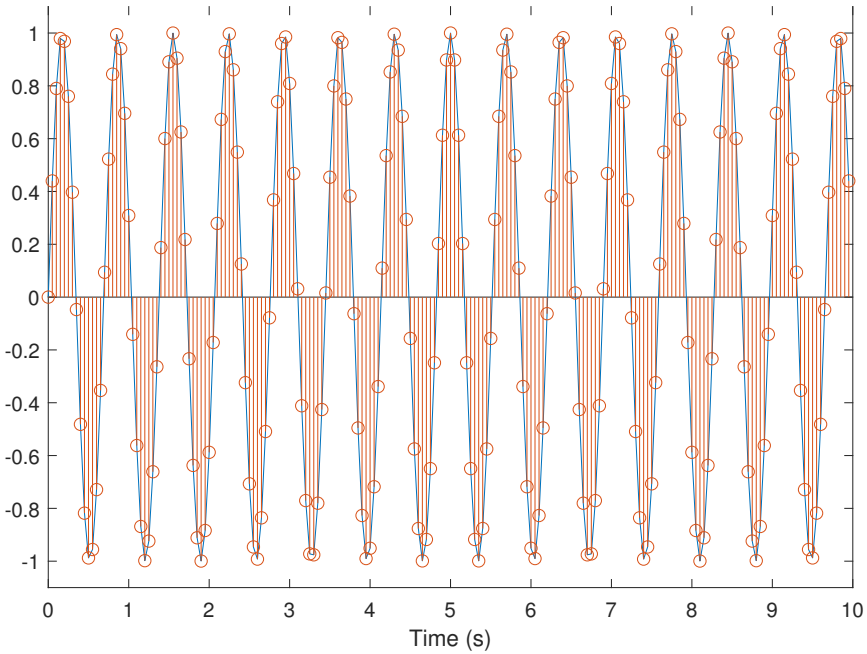
  It corresponds to the maximum chaos case: no temporal structure can be found in the time series

- When all samples are equal, the entropy is equal to zero and

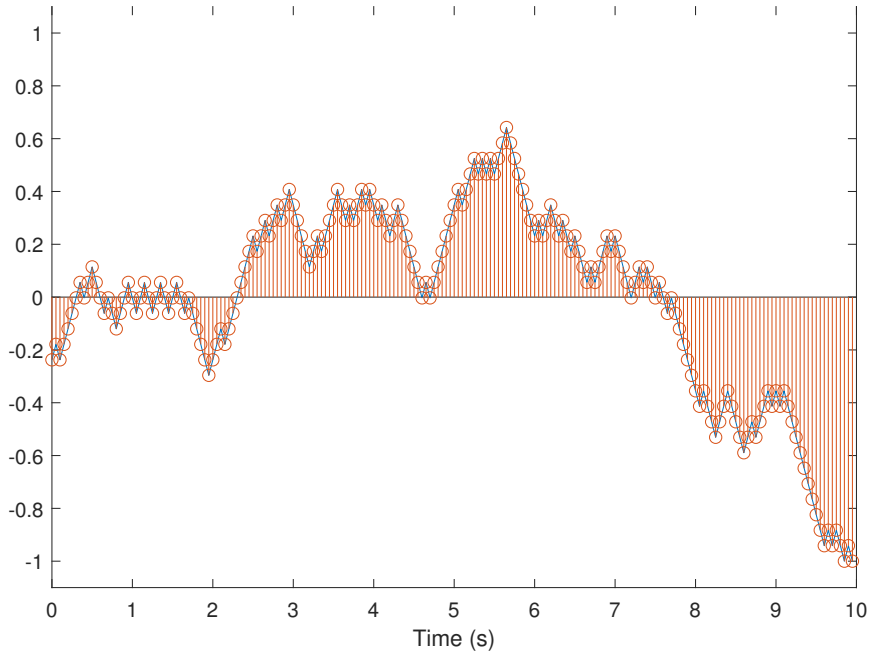
$$\text{ApEn} = 0$$

  On the contrary, there is no randomness in the data: everything is deterministic

Examples



Sinusoid
ApEn=0.2185



Random Walk
ApEn=0.4633

In general, ApEn is computed with $m = 3$ (to keep computation time limited)

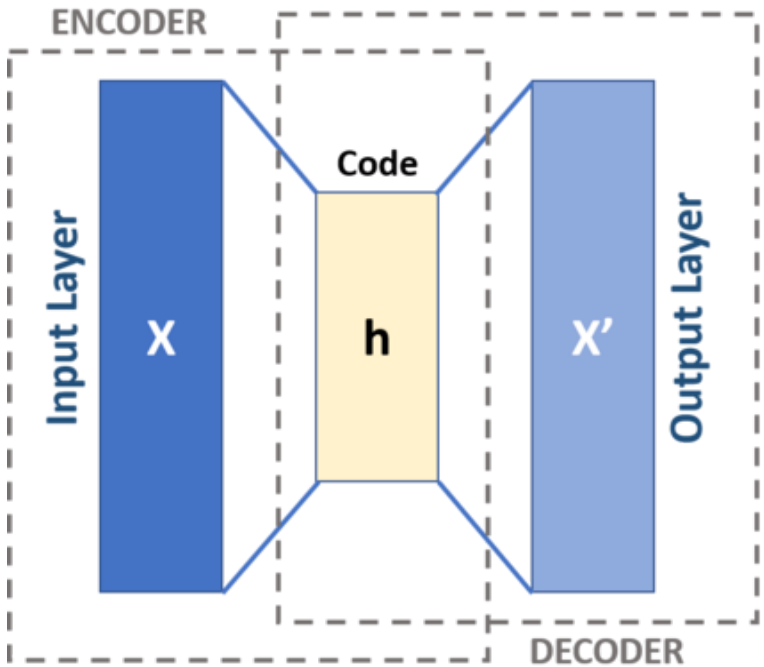
Improvements

- Several improvements of the Approximate Entropy exist, such as the Sample Entropy (see one of the proposed mini-project)
- In particular, ApEn estimates the probability of apparition of a sequence but counts each sequence as matching itself, which can introduce a bias
- Theoretical guarantees are difficult to achieve when dealing with real-valued time series, and are crucially dependent on the number of samples

Deep Learning features

- Deep Learning has emerged as a popular tool for extracting features
- Indeed, in most trained networks, the last layer stores a set of real numbers (often of limited size) that contains all relevant information for e.g. classification, prediction, etc...
- Even in an unsupervised setting, DL can be used to extract features thanks to artificial neural networks called **autoencoders** that learn efficient data codings in an unsupervised manner
- The main idea of these networks is to simultaneously learn two non-linear regression function :
  - $\phi : \mathcal{X} \to \mathbb{R}^D$ that extract the features (encoder)
  - $\psi : \mathbb{R}^D \to \mathcal{X}$ that reconstructs the time series from the features (decoder)

Autoencoders



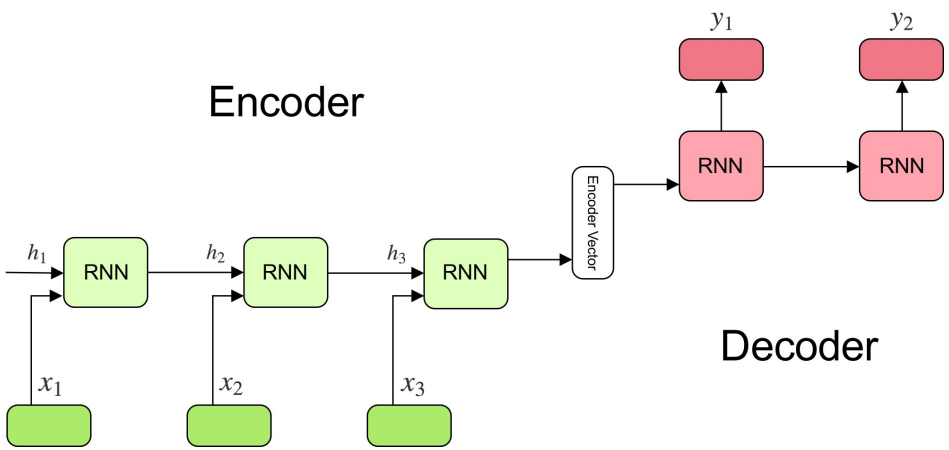
- Encoder ϕ is responsible for learning how to represent the input into lower dimensions
- Decoder ψ learns how to rebuild the smaller representations into the input again

$$\text{Aim: find } \phi, \psi = \underset{\phi,\psi}{\arg\min} \|\mathbf{X} - (\psi \circ \phi)\mathbf{X}\|^2$$

Autoencoders

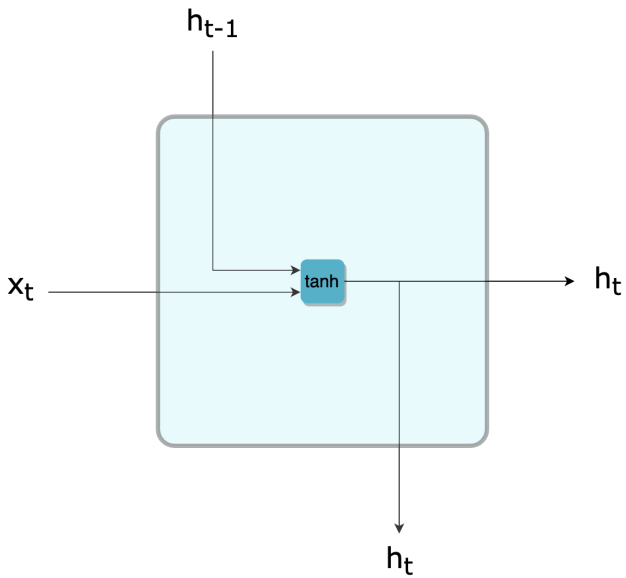
In order to work properly, autoencoders need

- Networks and cells that are able to deal with the temporal and sequential nature of the data : Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) [Chung et al., 2016], etc...
- A loss function that is able to compare the original time series and the reconstructed time series (see Lecture 1 on how to compare time series) : most of the time, the standard Euclidean distance is used (Root Mean Square Error), but recent approaches have used DTW for that [Vincent et al., 2019]

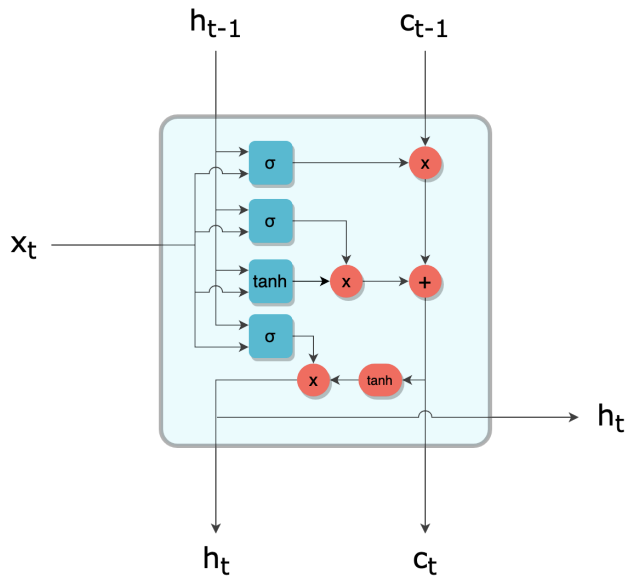


Cells for time series

- RNN networks:
  - Capable of dealing with sequences
  - Internal state **h** (of length D) for each cell
- LSTM networks [Gerts et al., 1999]:
  - Two internal states per cell : one for short memory **h**, one for long term memory **c** (both of length D)



RNN cell



LSTM cell

More info

- How to optimize the network, learn the parameters, build the layers, etc... : beyond the scope of this course
- Keep in mind that it is almost impossible to interpret the features...
- Useful resources:

https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352
https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/
https://machinelearningmastery.com/lstm-autoencoders/
https://www.kaggle.com/dimitreoliveira/time-series-forecasting-with-lstm-autoencoders

LSTM autoencoder in practice

- Two important parameters:
  - Number of time-steps N_w : window size
  - Input state size D : number of extracted features
- Number of parameters for a LSTM network:

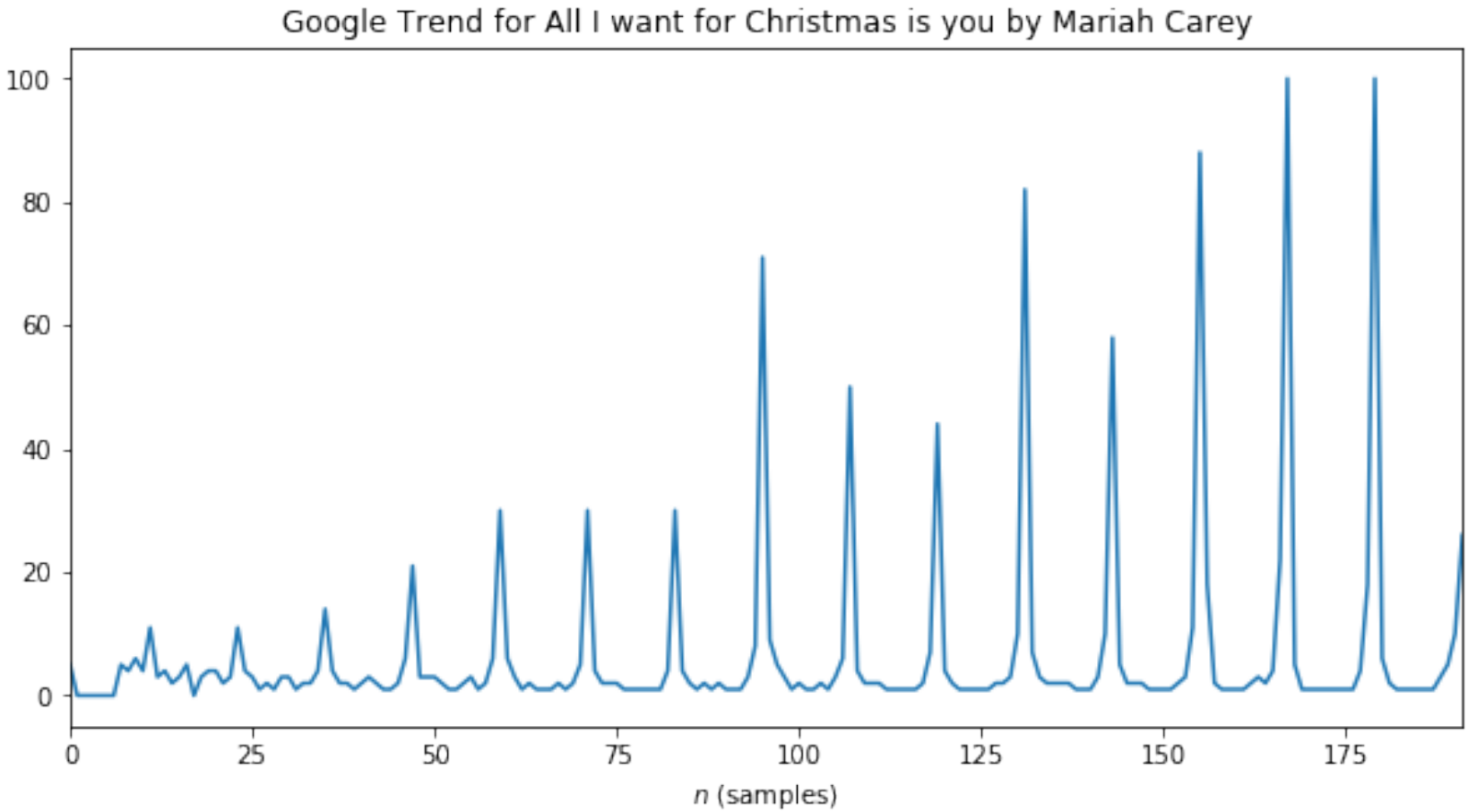
$$4(D^2 + DF + D)$$

where F is the dimensionality of a sample $x[n]$ (in case of multivariate time series).

- The input time series $\mathbf{x}$ of length N is first transformed into (possibly) overlapping frames of length N_w. These frames are stored in matrix $\mathbf{X}$
- Aim is to find ϕ, ψ

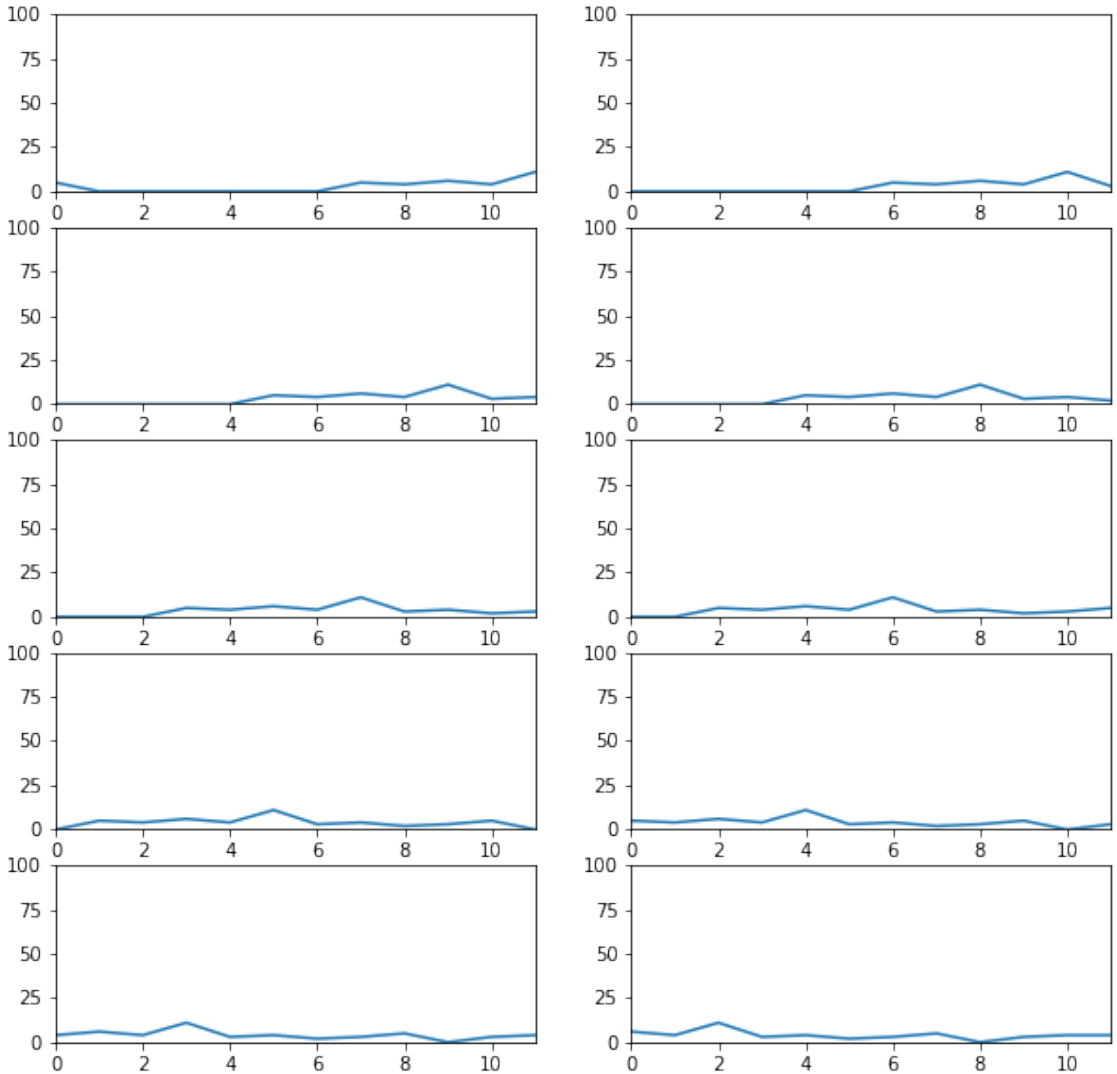
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|\mathbf{X} - (\psi \circ \phi)\mathbf{X}\|^2$$

Example



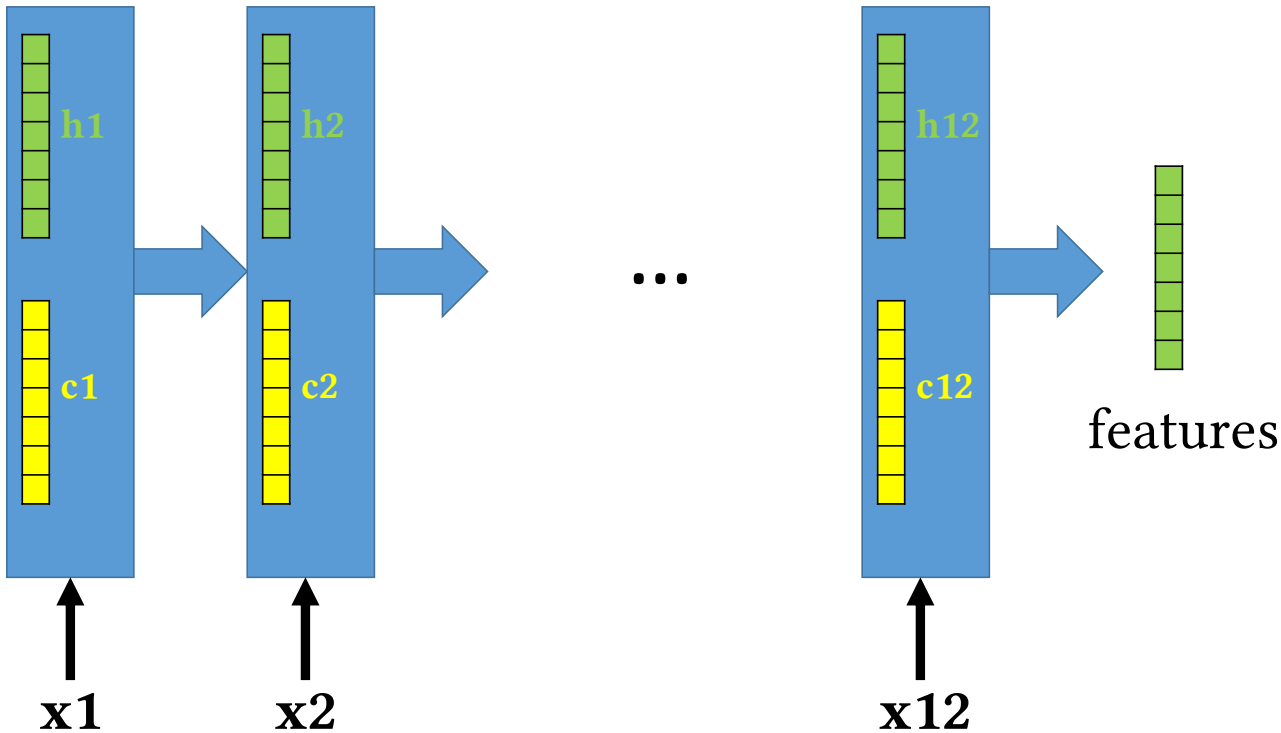
Original signal, 192 data points

Example



180 sequences of 12 samples : want to auto-encode each with only $D = 8$ parameters

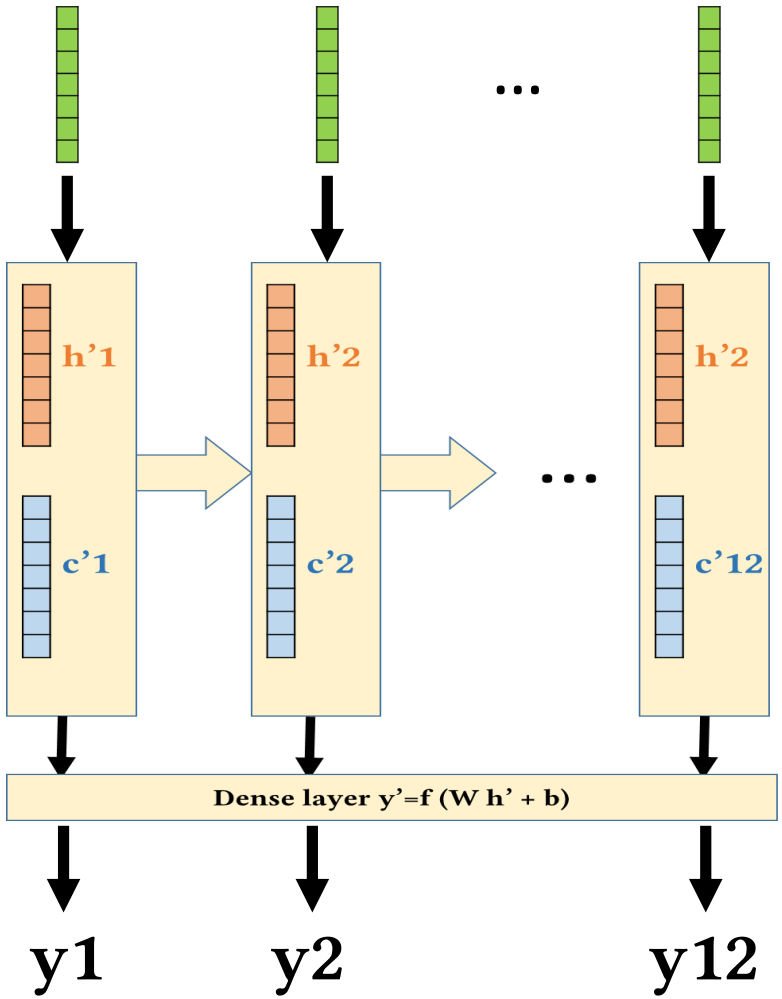
Example



Encoder ϕ with 12 cells (one per sample) and cell/hidden states with 8 parameters : 320 parameters ($D = 8$, $F = 1$)

- Input : Sequence of length $Nw = 12$
- Output : Set of features $D = 8$

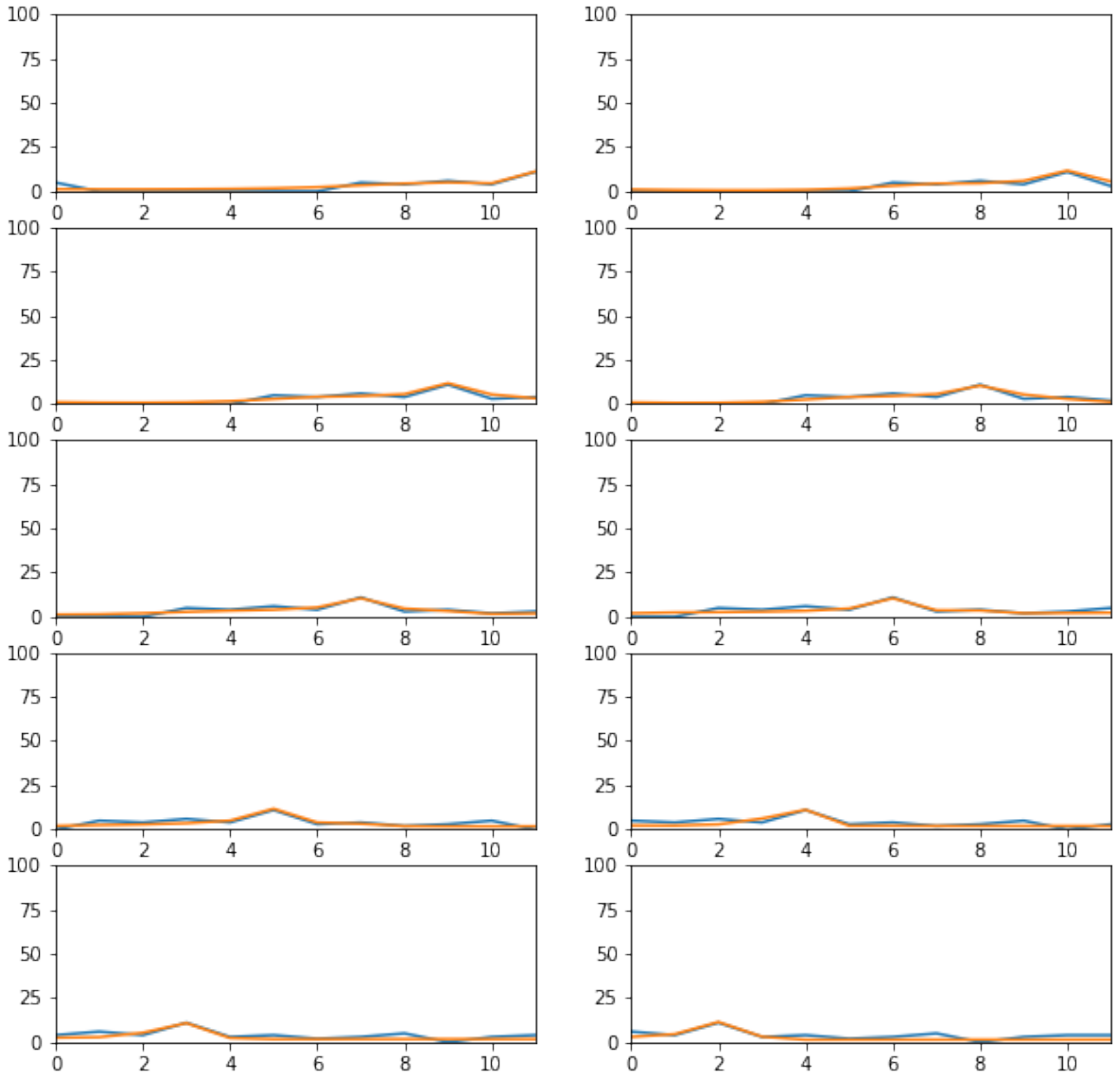
Example



Decoder ψ with 12 cells (one per sample) and cell/hidden states with 8 parameters : 544 parameters ($D = 8$, $F = 8$) + 9 parameters (dense layer)

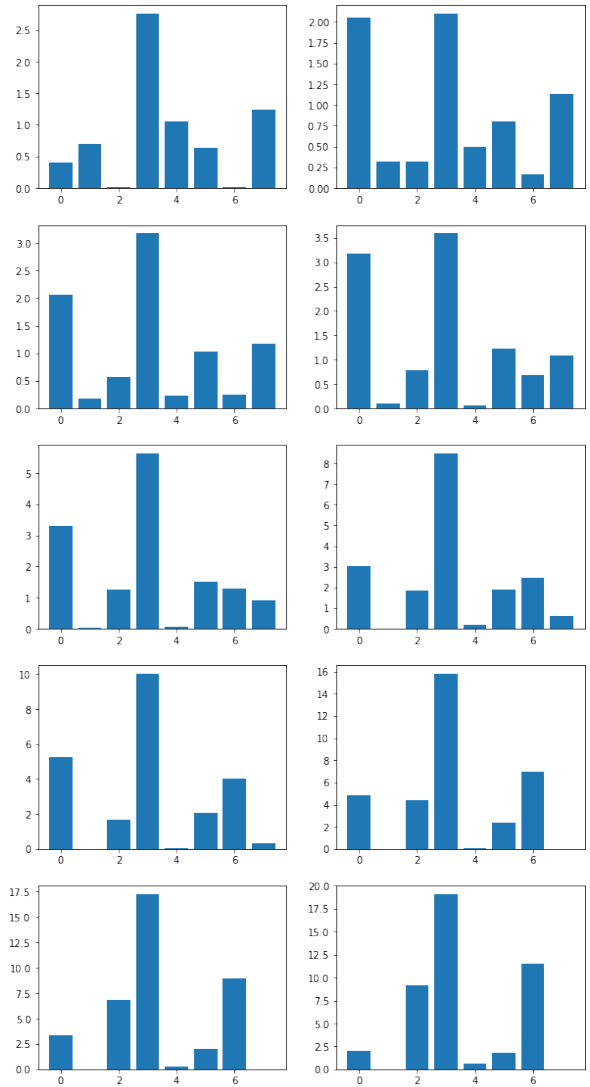
- Input : Set of features $D = 8$ (duplicated 12 times) seen as a multivariate time series ($F = 8$)
- Output : Sequence of length $Nw = 12$

Example



Reconstruction of the sequences with optimized parameters (in orange)

Example



Features extracted for each sequence

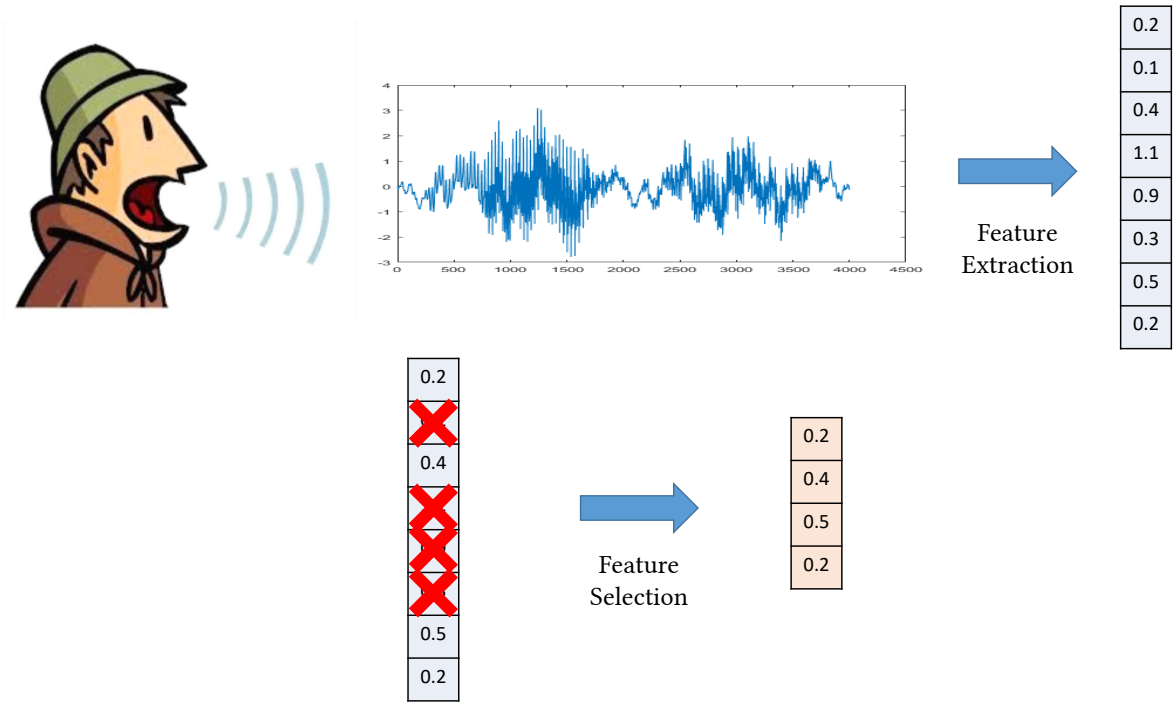
Other features

Of course, the number of features can be unlimited !

- **Topological data analysis:** notion of invariants and extraction of topological data features
- **Wavelet analysis :** wavelet coefficient (time-scale analysis)
- **Models :** model parameters can be used as features (see Lecture 3)

Contents

Problem 2: Feature Selection



- Given a dataset of M time series $\mathbf{x}$ represented each with D features, select the $K < D$ features that are relevant for a given task
- Can be performed in a supervised (labeled data) or unsupervised setting

What is a good feature?

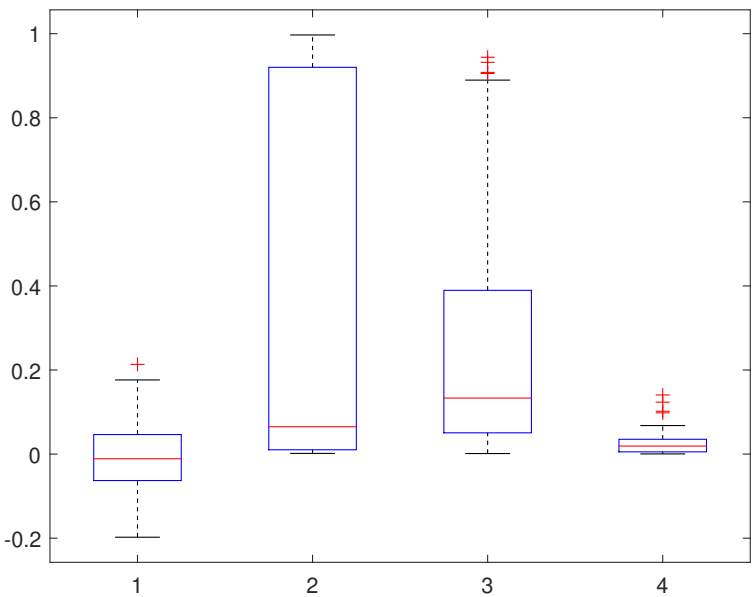
- Are they robust, well calculated and relevant?
- Do they allow to exhibit the differences present in the database?
- Do they have good generalization properties?
- Notations
  - M : number of observations
  - D : number of features
  - $\mathbf{x_1}, \ldots, \mathbf{x_M}$: observations
  - Data matrix $\mathbf{X} \in \mathbb{R}^{D \times M}$
  - Annotations (if available) $\mathbf{y} \in \{-1, +1\}^M$

Unsupervised case

Empirical observations:

- Reliability : presence of outliers?
- Utility of the features : do they allow to see differences between the observations?
- Visualization and clustering: can we distinguish groups in the database?

Reliability



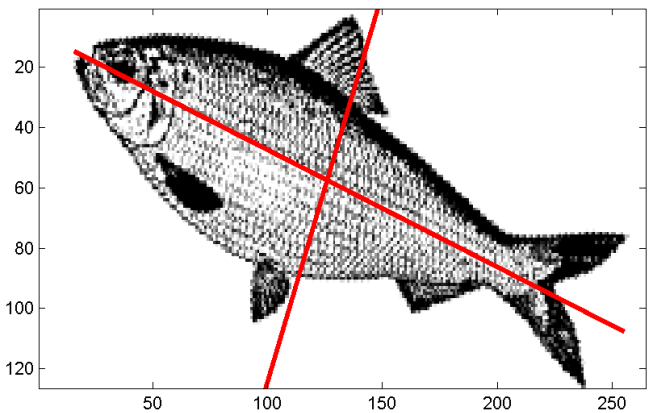
Boxplot: visualization of the quartiles

- Simple statistical *sanity checks*
  - Outlier detection by comparing the values to the standard deviation on all observations

$$X_{d,m} > 3 \times \text{std}\left(X_{:,m}\right)$$

  - Computation of 5% and 95% percentiles and comparison with minimum/maximum values
  - Simple variance test to detect features that are almost constant on the database
- The main idea is to check if extreme values are due to the properties of the observations or to any processing flaws (impulsive noise in the time series, bad feature computation, non-stationarities...): see Lecture 4 on Pre-processing

Utility of the features



Principal Component Analysis (PCA)

- Principle: detect the features that contribute the most to the global data variance
- Transform of the D initial features, possibly corrected to D new features called **principal components** that are uncorrelated
- Linear transform of the features f_d into the principal components

$$\tilde{f}_j = \sum_{d=1}^{D} U_{d,j} f_d$$

- Principal components $\tilde{f}_j$ are ordered according to their contribution to the global data variance: usually, we plot the data on the plane defined by the first two principal components

PCA for feature selection

Principal Component Analysis (PCA) to detect features that contribute the most to the data variance [Wold et al., 1987 ; Song et al., 2010]

1. Normalization of the data matrix so that each row (i.e. feature) is zero-mean and unit-variance

$$\tilde{X}_{d,:} = \frac{X_{d,:} - \mu_{X_{d,:}}}{\sigma_{X_{d,:}}}$$

2. Singular Value Decomposition (SVD) of $\tilde{\mathbf{X}}$:

$$\tilde{\mathbf{X}} = \underbrace{\mathbf{U}}_{D\times D}\underbrace{\mathbf{\Lambda}}_{D\times M}\underbrace{\mathbf{V}^t}_{M\times M}$$

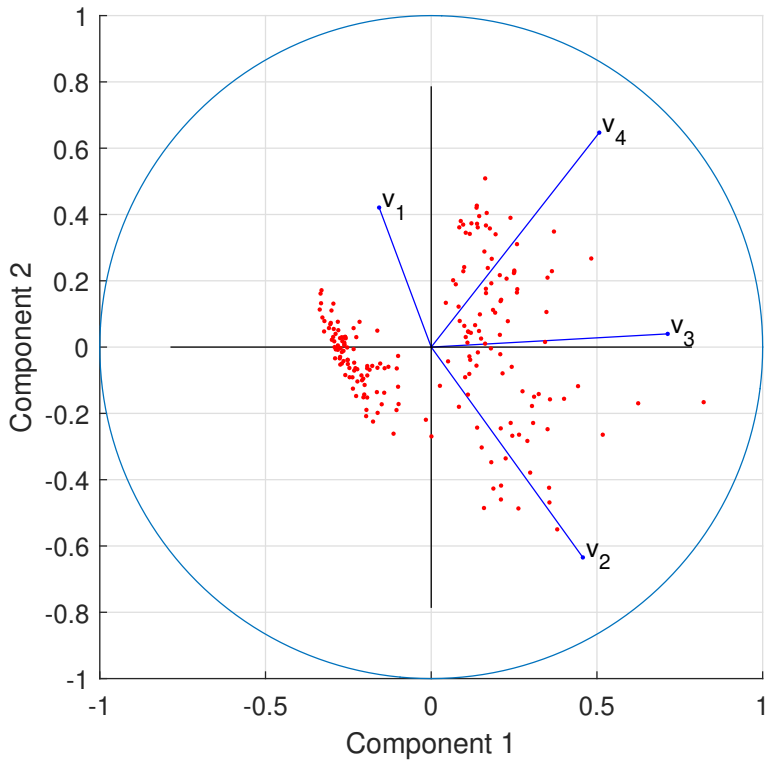
3. Plot of the contribution of each feature to the two first principal components on the *PCA correlation circle*

$$U_{d,j} : \text{ contribution of feature } d \text{ to the } j^{th} \text{ principal component}$$

4. Plot of the data points on the two first principal components. By denoting $\mathbf{S} = \mathbf{\Lambda}\mathbf{V}^t$

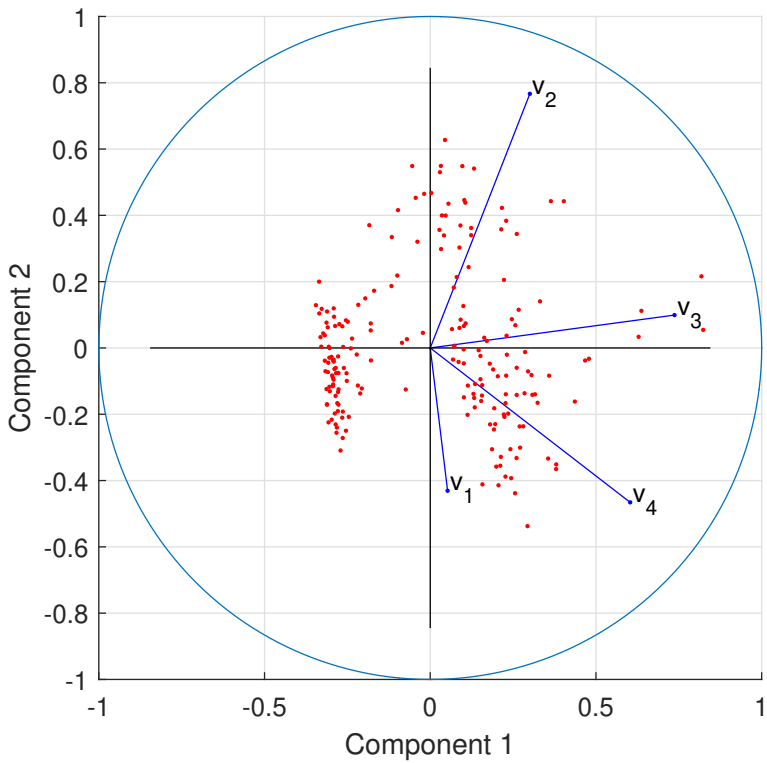
$$S_{j,m} : \text{ projection of observation } m \text{ on the } j^{th} \text{ principal component}$$

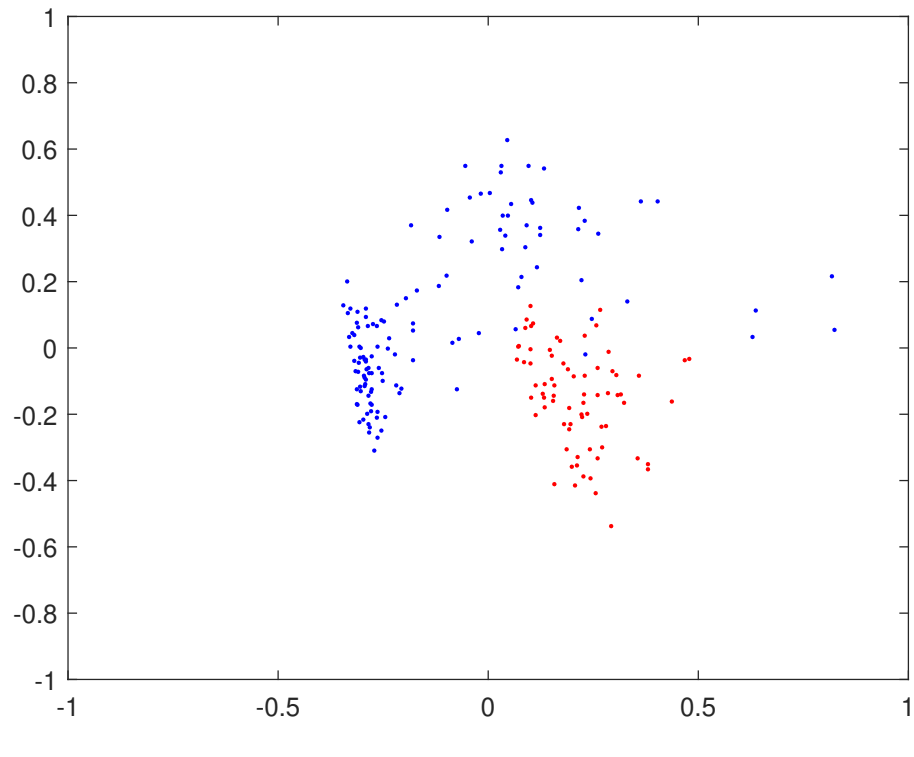
PCA visualization



- Good feature : close to the unit circle
- Allows to see redundant features and orthogonal features

Clustering?





- If we know that several classes are supposed to exist in the dataset, it is possible use a standard clustering algorithm (K-Means for instance) to visualize the obtained clusters
- Distances of the data points to the centroids can be used to assess the relevance of the feature representation

Supervised setting

- We seek to find the features that are in accordance with label information
- Three main classes of methods [Li et al., 2017] :
  - **Filter methods :** We test the adequacy of the feature with the annotations, thanks to several criteria/scores (e.g. correlation).
  - **Wrapper methods :** We test the features on a supervised classification task, by trying several combinations. The best features are kept.
  - **Embedded methods :** Mixed approaches where we jointly infer the relevance of the features and classify the data (decision tree, sparse methods...)

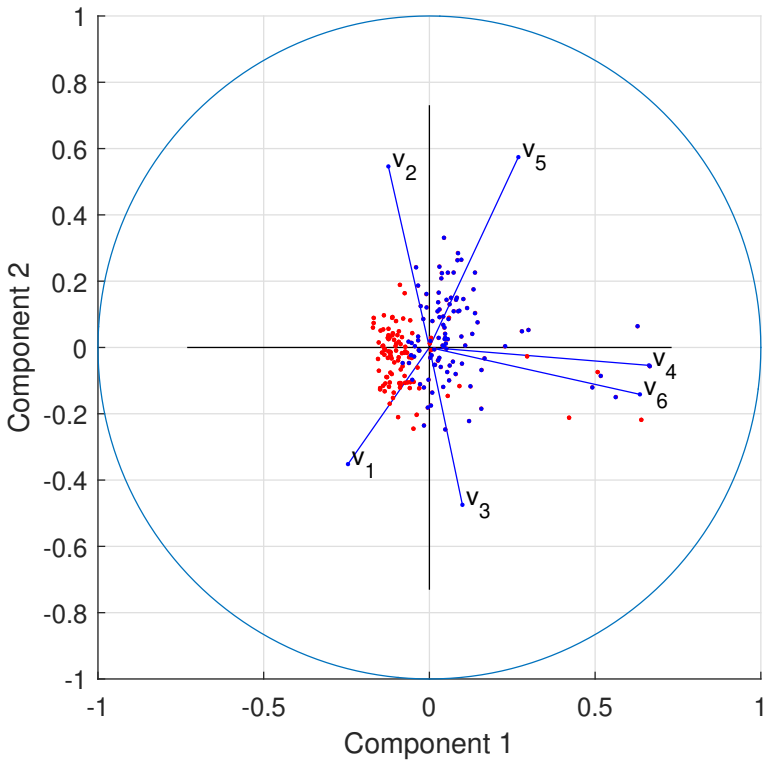
Filter methods

- We test (often individually) the relevance of each feature by studying its link with the annotations
- **Correlation score.** Computation of the Pearson correlation coefficient between the data vector $X_{d,:}$ of feature d and the annotation vector $\mathbf{y}$

$$\rho(d) = \frac{\text{cov}\left(X_{d,:}, \mathbf{y}\right)}{\sqrt{\text{var}\left(X_{d,:}\right)\text{var}\left(\mathbf{y}\right)}}$$

  - This coefficient is comprised between -1 and +1 and allows to see the linear correlation between observations and annotations
  - We keep all features for which $|\rho(d)| > \lambda$: greedy selection of the $K < D$ best features
  - Careful ! Since we select the feature one-by-one, there might exist some redundancy between highly correlated features !

Exemple



- Two classes of signals and $D = 6$ features
- According to PCA (unsupervised), features v_2, v_4, v_5, v_6 appear to be relevant
- By computing the correlation scores (supervised), we have

$$-0.02 \quad 0.08 \quad 0.32 \quad 0.45 \quad 0.82 \quad 0.23$$

Features v_4, v_5 are the most relevant for the classification task

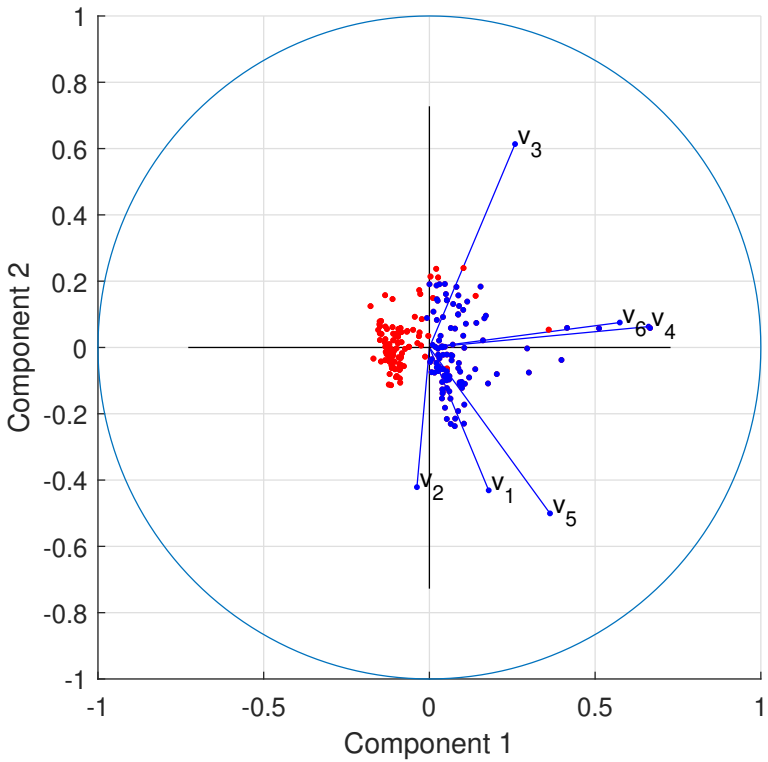
Filter methods

Fisher score. [Gu et al., 2011]

- Multiclass labels $\mathbf{y}$ with C classes
- n_c : number of data point in class c
- $\mu_{c,d}, \sigma_{c,d}$: mean/standard deviation of feature d for data point in class c
- μ_d : mean value of feature d on the whole database

$$\text{fisher\_score}(d) = \frac{\sum_{c=1}^{C} n_c \left(\mu_{c,d} - \mu_d\right)^2}{\sum_{c=1}^{C} n_c \sigma_{c,d}^2}$$

Example



- Two classes of signals and $D = 6$ features
- According to PCA (unsupervised), features v_3, v_4, v_5, v_6 appear to be relevant
- By computing the Fisher scores (supervised), we have

$$0.02 \quad 0.00 \quad 0.17 \quad 0.67 \quad 1.91 \quad 0.10$$

Features v_4, v_5 are the most relevant for the classification task

Scores based on information theory

Information theory can bring interesting scores for assessing the relevance of the features

- Assuming that the feature values have been discretized, it is possible to estimate the empirical joint and marginal probability distributions of the features and the annotations
- By using the concept of mutual information,

$$\text{information\_gain}(d) = I(X_{d,:}; \mathbf{y})$$

is a measure of dependency between the feature vector $X_{d,:}$ and the annotation vector $\mathbf{y}$. The higher this term is, the more dependent the variables are, and the more useful the feature is.

Scores based on information theory

- When selecting several features, we can extend this criterion by introducing the Minimum Redundancy Maximum Relevance (MRMR), that will both assess the fit between the feature values and the annotations, but also search for uncorrelated features

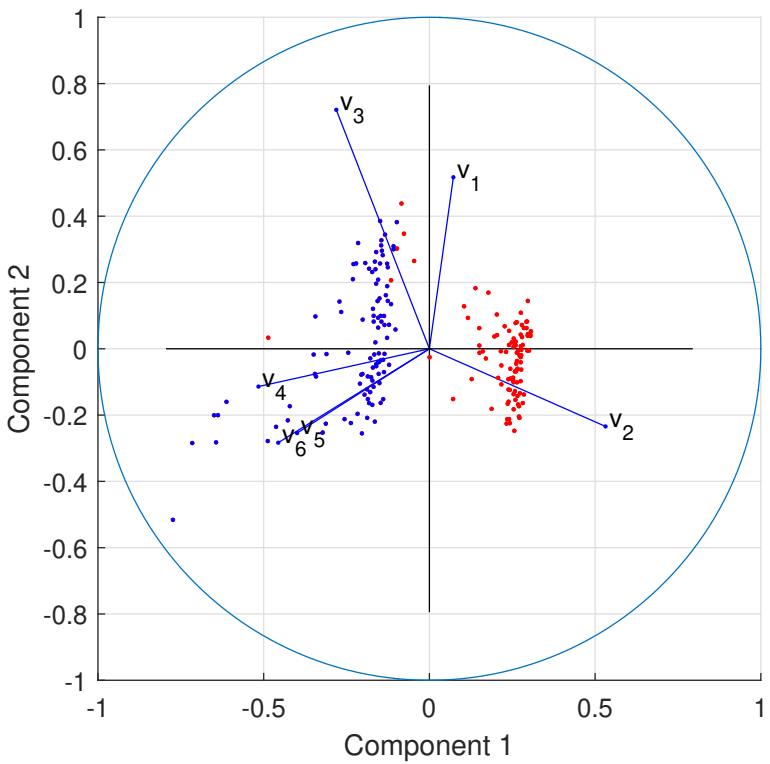
$$\text{MRMR}(d) = I(X_{d,:}; \mathbf{y}) - \frac{1}{|\mathcal{S}|} \sum_{j \in \mathcal{S}} I(X_{d,:}; X_{j,:})$$

  where $\mathcal{S}$ is the current set of selected features. The intuition is that with more non-redundant features selected, it becomes more difficult for new features to be redundant to the features that have already been in $\mathcal{S}$
- Many other criteria exist, for instance based on conditional entropies (see mini-projects)

Wrapper methods

- Test off-the-shelf classifiers (k-NN, SVM, CNN...) with different sets of features and compare the performances obtained with **cross-validation**
- Selection on the set of features giving the best performances on the training set
- If D is large, the number of possible configurations is huge (2^D-1). Several strategies can be implemented:
  - **Iterative addition of feature:** we take the best individual feature, then test all groups of two features including that one, etc...
  - **Iterative removal of feature:** we remove the features one-by-one by studying the consequences on the performances
  - **Stochastic approach:** we test random sets of features

Example



- Two classes of signals and $D = 6$ features
- Test with 1-NN (Euclidean distance in the feature space)
- One variable (Classification performances with Leave-One-Out)

$$0.49 \quad 0.97 \quad 0.64 \quad 0.88 \quad 0.94 \quad 0.7850$$

- Two variables (test of all $\binom{6}{2} = 15$ possibles configurations)

$$v_2 + v_5 : 1.0$$

$$v_2 + v_3 : 0.99$$

$$v_3 + v_5 : 0.98$$

$$\ldots$$

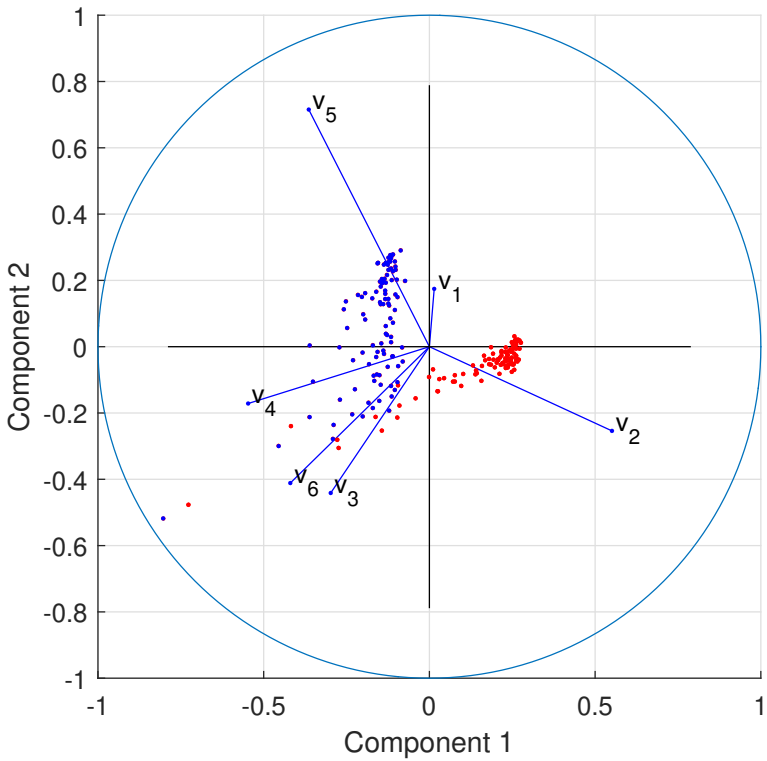
Embedded methods

- Use of regression techniques that induce some sparsity (such as L1-penalized regression or decision trees)
- **LASSO regression (Least Absolute Shrinkage and Selection Operator).** [Tibshirani, 1996]

$$\mathbf{z}^* = \underset{\mathbf{z}}{\operatorname{argmin}} \left\| \mathbf{y} - \mathbf{X}^T \mathbf{z} \right\|_2^2 + \lambda \left\| \mathbf{z} \right\|_1$$

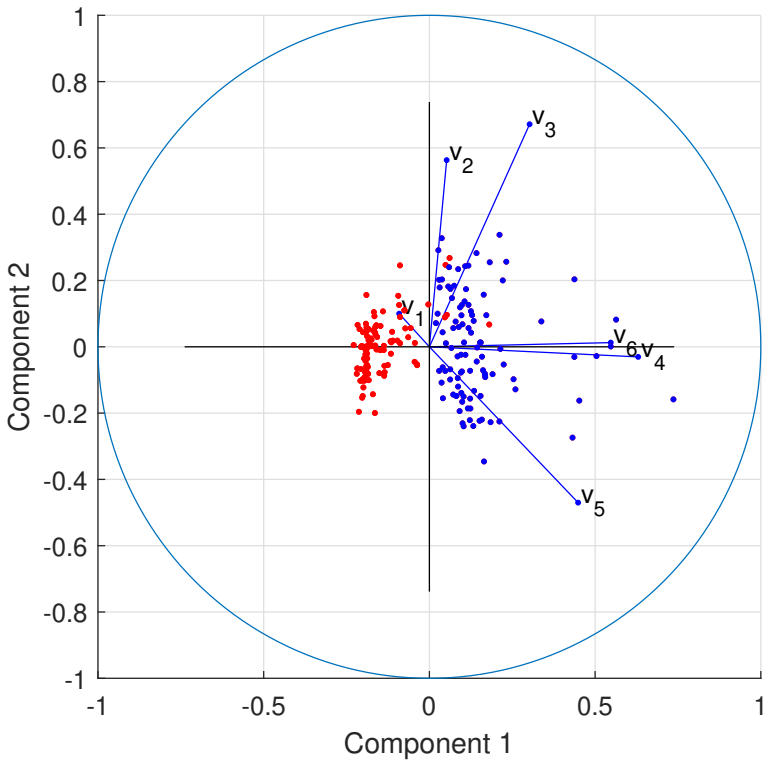
  - Sparse regression to retrieve annotations $\mathbf{y}$ from the data $\mathbf{X}$
  - Coefficients z_d reflects the importance of feature d in the regression. Thanks to the L1 sparsity constraint, lots of coefficients will be set to 0, hence providing a feature selection
  - How to solve the optimization problem: more information on Lecture 3 about Models and Representation Learning (basically proximal gradient descent)

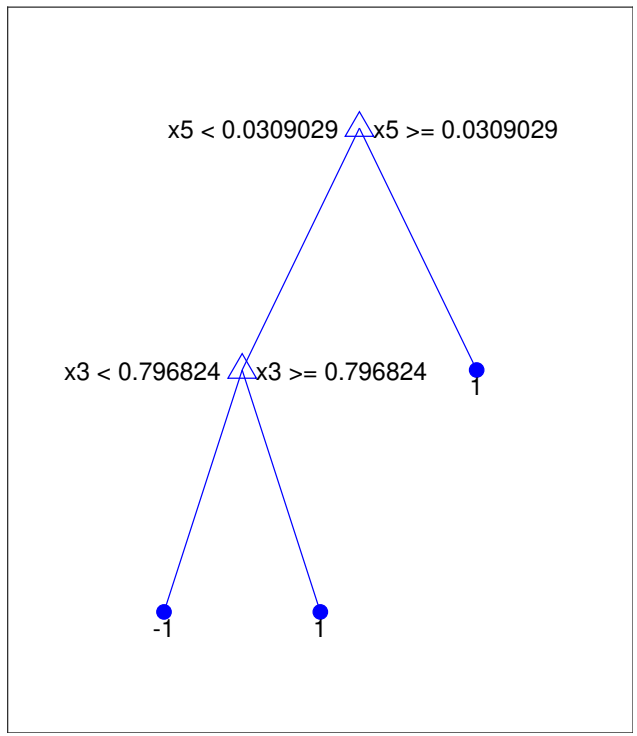
Example: LASSO



- Two classes of signals and $D = 6$ features.
- LASSO algorithm with several values for λ
  - 5 variables (1, 2, 4, 5, 6) : 0.97
  - 4 variables (2, 4, 5, 6) : 0.96
  - 3 variables (2, 5, 6) : 0.95

Example: Decision trees





Possible to only use two variables : 3 and 5

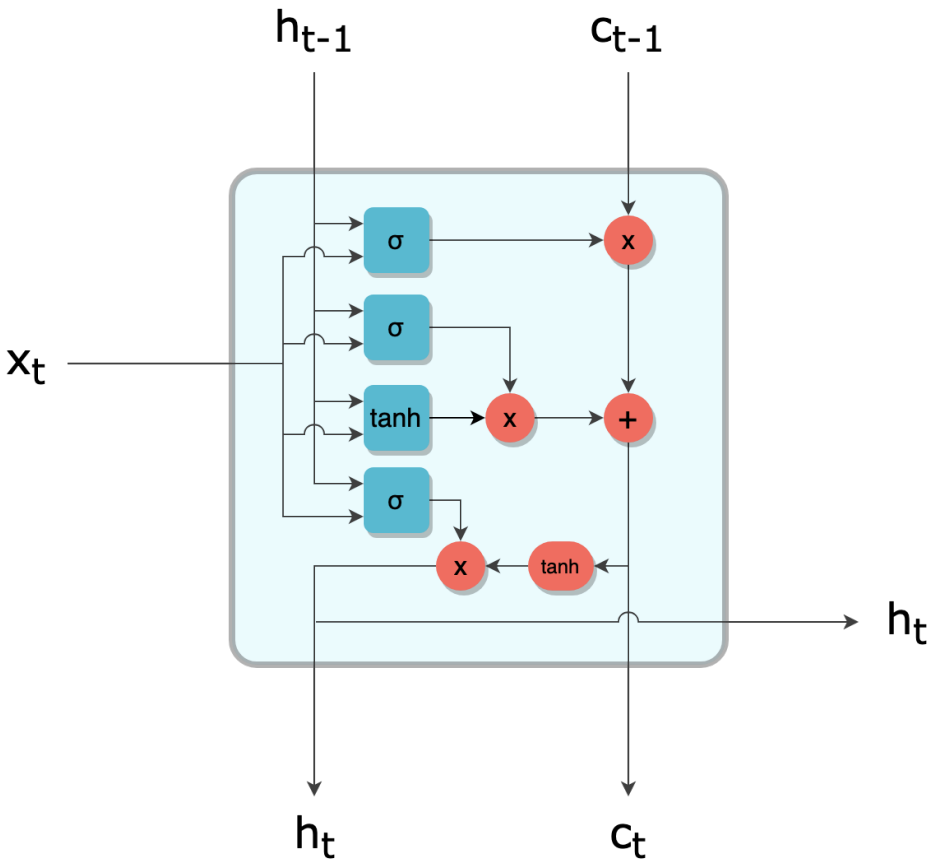
References

- Pincus, S. M. (1991). Approximate entropy as a measure of system complexity. Proceedings of the National Academy of Sciences, 88(6), 2297-2301.
- Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery, 29(6), 1505-1530.
- Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing SAX: a novel symbolic representation of time series. Data Mining and knowledge discovery, 15(2), 107-144.
- Chung, Y. A., Wu, C. C., Shen, C. H., Lee, H. Y., & Lee, L. S. (2016). Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder. arXiv preprint arXiv:1603.00982.
- Vincent, L. E., & Thome, N. (2019). Shape and Time Distortion Loss for Training Deep Time Series Forecasting Models. In Advances in Neural Information Processing Systems (pp. 4189-4201).
- Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.
- Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3), 37-52.
- Song, F., Guo, Z., & Mei, D. (2010, November). Feature selection using principal component analysis. In 2010 international conference on system science, engineering design and manufacturing informatization (Vol. 1, pp. 27-30). IEEE.
- Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2017). Feature selection: A data perspective. ACM Computing Surveys (CSUR), 50(6), 1-45.
- Gu, Q., Li, Z., & Han, J. (2011, July). Generalized Fisher score for feature selection. In Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (pp. 266-273).
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1), 267-288.

List of possible topics/projects

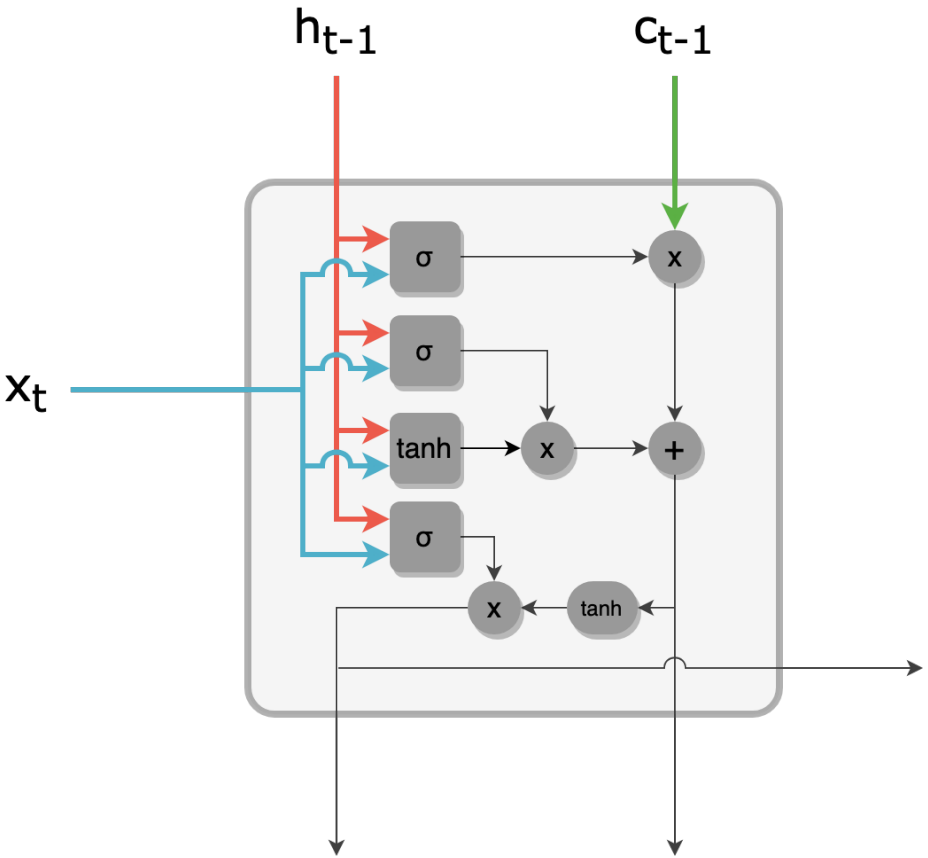
- Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery, 29(6), 1505-1530.
  How to use local symbolic features to classify time series
- Elsworth, S., & Güttel, S. (2020). Time series forecasting using LSTM networks: A symbolic approach. arXiv preprint arXiv:2003.05672.
  How to use symbolic representations for prediction
- Le Nguyen, T., Gsponer, S., Ilie, I., O'Reilly, M., & Ifrim, G. (2019). Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data mining and knowledge discovery, 33(4), 1183-1222.
  How to use multiscale symbolic representations to classify time series
- Richman, J. S., & Moorman, J. R. (2000). Physiological time-series analysis using approximate entropy and sample entropy. American Journal of Physiology-Heart and Circulatory Physiology, 278(6), H2039-H2049.
  How to extract information theory based features to study physiological time series.
- Gidea, M., & Katz, Y. (2018). Topological data analysis of financial time series: Landscapes of crashes. Physica A: Statistical Mechanics and its Applications, 491, 820-834.
  How to extract topological features to study financial data.
- Madiraju, N. S., Sadat, S. M., Fisher, D., & Karimabadi, H. (2018). Deep temporal clustering: Fully unsupervised learning of time-domain features. arXiv preprint arXiv:1802.01059.
  How to use deep learning to extract time-domain features
- He, X., Cai, D., & Niyogi, P. (2006). Laplacian score for feature selection. In Advances in neural information processing systems (pp. 507-514).
  How to apply unsupervised feature selection.
- Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2017). Feature selection: A data perspective. ACM Computing Surveys (CSUR), 50(6), 1-45.
  How to apply a large number of feature selection methods (multitude of topics in this article including information theory!)
- Barandas, M., Folgado, D., Fernandes, L., Santos, S., Abreu, M., Bota, P., ... & Gamboa, H. (2020). TSFEL: Time series feature extraction library. SoftwareX, 11, 100456.
- Längkvist, M., Karlsson, L., & Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters, 42, 11-24.

LSTM cell

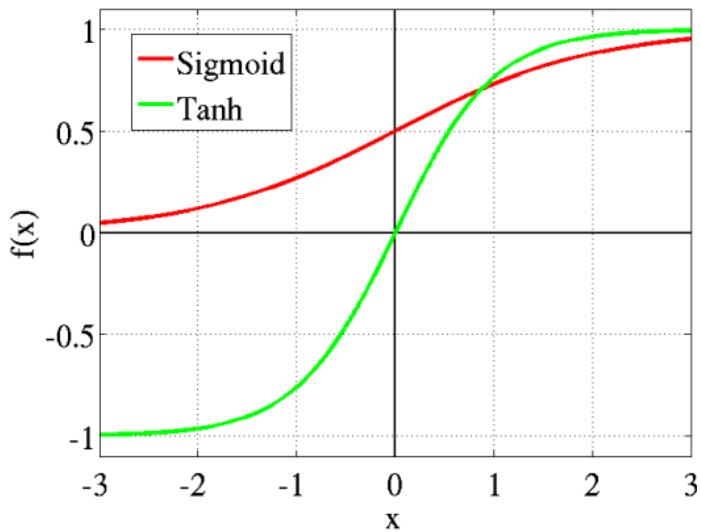


- Inputs:
  - x_t: signal value at time t (scalar)
  - $\mathbf{h}_{t-1}$: hidden state of previous cell (vector of length D : number of features)
  - $\mathbf{c}_{t-1}$: cell state of previous cell (vector of length D)
- Outputs:
  - $\mathbf{c}_t$: cell state of current cell (vector of length D)
  - $\mathbf{h}_t$: hidden state of current cell (vector of length D)

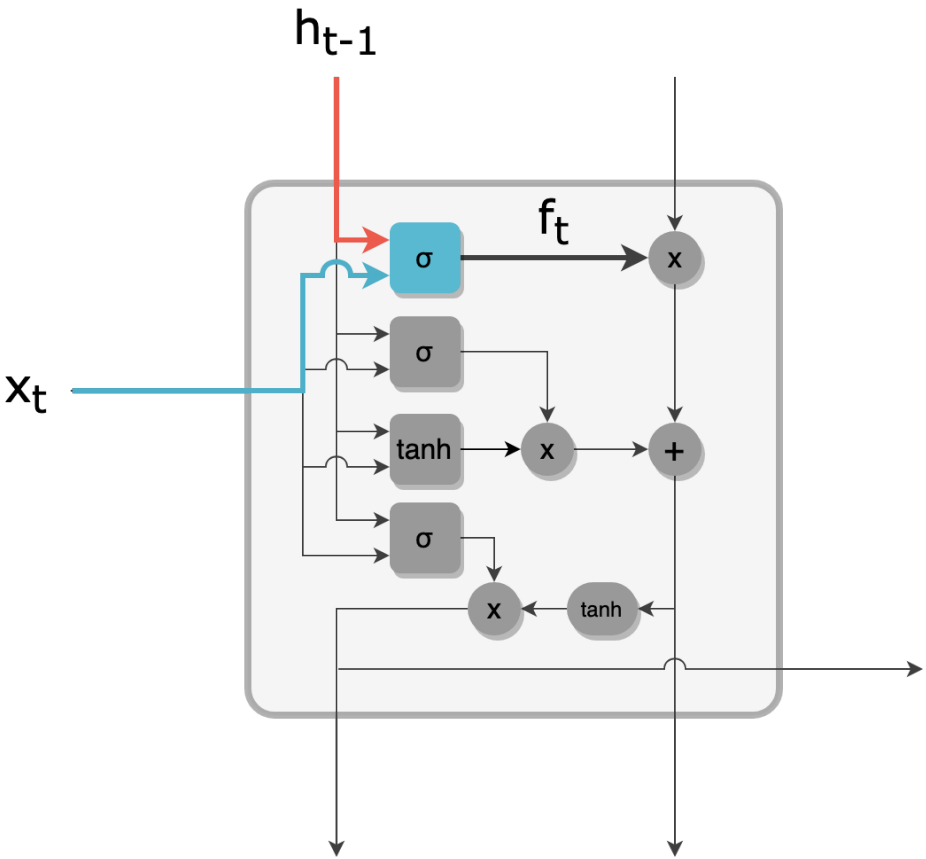
LSTM cell



- $\mathbf{h}_t$ is the **hidden state** and captures the short-time memory of the cell
- $\mathbf{c}_t$ is the **cell state** and captures the long-time memory of the cell



LSTM cell

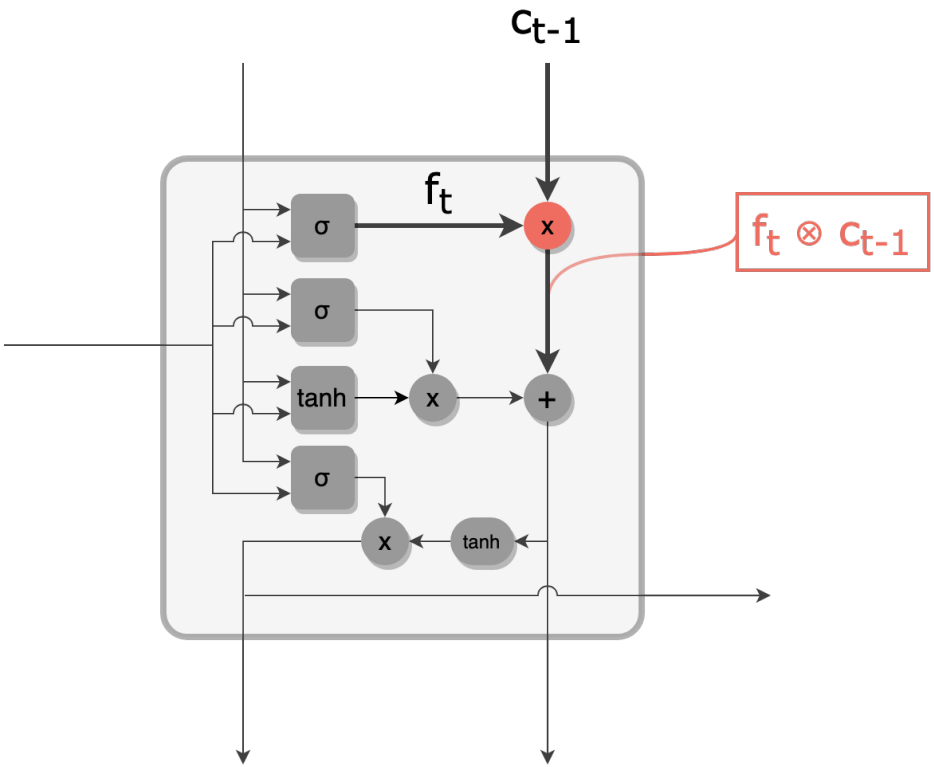


Forget gate

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_\mathbf{f}[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_\mathbf{f}\right)$$

- Vector of length D with values between 0 and 1
- Used as a filter to *forget* the some of the previous cell state values

LSTM cell

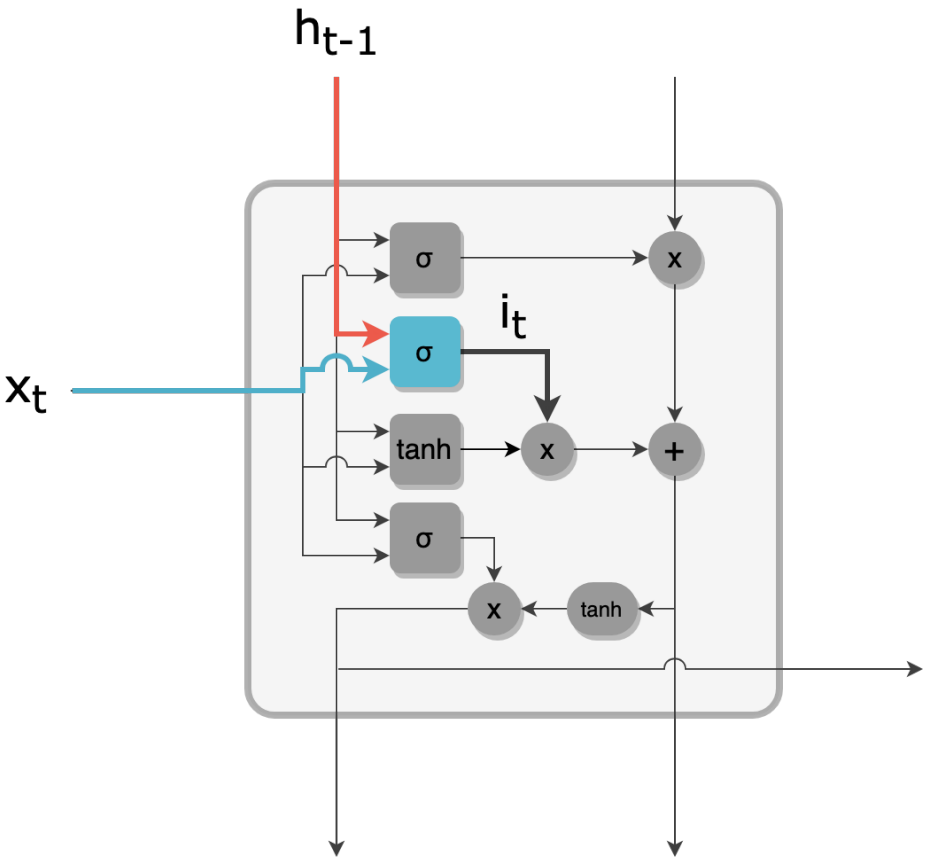


Forget gate

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_\mathbf{f}[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_\mathbf{f}\right)$$

- Vector of length D with values between 0 and 1
- Used as a filter to *forget* the some of the previous cell state values

LSTM cell

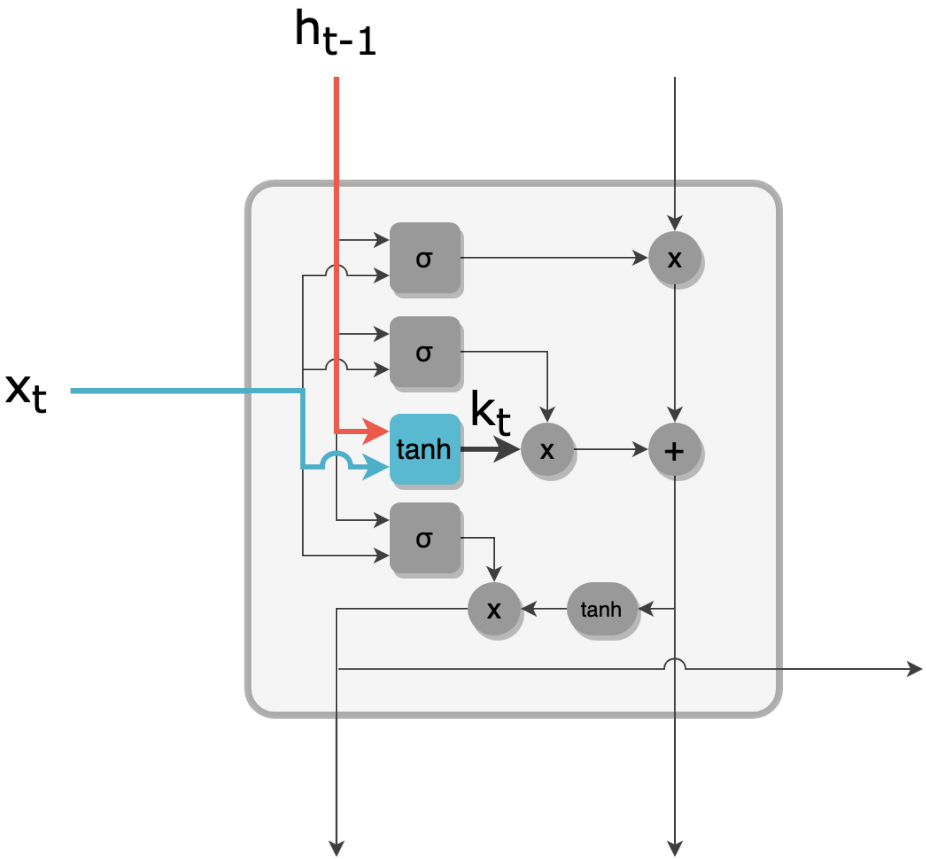


Input gate

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_\mathbf{i}[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_\mathbf{i}\right)$$

- Vector of length D with values between 0 and 1
- Similar than the forget gate

LSTM cell

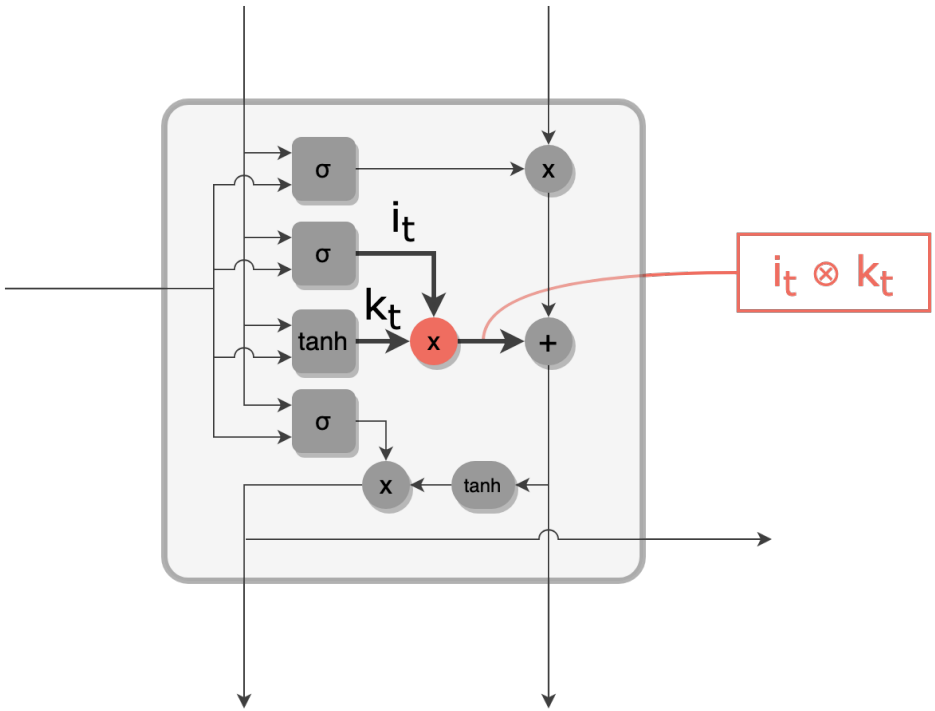


Input modulation gate

$$\mathbf{k}_t = \tanh\left(\mathbf{W}_\mathbf{k}[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_\mathbf{k}\right)$$

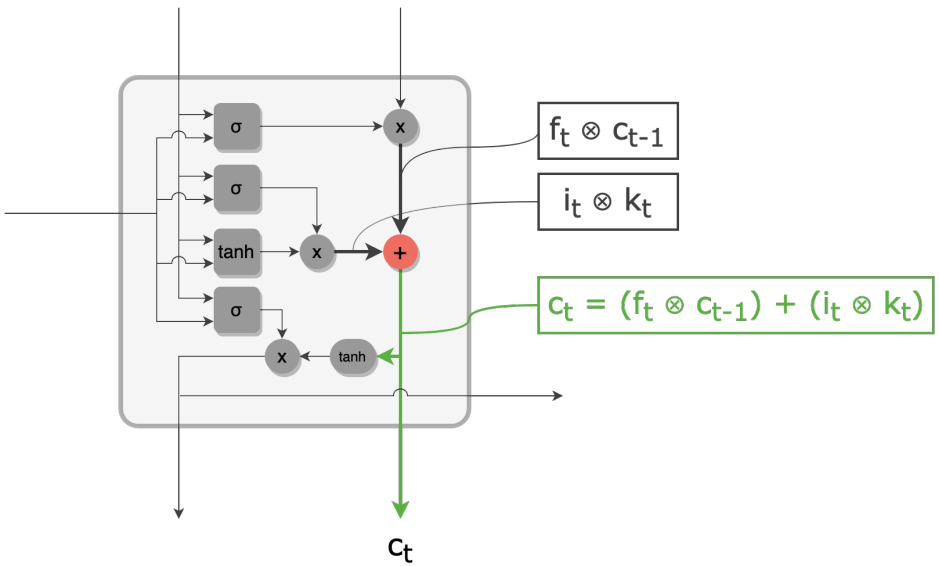
- Vector of length D
- Candidate for the new cell state

LSTM cell



Candidate $\mathbf{k}_t$ is filtered with $\mathbf{i}_t$ to form a filtered candidate

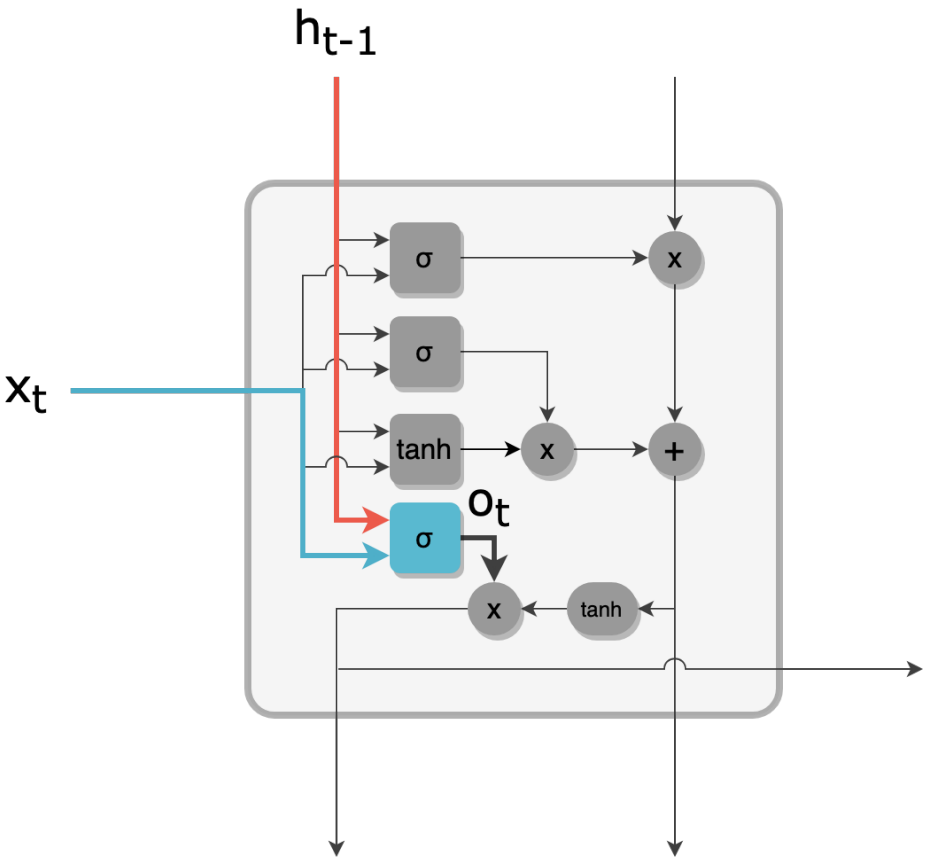
LSTM cell



The new cell state is the sum of:

- The filtered previous cell state (where useless information has been removed)
- The filtered candidate cell state

LSTM cell

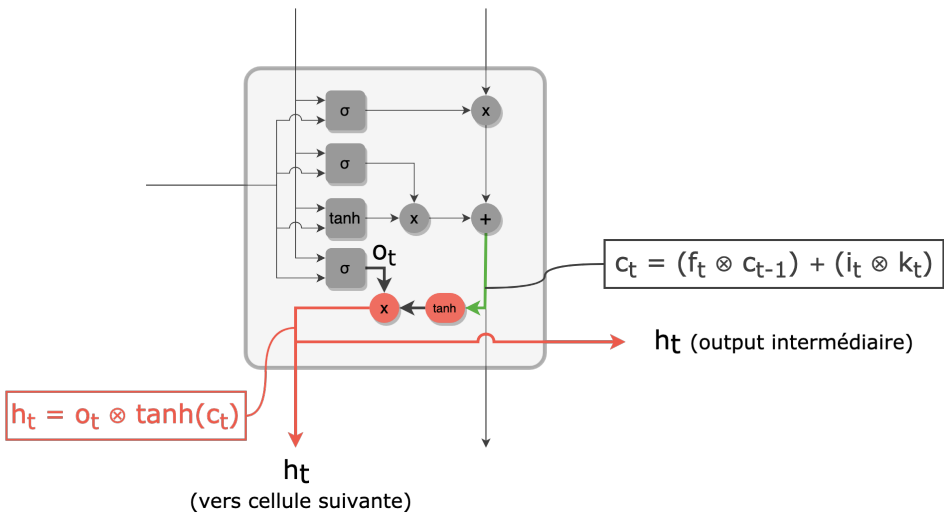


Output gate

$$\mathbf{o}_t = \sigma\left(\mathbf{W_o}[\mathbf{h}_{t-1}, x_t] + \mathbf{b_o}\right)$$

- Vector of length D with values between 0 and 1
- Similar than the forget gate and the input gate

LSTM cell



The new hidden state is the product of the:

- The output state
- The current cell state renormalized in $]-1,+1[$